Hands-on session “developers”

Outline

- Contribute to pysteps
- Git flow
- Best practices
- Test the code
- Document the code

Contribute to pysteps

Welcome! Pysteps is a community-driven initiative for developing and maintaining an easy to use, modular, free and open-source Python framework for short-term ensemble prediction systems.

There are many ways to contribute to pysteps:

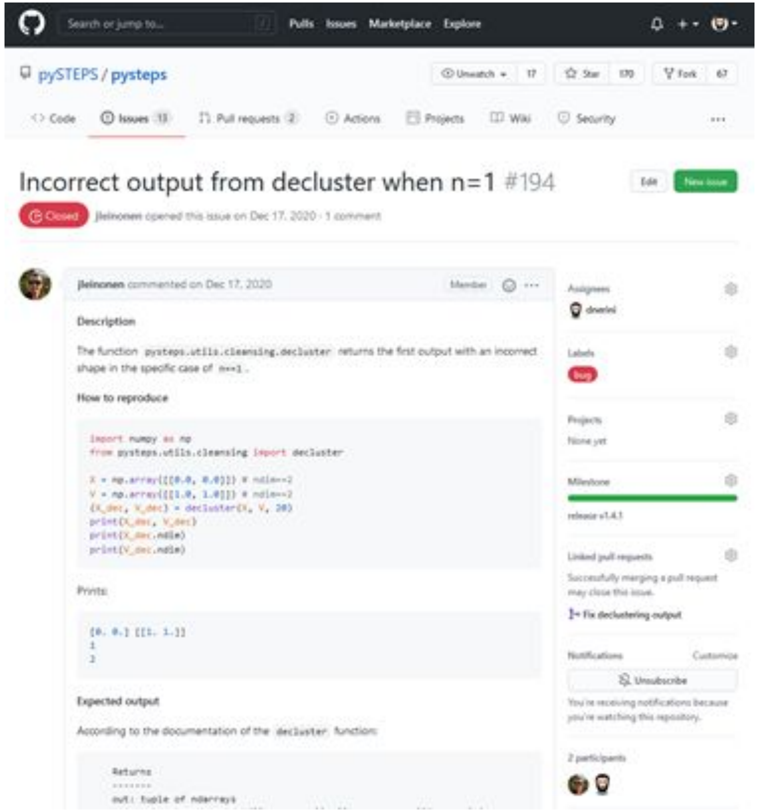
1. Bug reports and feature requests;
2. Code contributions with new features or bug fixes;
3. Documentation;
4. Writing examples;
5. Writing tests;
6. Providing feedback.

ISSUE:
RECENT UPDATE BROKE SUPPORT FOR HARDWARE I NEED FOR MY JOB.
WORKAROUND:
IF WE WAIT LONG ENOUGH, THE EARTH WILL EVENTUALLY BE CONSUMED BY THE SUN.

xkcd.com/1822

Step 0: Open an issue

1. Explain what you want to do and why.
2. Discuss how you plan to do it.
3. Make sure to start on the right track (and avoid dead ends).
4. Follow templates, e.g. bug reports:
    a. clear and descriptive title
    b. steps to reproduce
    c. obtained result
    d. expected result



Incorrect output from decluster when n=1 #194

jleinonen opened this issue on Dec 17, 2020 · 1 comment

jleinonen commented on Dec 17, 2020

Description

The function `pysteps.utils.cleansing.decluster` returns the first output with an incorrect shape in the specific case of `n==1`.

How to reproduce

```
import numpy as np
from pysteps.utils.cleansing import decluster

X = np.array([[0.0, 0.0]]) # ndim==2
V = np.array([[1.0, 1.0]]) # ndim==2
(X_dec, V_dec) = decluster(X, V, 20)
print(X_dec, V_dec)
print(X_dec.ndim)
print(V_dec.ndim)
```

Prints:

```
[0. 0.] [[1. 1.]]
1
2
```

Expected output

According to the documentation of the `decluster` function:

```
    Returns
    -------
    out: tuple of ndarrays
```

Git flow

1. **Create a branch**

Create a branch in your repository. A short, descriptive branch name enables your collaborators to see ongoing work at a glance. For example, *new-blending-weights*, *testing-probabilistic-nowcasting*.

Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**

Your branch is a safe place to make changes. If you make a mistake, you can revert your changes or push additional changes to fix the mistake. Your changes will not end up on the default branch until you merge your branch.

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**
    a. **commits**

- Any file addition, edit, or deletion is a commit.
- Each commit is an individual unit of change – this makes it easier to roll back a given change if necessary.
- Write clear commit messages.
  git commit -m «Concise commit message» -m «A more verbose message»

Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**
3. **Create a pull request**

Create a pull request to ask collaborators for feedback on your changes. Pull request review is so valuable that some repositories require an approving review before pull requests can be merged. If you want early feedback or advice before you complete your changes, you can mark your pull request as a draft.

NOTE: The work is not done yet, here it comes the most important/interactive part!!!

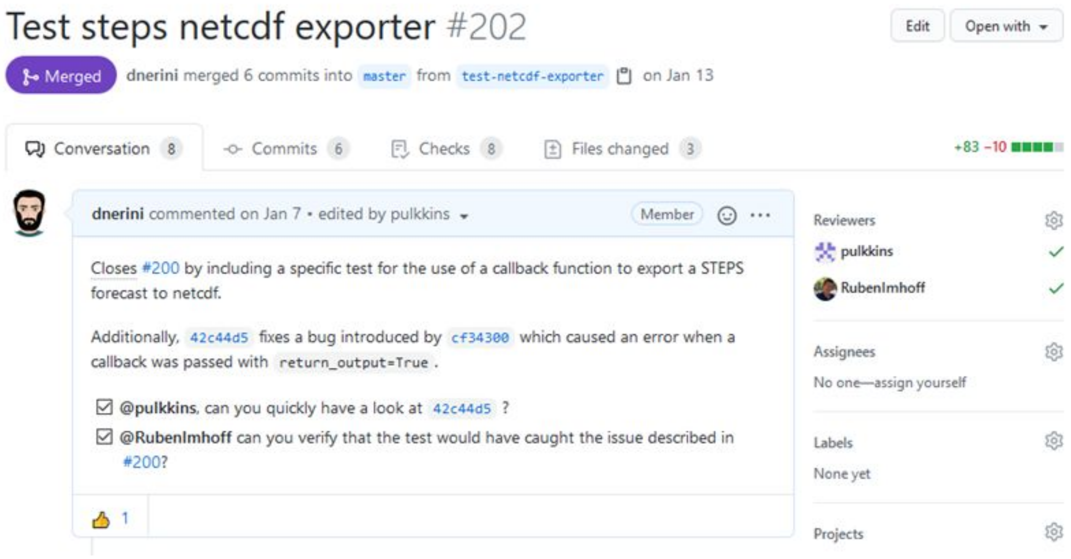
Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**
3. **Create a pull request**
4. **Address review comments**

Reviewers should leave questions, comments, and suggestions. Reviewers can comment on the whole pull request or add comments to specific lines. You and reviewers can insert images or code suggestions to clarify comments.

You can continue to commit and push changes in response to the reviews. *Your pull request will update automatically.*

Examples of review comments?



Test steps netcdf exporter #202

Merged dnerini merged 6 commits into `master` from `test-netcdf-exporter` on Jan 13

dnerini commented on Jan 7 • edited by pulkkins

Closes #200 by including a specific test for the use of a callback function to export a STEPS forecast to netcdf.

Additionally, `42c44d5` fixes a bug introduced by `cf34300` which caused an error when a callback was passed with `return_output=True` .

- [x] @pulkkins, can you quickly have a look at `42c44d5` ?
- [x] @RubenImhoff can you verify that the test would have caught the issue described in #200?

👍 1

Reviewers
pulkkins ✓
RubenImhoff ✓

Assignees
No one—assign yourself

Labels
None yet

Projects

Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**
3. **Create a pull request**
4. **Address review comments**
5. **Merge your pull request**

Once your pull request is approved, merge your pull request. This will automatically merge your branch so that your changes appear on the default branch.

Git flow

1. **Create a branch**
2. **Make changes/improvements/developments**
3. **Create a pull request**
4. **Address review comments**
5. **Merge your pull request**
6. **Delete your branch**

After you merge your pull request, delete your branch. This indicates that the work on the branch is complete and prevents you or others from accidentally using old branches.

Best practices: Naming conventions

- ➔ Follow the conventions.
- ➔ Use concise, self-explanatory names.
- ➔ Avoid mathematical syntax («X», «Y», «Z»).
- ➔ The length of a variable name should reflect its importance.
- ➔ The name of a function should tell what it does.
- ➔ Beware of typos and grammar, use a spell checker.

Type	Convention
Packages	lower_with_under
Module	lower_with_under
Classes	CapWords
Exceptions	CapWords
Functions	lower_with_under()
Global/Class Constants	CAPS_WITH_UNDER
Global/Class Variables	lower_with_under
Instance Variables	lower_with_under
Method Names	lower_with_under()
Function/Method Parameters	lower_with_under
Local Variables	lower_with_under

Best practices: Code styling

Follow Python's code style guidelines (PEP8).

Use an auto-formatter (black is pre-committed in pySTEPS).

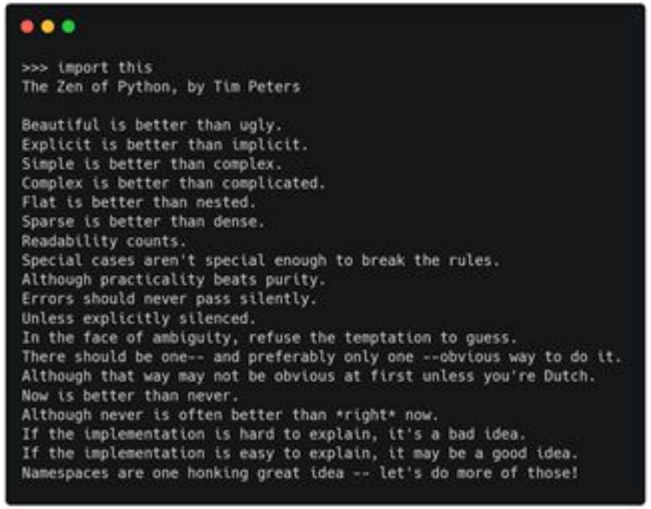
Avoid extraneous whitespace

Max line-length: 88 characters

Always indent wrapped code for readability

Always use four spaces for indentation (don't use tabs).

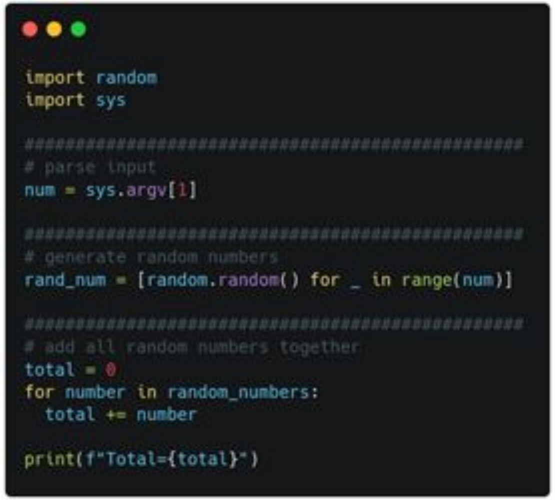
Avoid writing multiple statements in the same line.



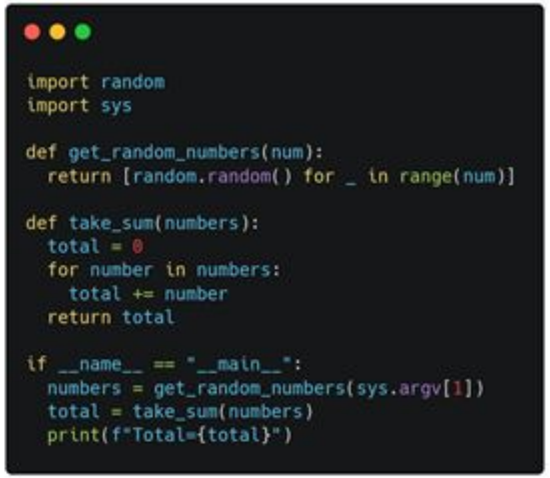
```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Best practices: Readability/Modularity

Avoid long scripts by organizing your code into functions. Each function should represent only one action.



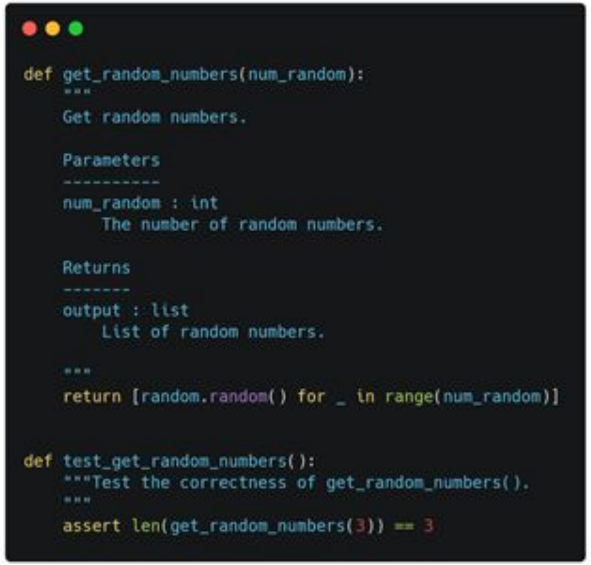
```python
import random
import sys

#################################################
# parse input
num = sys.argv[1]

#################################################
# generate random numbers
rand_num = [random.random() for _ in range(num)]

#################################################
# add all random numbers together
total = 0
for number in random_numbers:
  total += number

print(f"Total={total}")
```



```python
import random
import sys

def get_random_numbers(num):
  return [random.random() for _ in range(num)]

def take_sum(numbers):
  total = 0
  for number in numbers:
    total += number
  return total

if __name__ == "__main__":
  numbers = get_random_numbers(sys.argv[1])
  total = take_sum(numbers)
  print(f"Total={total}")
```

Test the code

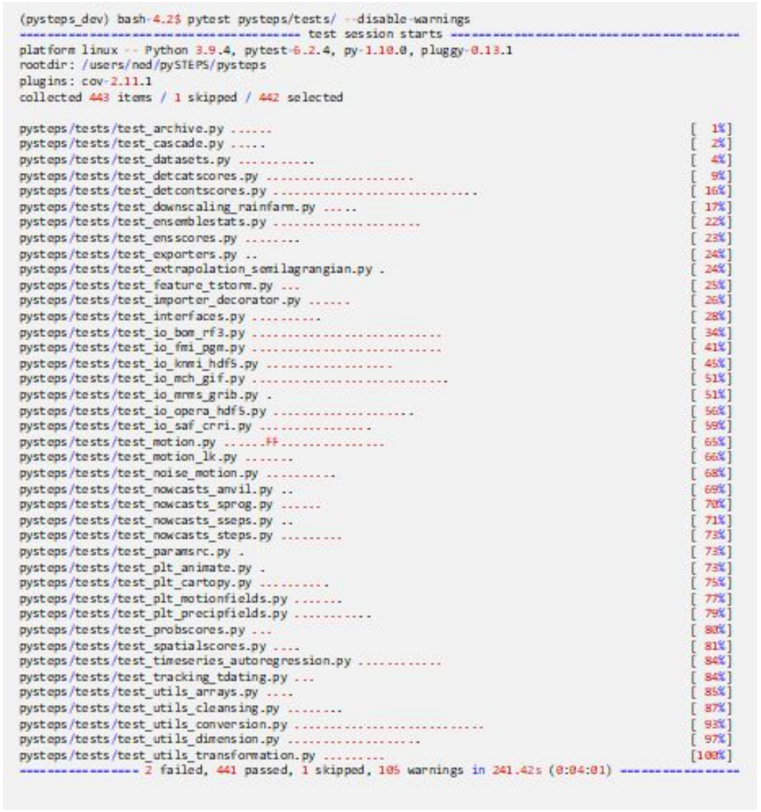
- Create a minimal test routine for your code change.
- e.g., if you fixed a bug, implement a new test that can reproduce that same bug.
- The tests should be placed under the pysteps.tests module.
- The file should follow the **test_*.py** naming convention and have a descriptive name.



```
def get_random_numbers(num_random):
    """
    Get random numbers.

    Parameters
    ----------
    num_random : int
        The number of random numbers.

    Returns
    -------
    output : list
        List of random numbers.

    """
    return [random.random() for _ in range(num_random)]


def test_get_random_numbers():
    """Test the correctness of get_random_numbers().
    """
    assert len(get_random_numbers(3)) == 3
```

Test your changes

- Tests are organized in a dedicated folder in your project.
- Run the whole test suite with pytest to make sure that your changes didn't break anything elsewhere.



```
(pysteps_dev) bash-4.2$ pytest pysteps/tests/ --disable-warnings
=================================== test session starts ====================================
platform linux -- Python 3.9.4, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /users/ned/pySTEPS/pysteps
plugins: cov-2.11.1
collected 443 items / 1 skipped / 442 selected

pysteps/tests/test_archive.py ......                                                  [  1%]
pysteps/tests/test_cascade.py .....                                                   [  2%]
pysteps/tests/test_datasets.py ...........                                            [  4%]
pysteps/tests/test_detcatscores.py .....................                                  [  9%]
pysteps/tests/test_detcontscores.py .............................                      [ 16%]
pysteps/tests/test_downscaling_rainfarm.py .....                                      [ 17%]
pysteps/tests/test_ensemblestats.py .....................                             [ 22%]
pysteps/tests/test_ensscores.py ........                                              [ 23%]
pysteps/tests/test_exporters.py ..                                                    [ 24%]
pysteps/tests/test_extrapolation_semilagrangian.py .                                  [ 24%]
pysteps/tests/test_feature_tstorm.py ...                                              [ 25%]
pysteps/tests/test_importer_decorator.py ......                                       [ 26%]
pysteps/tests/test_interfaces.py ..........                                           [ 28%]
pysteps/tests/test_io_bom_rf3.py ...........................                           [ 34%]
pysteps/tests/test_io_fmi_pgm.py ...........................                           [ 41%]
pysteps/tests/test_io_knmi_hdf5.py ..................                                  [ 45%]
pysteps/tests/test_io_mch_gif.py ............................                          [ 51%]
pysteps/tests/test_io_mrms_grib.py .                                                  [ 51%]
pysteps/tests/test_io_opera_hdf5.py ....................                               [ 56%]
pysteps/tests/test_io_saf_crri.py ................                                     [ 59%]
pysteps/tests/test_motion.py ......FF...............                                  [ 65%]
pysteps/tests/test_motion_lk.py .......                                               [ 66%]
pysteps/tests/test_noise_motion.py ..........                                         [ 68%]
pysteps/tests/test_nowcasts_anvil.py ..                                               [ 69%]
pysteps/tests/test_nowcasts_sprog.py ......                                           [ 70%]
pysteps/tests/test_nowcasts_sseps.py ..                                               [ 71%]
pysteps/tests/test_nowcasts_steps.py .........                                        [ 73%]
pysteps/tests/test_paramsrc.py .                                                      [ 73%]
pysteps/tests/test_plt_animate.py .                                                   [ 73%]
pysteps/tests/test_plt_cartopy.py ..........                                          [ 75%]
pysteps/tests/test_plt_motionfields.py .......                                        [ 77%]
pysteps/tests/test_plt_precipfields.py ...........                                    [ 79%]
pysteps/tests/test_probscores.py ...                                                  [ 80%]
pysteps/tests/test_spatialscores.py ....                                              [ 81%]
pysteps/tests/test_timeseries_autoregression.py ............                          [ 84%]
pysteps/tests/test_tracking_tdating.py ...                                            [ 84%]
pysteps/tests/test_utils_arrays.py ....                                               [ 85%]
pysteps/tests/test_utils_cleansing.py ........                                        [ 87%]
pysteps/tests/test_utils_conversion.py ...........................                     [ 93%]
pysteps/tests/test_utils_dimension.py ...................                              [ 97%]
pysteps/tests/test_utils_transformation.py .........                                  [100%]
================ 2 failed, 441 passed, 1 skipped, 105 warnings in 241.42s (0:04:01) ================
```

Examples: documentation

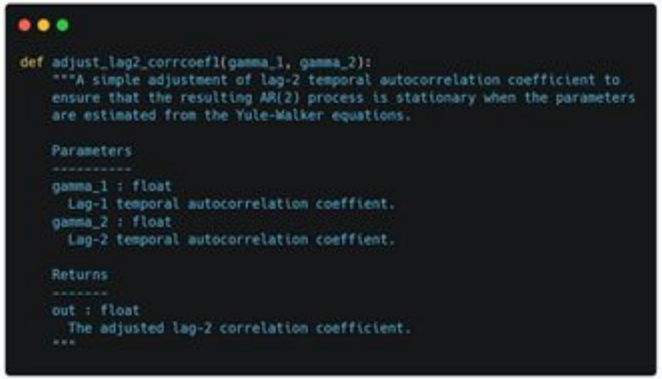
Document all your modules, class, and functions (PEP257).

A summary line, followed by a single blank line, followed by a more elaborate description.

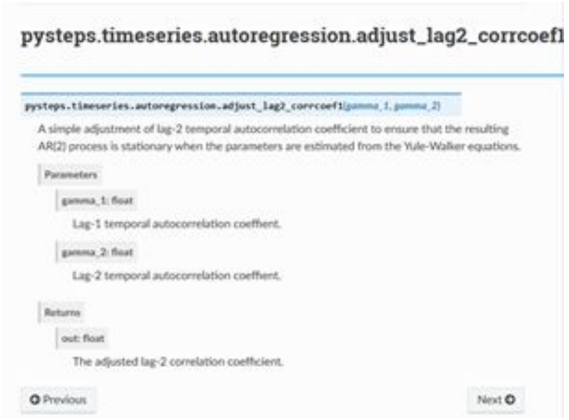
Use, for example, the Numpy's docstring format.

Can be used by automatic documentation tools such as Sphinx.

Add usage example.



```
def adjust_lag2_corrcoef1(gamma_1, gamma_2):
    """A simple adjustment of lag-2 temporal autocorrelation coefficient to
    ensure that the resulting AR(2) process is stationary when the parameters
    are estimated from the Yule-Walker equations.

    Parameters
    ----------
    gamma_1 : float
      Lag-1 temporal autocorrelation coeffient.
    gamma_2 : float
      Lag-2 temporal autocorrelation coeffient.

    Returns
    -------
    out : float
      The adjusted lag-2 correlation coefficient.
    """
```



pysteps.timeseries.autoregression.adjust_lag2_corrcoef1

`pysteps.timeseries.autoregression.adjust_lag2_corrcoef1(gamma_1, gamma_2)`

A simple adjustment of lag-2 temporal autocorrelation coefficient to ensure that the resulting AR(2) process is stationary when the parameters are estimated from the Yule-Walker equations.

Parameters

gamma_1: float

Lag-1 temporal autocorrelation coeffient.

gamma_2: float

Lag-2 temporal autocorrelation coeffient.

Returns

out: float

The adjusted lag-2 correlation coefficient.

Minor issues/improvements for the hands-on activities

1. Add tests
    - Issue 1: Test that a ValueError exception is raised when the input array contains NaNs
    - Issue 2: Test that input arrays with more than 2 dimensions raise a ValueError exception
    - Issue 3: Test that a 1D-array with the coordinates' scale is accepted
    - Issue 4: Test that a ValueError exception is raised when the input array contains NaNs
    - Issue 5: Test that input arrays with more than 2 dimensions raise a ValueError exception
    - Issue 6: Test that input arrays with incorrect dimensions raise a ValueError exception
    - Issue 7: Test that passing the timesteps raises a ValueError exception if they are not sorted in ascending order
2. Small code-refactorings
    - Issue 8: Improve the get_method interface from the pysteps.verification.interface.py module
    - Issue 9: Replace collections.abc.Iterable by an explicit import of the Iterable class in detcatscores module
    - Issue 10: Replace collections.abc.Iterable by an explicit import of the Iterable class in the detcontscores module
3. Replace old style string formatting (% Operator) with f-strings
    - Issue 11: Replace old style string formatting (% operator) with f-strings in pysteps.utils.tapering module
    - Issue 12: Replace old style string formatting (% operator) with f-strings in pysteps.blending.steps module
    - Issue 13: Replace old style string formatting (% operator) with f-strings in pysteps.utils.spectral module
    - Issue 14: Replace old style string formatting (% operator) with f-strings in pysteps.io.archive module
    - Issue 15: Replace old style string formatting (% operator) with f-strings in pysteps.utils.cleansing module

Minor issues/improvements for the hands-on activities

- Testing:
    - Test that a ValueError exception is raised when the input array contains NaNs.
- Code refactoring:
    - Improve the get_method interface from the pysteps.verification.interface.py module.
- Improve code style:
    - Replace old style string formatting (% operator) with f-strings in pysteps.io.archive module

Examples: bug reports and feature request

-
- Bug fixing

Examples: contributions, new features, or bug fixes

Agregar una funcionalidad nueva

Arreglar un bug

Examples: usage examples