National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Image comparison (and GUI functionality) testing

…in Python, for cf-plot (and cf-view)

Sadie Bartholomew
CMS meeting, 2024-11-29

We have a plotting package (cf-plot) and a GUI package (cf-view). Neither had any testing (eep!). How do we test their *functionality* is as we intend/document?

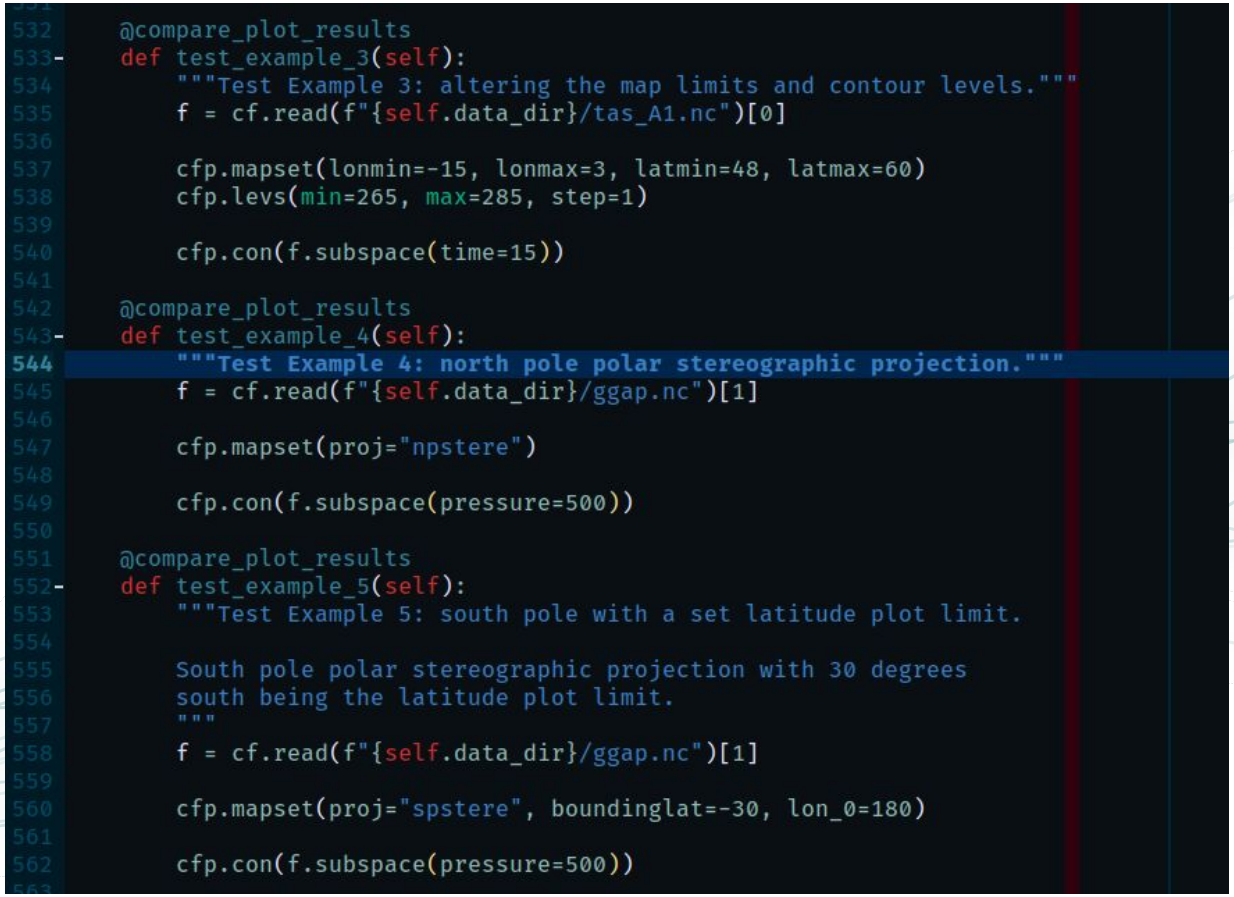
The code we expect to work (and document for the API and as working examples, etc.) should certainly run without error, but just as importantly it should do what we expect visually and functionally - for example produce a correct plot with the right configuration (for cf-plot), or invoke the intended behaviour and without any unintended consequences alongside that (for cf-view).

RE https://github.com/NCAS-CMS/cf-plot/issues/12, https://github.com/NCAS-CMS/cf-plot/issues/13

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

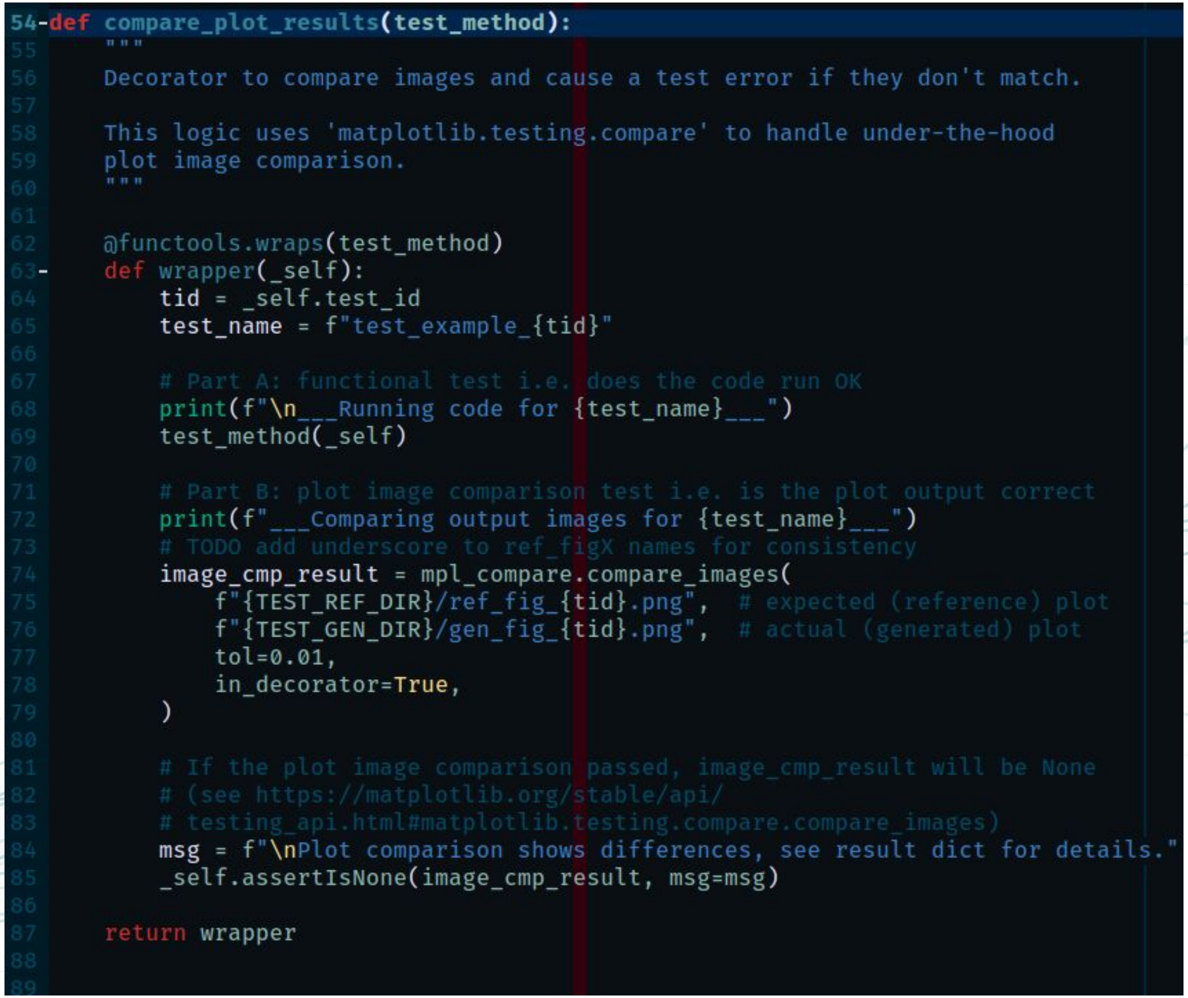
1. Ensuring plots are as expected with image comparison testing (for cf-plot)

Re-using matplotlib capability to decorate test methods



```
    @compare_plot_results
    def test_example_3(self):
        """Test Example 3: altering the map limits and contour levels."""
        f = cf.read(f"{self.data_dir}/tas_A1.nc")[0]

        cfp.mapset(lonmin=-15, lonmax=3, latmin=48, latmax=60)
        cfp.levs(min=265, max=285, step=1)

        cfp.con(f.subspace(time=15))

    @compare_plot_results
    def test_example_4(self):
        """Test Example 4: north pole polar stereographic projection."""
        f = cf.read(f"{self.data_dir}/ggap.nc")[1]

        cfp.mapset(proj="npstere")

        cfp.con(f.subspace(pressure=500))

    @compare_plot_results
    def test_example_5(self):
        """Test Example 5: south pole with a set latitude plot limit.

        South pole polar stereographic projection with 30 degrees
        south being the latitude plot limit.
        """
        f = cf.read(f"{self.data_dir}/ggap.nc")[1]

        cfp.mapset(proj="spstere", boundinglat=-30, lon_0=180)

        cfp.con(f.subspace(pressure=500))
```

- The basic test suite has a standard structure with test methods defined testing specific functionality (named `test_example_N`). Ideally we can use those untouched to check the code runs, but have a way to compare the plot output with *an expected pre-generated image*
- Use a decorator to do so - I've called it `compare_plot_results`.



```python
def compare_plot_results(test_method):
    """
    Decorator to compare images and cause a test error if they don't match.

    This logic uses 'matplotlib.testing.compare' to handle under-the-hood
    plot image comparison.
    """

    @functools.wraps(test_method)
    def wrapper(_self):
        tid = _self.test_id
        test_name = f"test_example_{tid}"

        # Part A: functional test i.e. does the code run OK
        print(f"\n___Running code for {test_name}___")
        test_method(_self)

        # Part B: plot image comparison test i.e. is the plot output correct
        print(f"___Comparing output images for {test_name}___")
        # TODO add underscore to ref_figX names for consistency
        image_cmp_result = mpl_compare.compare_images(
            f"{TEST_REF_DIR}/ref_fig_{tid}.png",  # expected (reference) plot
            f"{TEST_GEN_DIR}/gen_fig_{tid}.png",  # actual (generated) plot
            tol=0.01,
            in_decorator=True,
        )

        # If the plot image comparison passed, image_cmp_result will be None
        # (see https://matplotlib.org/stable/api/
        # testing_api.html#matplotlib.testing.compare.compare_images)
        msg = f"\nPlot comparison shows differences, see result dict for details."
        _self.assertIsNone(image_cmp_result, msg=msg)

    return wrapper
```

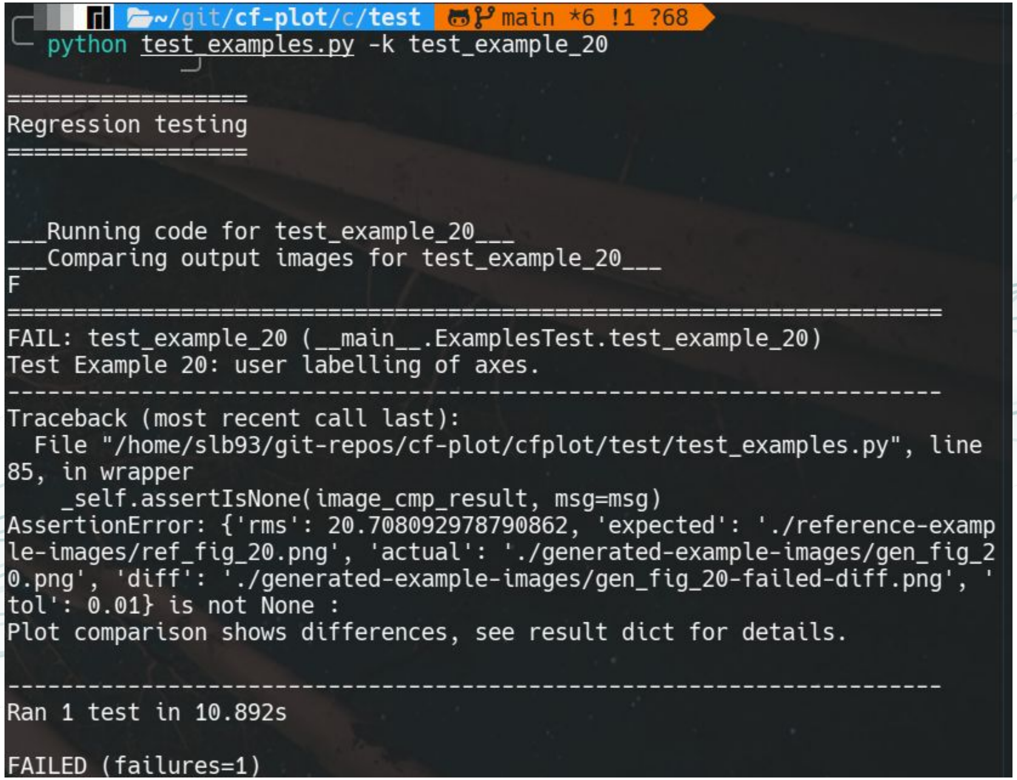
- `matplotlib.testing.compare` has dedicated utilities for comparing image results used to test matplotlib itself. Best re-use those and not write our own!
- They provide their own decorator but in order to separate out images to compare into separate named directories, I wrote my own making use of their `compare_images`* function.
- Expected images for the output plots are pre-saved in one directory and generated plots are put into a separate one

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

*https://matplotlib.org/stable/api/testing_api.html#matplotlib.testing.compare.compare_images

Tests pass/fail/error on code *and* on image comparison

Example: running a single test method (`test_example_20`) which here is hacked (via adding an unexpected parameter) to fail the image comparison check despite passing for code running fine.



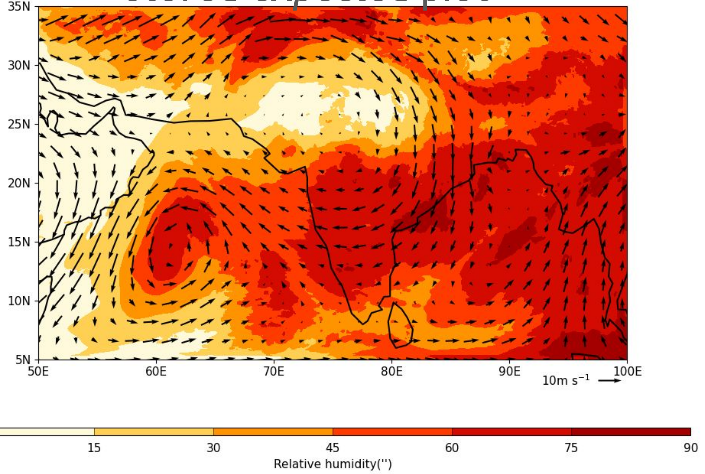
```
~/git/cf-plot/c/test   main *6 !1 ?68
python test_examples.py -k test_example_20

==================
Regression testing
==================


___Running code for test_example_20___
___Comparing output images for test_example_20___
F
======================================================================
FAIL: test_example_20 (__main__.ExamplesTest.test_example_20)
Test Example 20: user labelling of axes.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/slb93/git-repos/cf-plot/cfplot/test/test_examples.py", line
85, in wrapper
    _self.assertIsNone(image_cmp_result, msg=msg)
AssertionError: {'rms': 20.708092978790862, 'expected': './reference-examp
le-images/ref_fig_20.png', 'actual': './generated-example-images/gen_fig_2
0.png', 'diff': './generated-example-images/gen_fig_20-failed-diff.png', '
tol': 0.01} is not None :
Plot comparison shows differences, see result dict for details.

----------------------------------------------------------------------
Ran 1 test in 10.892s

FAILED (failures=1)
```

- Particularly useful is that when there is a failure on image comparison, we get a dictionary quantifying the difference and a 'diff' image showing where in the image differences were detected

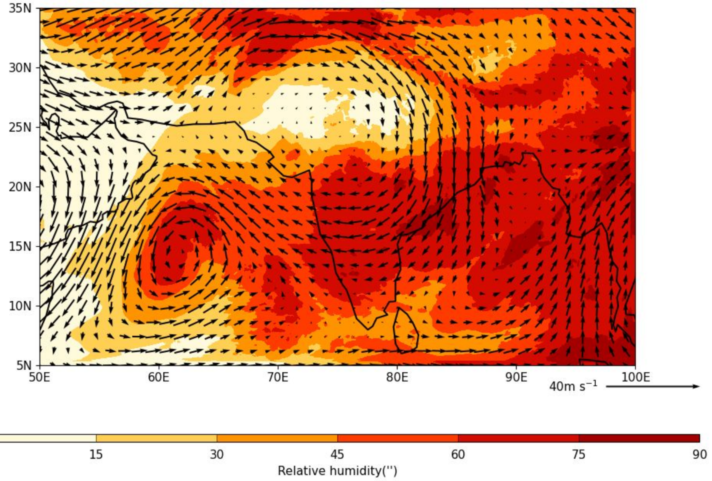
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Image-comparison testing: example 1 (artificially made)

Stored *expected* plot



Actual plot (dec. vector spacing and inc. key arrow size)



'*Diff*' image generated by image comparison of two

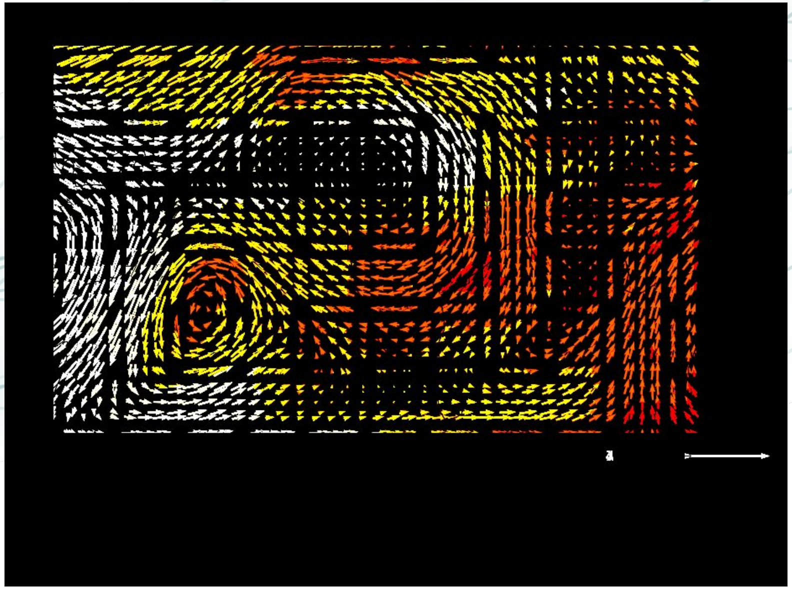
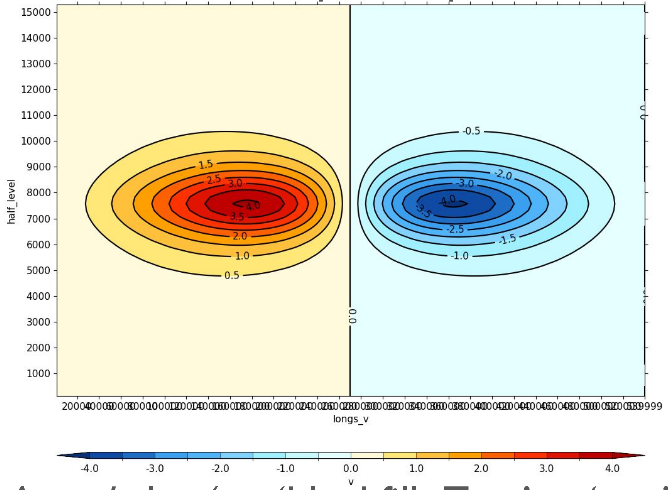
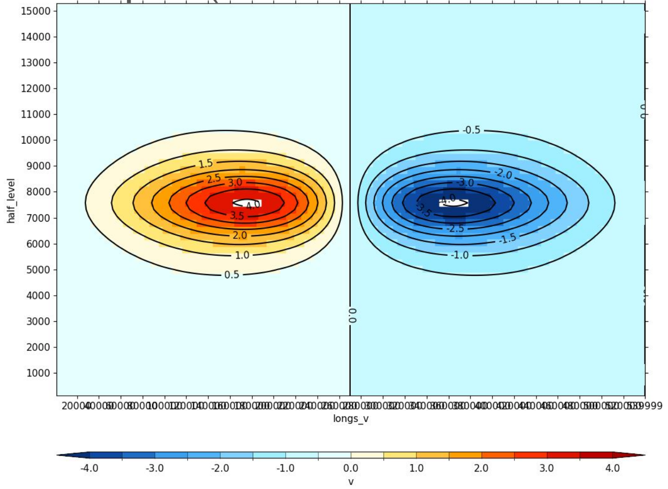


Image-comparison testing: example 1 (artificially made)

Stored *expected* plot



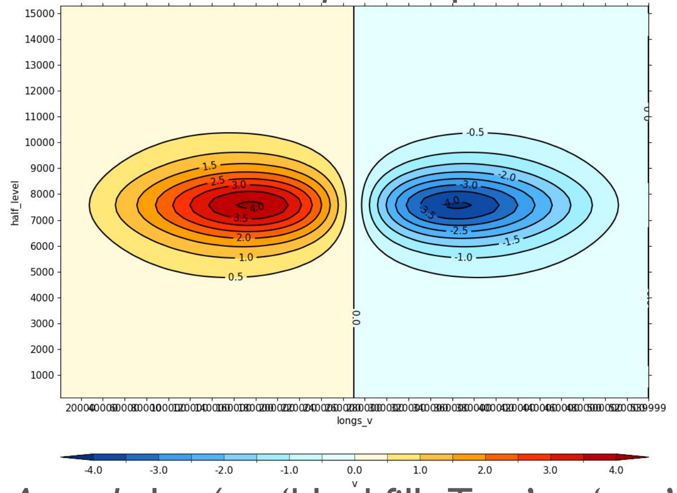
Actual plot (set ‘blockfill=True’ to ‘con’ call)



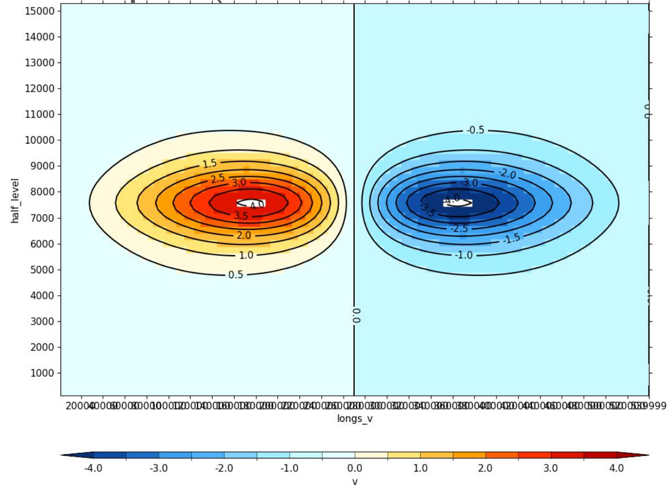
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Image-comparison testing: example 1 (artificially made)

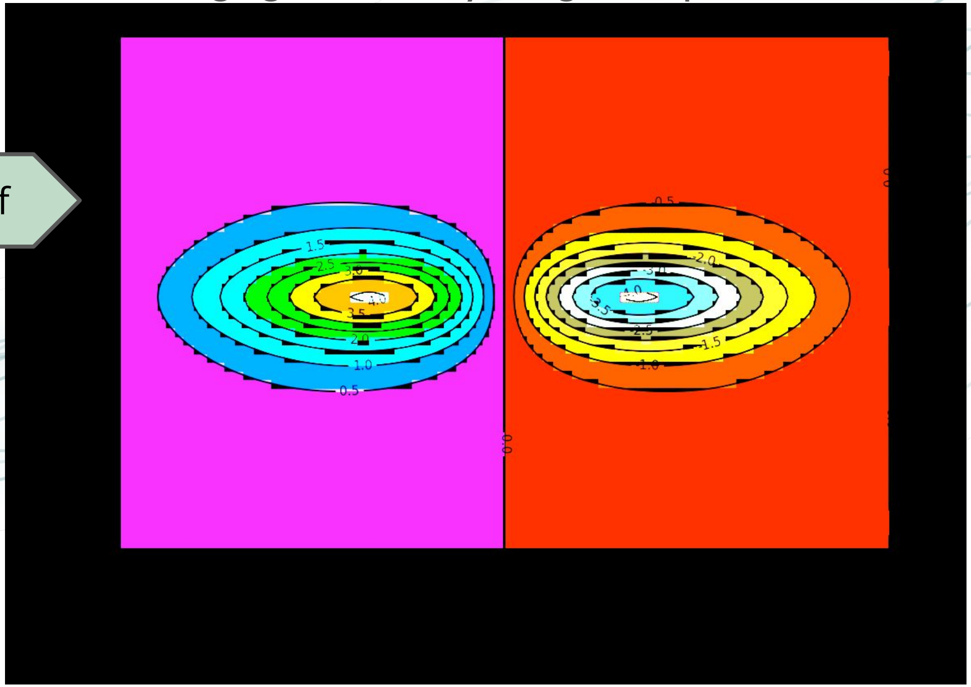
Stored *expected* plot



Actual plot (set 'blockfill=True' to 'con' call)



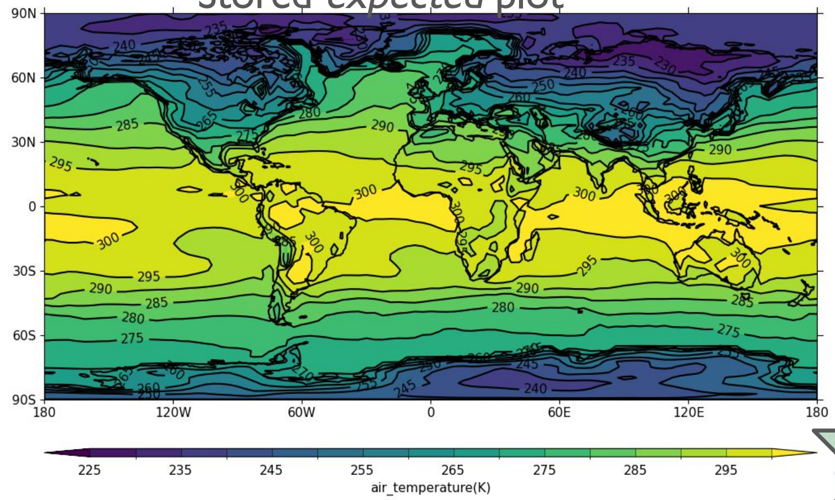
'*Diff*' image generated by image comparison of two



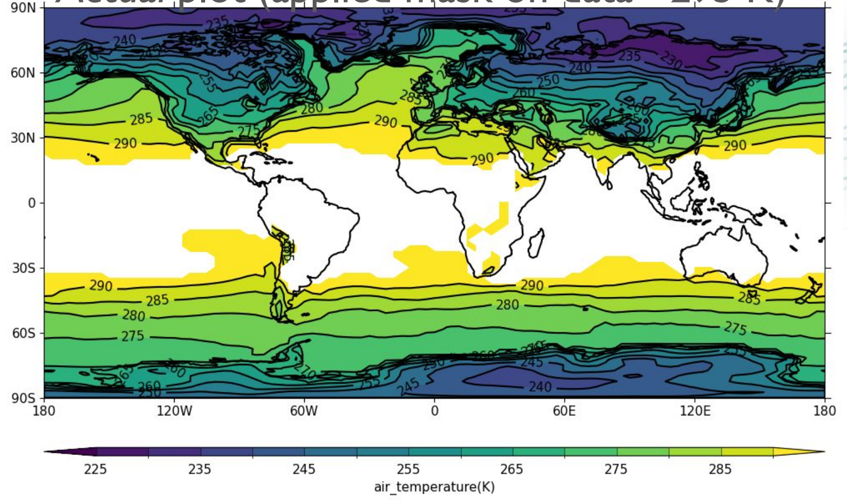
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Image-comparison testing: example 3 (artificially made)

Stored *expected* plot



Actual plot (applied mask on data >295 K)



diff

'*Diff*' image generated by image comparison of two

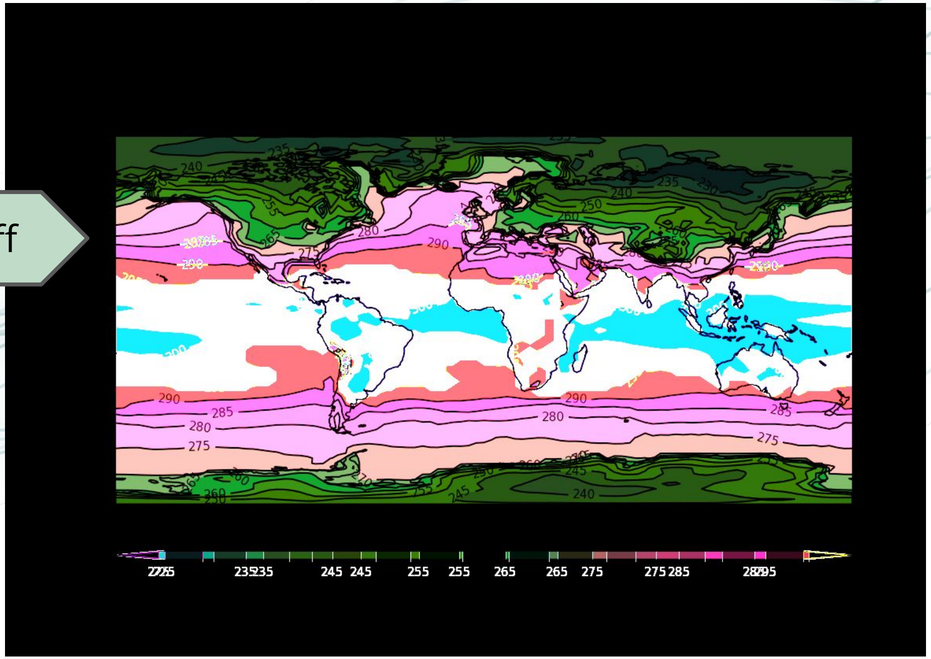
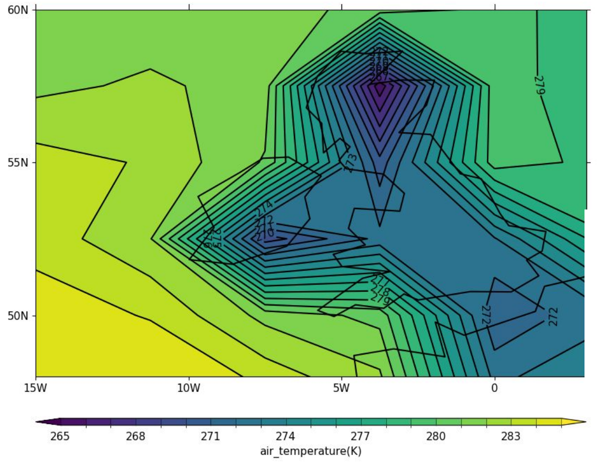
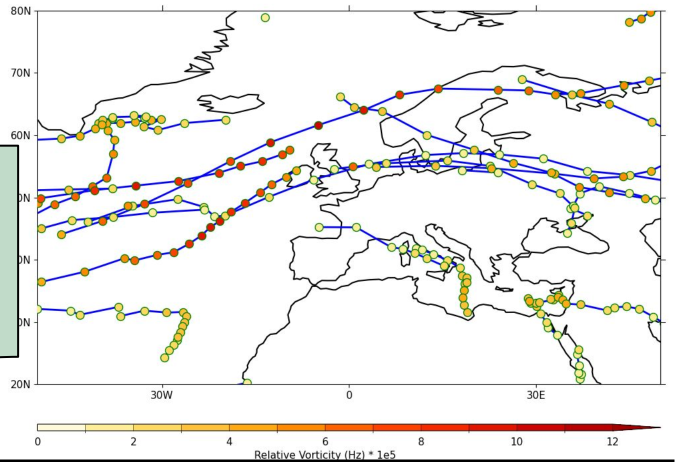


Image-comparison testing: working backwards, for real cases



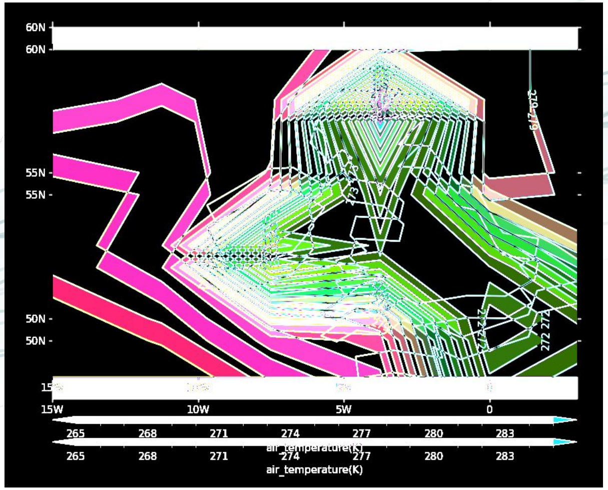
← Stored *expected* plots →

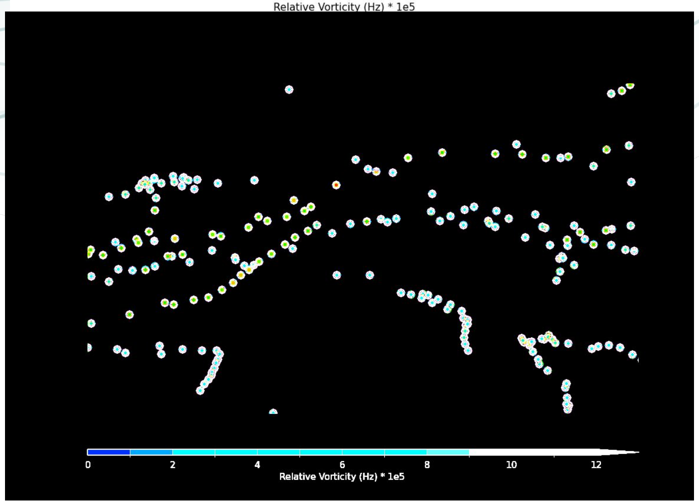


?

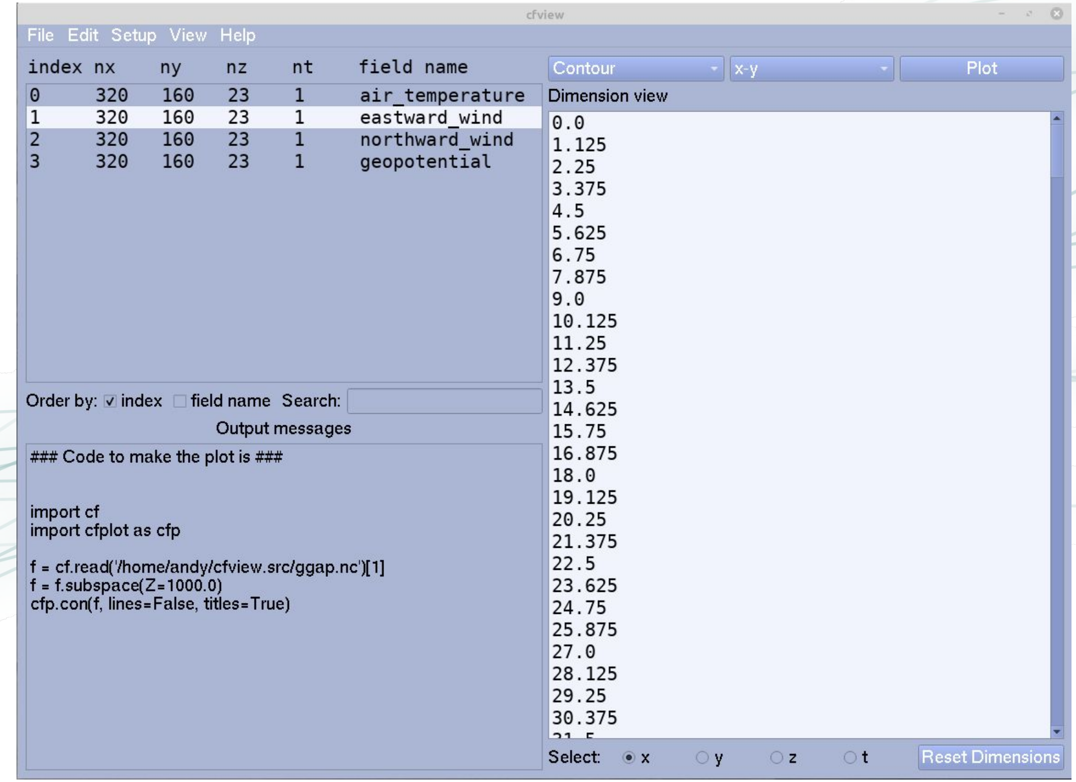
Guess why the generated image was not correct from the expected and diff

← Resulting *diff* plots →





2. Ensuring a PyQt GUI functions and looks as expected with PyQt testing (for cf-view)



```
index nx    ny    nz    nt    field name
0     320   160   23    1     air_temperature
1     320   160   23    1     eastward_wind
2     320   160   23    1     northward_wind
3     320   160   23    1     geopotential
```

```
0.0
1.125
2.25
3.375
4.5
5.625
6.75
7.875
9.0
10.125
11.25
12.375
13.5
14.625
15.75
16.875
18.0
19.125
20.25
21.375
22.5
23.625
24.75
25.875
27.0
28.125
29.25
30.375
```

```
### Code to make the plot is ###


import cf
import cfplot as cfp

f = cf.read('/home/andy/cfview.src/ggap.nc')[1]
f = f.subspace(Z=1000.0)
cfp.con(f, lines=False, titles=True)
```

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Exploratory stage, in this case: what are the options?

Possible frameworks for testing:

- **pytest-qt** (https://pytest-qt.readthedocs.io/en/latest/) is a pytest plugin that allows programmers to write tests for PyQt5, **PyQt6**, PySide2 and PySide6 applications ✅
- Some commercial GUI testing kits, such as Squish (https://www.qt.io/product/quality-assurance/squish), but they require payment - so that's a ❌

Some tutorials/walk-throughs I found on how to use pytest-qt:

- official docs tutorial: https://pytest-qt.readthedocs.io/en/latest/tutorial.html
- user blog tutorial: https://ilmanzo.github.io/post/testing_pyside_gui_applications/
- another user tutorial, as a GitHub repo: https://github.com/jmcgeheeiv/pyqttestexample

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Exploratory stage: what `pytest-qt` can do

- After Bryan's advice: ask ChatGPT for guidance on use [take to tab having asked ChatGPT the question '*How do I use pytest-qt to test a PyQT6 GUI?*']
- From the tutorials and ChatGPT guidance, my initial understanding is:
    - You can simulate user interaction e.g. clicks with a `qtbot` class
    - You can *wait* for certain *signals* or *conditions* to be emitted via `waitSignal`, `waitUntil` and `waitCallback`
    - You can also test start-up and exit states

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL