A NEW TEST SYSTEM FOR ABINIT

BOTTOM-UP APPROACH ON STRUCTURED DATA

T. CAVIGNAC

ÉCOLE POLYTECHNIQUE DE LOUVAIN, UNIVERSITÉ CATHOLIQUE DE LOUVAIN-LA-NEUVE

ÉCOLE CENTRALE DE LYON

ABINIT DEVELOPER WORKSHOP, MAI 2019
Why do we test?

We have to test to

- Find bugs
- Grant quality of the physical results
- Prevent breaking old features working on new ones
1 The need of a new comparison method

2 The solution proposed

3 What is coming next
THE NEED OF A NEW COMPARISON METHOD
Characteristics of the current comparison method

- Linear comparison of lines
- Extracting of every floating point and individual comparison
- One tolerance, used as absolute and relative, for the whole test
- Auxiliaries tolerances used if the main one is not respected
Strength:

- Systematic/comprehensive top-down approach
- Strict by design
- Does not require specific format of the output, except for the first character of the line
- Just work™

Flaws:

- Linear analysis fail if the number of significant line differ
- Unaware of physics
- Hardly any extension possibilities
- Very rigid configuration leads to weakening the whole test when a few lines are hard to get right
Figure: Repartition of the main fldiff tolerances in the pool of tests
Repartition of the auxiliary tolerances in the pool of tests:
(left) $\log_{10}$ of absolute tolerance, peak between -5 and -2
(right) $\log_{10}$ of relative tolerance, peak between -1.5 and 0
**Figure:** "Quality" of tests according to their tolerances (left: all, top right: v3 only, bottom right: v8 only)
The solution proposed
Based on structured data in the form of YAML documents embedded in the main output file
- YAML documents produced by Fortran
- Bottom-up approach
- Configured with a separate file also written in YAML
- Aware of the "iteration state"
- Testing side written in Python and integrated with the existing testsuite
- Integration of Numpy and Pandas
**Strength:**

- Great flexibility
- Open lots of new possibilities
- Backward compatible: YAML documents can be ignored and the test bot will behave as it did before
- Allow physics aware analysis
- Matching of tester and reference documents is done through label and iteration state

**Flaws:**

- Ask for more configuration when enabled
- Have to be configured for each test and each physical quantity
- Brand new, need real world testing

The two methods are complementary and will be used together.
Figure: Example of a YAML document in ABINIT output
Two level of API:
- `m_neat`: high-level API, should be called in computations routines
- `m_yaml_out`: low-level API, actually produce YAML documents, supposed to be called only from `m_neat`.

Additional toolboxes:
- `m_stream_string`: variable-size string type, can be used as a buffer to build a YAML document
- `m_pair_list`: structure to store key-value pairs, keys are strings and values integers, real numbers or strings.
1. Use m_pair_list to store values as the computation go on:
call pl%set("Etot", r=etot_val)

2. Pass data to a m_neat routine you wrote before

3. It will call m_yaml_out routines to build a document and use stream_wrtout to output it
   call yaml_single_dict("Etot", "", pl, 30, 100, stream=mydoc)
call stream_wrtout(mydoc, iout)
Input file TEST_INFO section and YAML test configuration

- structures.py
- conf_parser.py
Actual test configuration belongs here. Define the rules for each piece of data and the logic of the test.

Figure: An example of YAML configuration file
YAML provides facilities to have specialized logic for some data structures.

Figure: Example of a structure definition
Here are defined the rules used in YAML configuration file. The actual comparison functions (constraints) belong here as well as their parameters declarations.

```python
@conf_parser.constraint(exclude=['tol', 'ceil', 'ignore'])
def tol_abs(tol, ref, tested):
    ...
    Valid if the absolute difference between the values is below the given tolerance.
    ...
    return abs(ref - tested) < tol
```

**Figure:** Example of constraint definition
What is coming next
We need you!

How you can help:

- Read the documentation (located at ~abinit/doc/developers/new_testsuite.md), give us feedback on it
- Add YAML testing to your old tests
- Use YAML testing in your new tests
Parameterized tests
Test starting from precomputed binaries
Strongly noisy tests giving stable processed quantities
New processing in test (linear regression, statistics, simpler consistency tests...)
I want to express my gratitude to Jean-Michel Beuken for his help with the test farm and the building process of ABINIT. I also want to thank Matteo Giantomassi and Xavier Gonze for giving me the possibility of doing this work and for their supervision.
Thank you for your attention!
Questions ?