pyrcmip

Release 0.5.2+14.gf0e1e66.dirty

Zeb Nicholls, Jared Lewis

Mar 08, 2022

DOCUMENTATION

pyrcmip is a tool for validating and uploading results to RCMIP. The Reduced Complexity Model Intercomparison Project (RCMIP) is a project to evaluate reduced-complexity (also known as simple) climate models and compare them against CMIP coupled models.

CHAPTER

ONE

LICENSE

pyrcmip is free software under a BSD 3-Clause License, see LICENSE.

If you make use of pyrcmip or any of the RCMIP project, please cite Nicholls et al., *GMDD* 2020 [1].

CHAPTER

TWO

REFERENCES

2.1 Installation

The easiest way to install pyrcmip is with pip. At this stage pyrcmip only supports Python 3.6+.

```
# if you're using a virtual environment, make sure you're in it
pip install pyrcmip
```

2.2 Submitting results

If you're interested in submitting results to RCMIP then you're in the right place. Here we go through the process of preparing and submitting results to RCMIP. If you have any issues with this guide, or feel it could be improved, please don't hesitate to raise an issue in the pyrcmip issue tracker or make a merge request.

A set of Jupyter Notebooks for the running the RCMIP experiments and uploading the results using the Geoffroy et al. (2013) two-layer model, as implemented in openscm-twolayermodel are available in notebooks/example-model-pipeline. These notebooks can be launched directly using binder. We would love to share more examples of running your models using the RCMIP protocol.

2.2.1 Performing the experiments

The first step to submitting is performing the experiments. Our protocol is currently available from the RCMIP website, under the initial datasets header. Please follow the protocol as closely as possible. If you have any questions about the protocol or how to follow it, please raise an issue in the pyrcmip issue tracker.

2.2.2 Preparing the submission

Having performed the experiments, next you need to prepare your submission. Submission via pyrcmip is a largely automated process, hence looks a little different to how submission looked in RCMIP phase 1.

For submission via pyrcmip, you need three things:

1. Timeseries to be submitted
2. Model reported metrics
3. Metadata about your submission

Timeseries

The first part of the submission is the timeseries. These can be provided in one of three ways.

1. As the `your_data` sheet in our submission protocol (e.g. https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip_model_output_test.xlsx).
2. As a standalone csv (or gzipped csv) of the same format as the `your_data` sheet in our submission protocol (https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip_model_output_test.csv).
3. As a standalone netCDF file in scmdata's netCDF format (e.g. https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip_model_output_test.csv, further details on the format at https://github.com/openscm/scmdata/blob/v0.6.3/notebooks/netcdf.ipynb).

Differences from RCMIP Phase 1

For those who submitted to RCMIP Phase 1, please note the following two differences:

1. we now ask for an extra column `ensemble_member`, which provides an index so we can distinguish different model configurations within a probabilistic ensemble
2. the column headings have changed slightly (our readers should be able to handle the old style, but updating if you can would be much appreciated)

Model reported metrics

We also ask you to report some metrics which cannot be derived from any RCMIP experiments. At this stage, the only such metric is Equilibrium Climate Sensitivity (none of our experiments are long enough to reach true equilibrium). We ask that you submit a csv which documents the Equilibrium Climate Sensitivity of each `ensemble_member` provided in the timeseries part of the submission. An example of such a csv is shown in https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip_model_reported_metrics_test.csv.

Metadata

The final part of the submission is metadata. This simply provides metadata about your model which can be used as documentation. This metadata can be provided in one of two ways:

1. as a csv of the same format as https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip_model_metadata_test.csv
2. by saving the `meta_model` sheet of our submission protocol as a standalone csv (this should result in a csv like https://gitlab.com/rcmip/pyrcmip/-/tree/master/tests/data/rcmip-model-meta-test.csv)

Differences from RCMIP Phase 1

We have only made one change compared to RCMIP Phase 1:

1. we have removed the ECS column from the `meta_model` sheet

2.2.3 Validating the submission

Once you have prepared your submission, you can then use RCMIP's command-line interface to validate it. This is done using the `rcmip validate` command. For full details, please see the validate section in our *Command-line interface* documentation. This command will validate your submission, highlighting any errors it finds and providing you with a green light otherwise. If your submission does not pass validation, you will not be able to upload it in the next step. If you have any questions or issues with validation, please raise an issue in the pyrcmip issue tracker.

Note: The validation and uploading process can take some time (and a lot of memory) especially with large ensembles.

If you are having issues uploading large ensembles of results, split the input timeseries into smaller, more manageable chunks and pass all those chunks to the `validate` or `upload` command. Each chunk will be processed independently.

2.2.4 Uploading the submission

Once your submission has been validated, you can then upload it. This is done using the `rcmip upload` command. For full details, please see the upload section in our *Command-line interface* documentation. This command will validate (again, just in case) and then upload your submission (assuming the validation passed). If you have any questions or issues with upload, please raise an issue in the pyrcmip issue tracker.

2.3 Development

If you're interested in contributing to pyrcmip, we'd love to have you on board! This section of the docs details how to get setup to contribute and how best to communicate.

2.3.1 Contributing

All contributions are welcome, some possible suggestions include:

- tutorials (or support questions which, once solved, result in a new tutorial :D)
- blog posts
- improving the documentation
- bug reports
- feature requests
- pull requests

Please report issues or discuss feature requests in the pyrcmip issue tracker. If your issue is a feature request or a bug, please use the templates available, otherwise, simply open a normal issue :)

As a contributor, please follow a couple of conventions:

- Create issues in the pyrcmip issue tracker for changes and enhancements, this ensures that everyone in the community has a chance to comment
- Be welcoming to newcomers and encourage diverse new contributors from all backgrounds: see the Python Community Code of Conduct

2.3.2 Getting setup

To get setup as a developer, we recommend the following steps (if any of these tools are unfamiliar, please see the resources we recommend in *Development tools*):

1. Install conda and make
2. Run `make conda-environment`, if that fails you can try doing it manually by reading the commands from the `Makefile`
3. Make sure the tests pass by running `make test`, as above if that fails you can try doing it manually by reading the commands from the `Makefile`

Getting help

Whilst developing, unexpected things can go wrong (that's why it's called 'developing', if we knew what we were doing, it would already be 'developed'). Normally, the fastest way to solve an issue is to contact us via the issue tracker. The other option is to debug yourself. For this purpose, we provide a list of the tools we use during our development as starting points for your search to find what has gone wrong.

Development tools

This list of development tools is what we rely on to develop pyrcmip reliably and reproducibly. It gives you a few starting points in case things do go inexplicably wrong and you want to work out why. We include links with each of these tools to starting points that we think are useful, in case you want to learn more.

- Git
- Make
- **Conda virtual environments**
  - note the common gotcha that `source activate` has now changed to `conda activate`
  - we use conda instead of pure pip environments because they help us deal with Iris' dependencies: if you want to learn more about pip and pip virtual environments, check out this introduction
- **Tests**
  - we use a blend of pytest and the inbuilt Python testing capabilities for our tests so checkout what we've already done in `tests` to get a feel for how it works
- **Continuous integration (CI)**
  - we use GitLab CI for our CI but there are a number of good providers
- **Jupyter Notebooks**
  - we'd recommend simply installing `jupyter` (`conda install jupyter`) in your virtual environment
- Sphinx

Other tools

We also use some other tools which aren't necessarily the most familiar. Here we provide a list of these along with useful resources.

- **Regular expressions**
  - we use regex101.com to help us write and check our regular expressions, make sure the language is set to Python to make your life easy!

2.3.3 Formatting

To help us focus on what the code does, not how it looks, we use a couple of automatic formatting tools. These automatically format the code for us and tell use where the errors are. To use them, after setting yourself up (see *Getting setup*), simply run `make black` and `make flake8`. Note that `make black` can only be run if you have committed all your work i.e. your working directory is 'clean'. This restriction is made to ensure that you don't format code without being able to undo it, just in case something goes wrong.

2.3.4 Buiding the docs

After setting yourself up (see *Getting setup*), building the docs is as simple as running `make docs` (note, run `make -B docs` to force the docs to rebuild and ignore make when it says '... index.html is up to date'). This will build the docs for you. You can preview them by opening `docs/build/html/index.html` in a browser.

For documentation we use Sphinx. To get ourselves started with Sphinx, we started with this example then used Sphinx's getting started guide.

Gotchas

To get Sphinx to generate pdfs (rarely worth the hassle), you require Latexmk. On a Mac this can be installed with `sudo tlmgr install latexmk`. You will most likely also need to install some other packages (if you don't have the full distribution). You can check which package contains any missing files with `tlmgr search --global --file [filename]`. You can then install the packages with `sudo tlmgr install [package]`.

Docstring style

For our docstrings we use numpy style docstrings. For more information on these, here is the full guide and the quick reference we also use.

2.3.5 Releasing

The steps to release a new version of pyrcmip are shown below. Please do all the steps below and all the steps for both release platforms.

First step

1. Test installation with dependencies `make test-install`
2. Update `CHANGELOG.rst`:
   - add a header for the new version between `master` and the latest bullet point
   - this should leave the section underneath the master header empty
3. `git add .`
4. `git commit -m "Prepare for release of vX.Y.Z"`
5. `git tag vX.Y.Z`
6. Test version updated as intended with `make test-install`

PyPI

If uploading to PyPI, do the following (otherwise skip these steps)

1. `make publish-on-testpypi`
2. Go to test PyPI and check that the new release is as intended. If it isn't, stop and debug.
3. Test the install with `make test-testpypi-install` (this doesn't test all the imports as most required packages are not on test PyPI).

Assuming test PyPI worked, now upload to the main repository

1. `make publish-on-pypi`
2. Go to pyrcmip's PyPI and check that the new release is as intended.
3. Test the install with `make test-pypi-install` (a pip only install will throw warnings about Iris not being installed, that's fine).

Push to repository

Finally, push the tags and the repository

1. `git push`
2. `git push --tags`

Conda

Note: Conda releases are not yet operational

1. If you haven't already, fork the pyrcmip conda feedstock. In your fork, add the feedstock upstream with `git remote add upstream https://github.com/conda-forge/pyrcmip-feedstock` (`upstream` should now appear in the output of `git remote -v`)
2. Update your fork's master to the upstream master with:
    1. `git checkout master`
    2. `git fetch upstream`
    3. `git reset --hard upstream/master`
3. Create a new branch in the feedstock for the version you want to bump to.
4. Edit `recipe/meta.yaml` and update:
    - version number in line 1 (don't include the 'v' in the version tag)
    - the build number to zero (you should only be here if releasing a new version)
    - update `sha256` in line 9 (you can get the sha from pyrcmip's PyPI by clicking on 'Download files' on the left and then clicking on 'SHA256' of the `.tar.gz` file to copy it to the clipboard)
5. `git add .`
6. `git commit -m "Update to vX.Y.Z"`
7. `git push`
8. Make a PR into the pyrcmip conda feedstock
9. If the PR passes (give it at least 10 minutes to run all the CI), merge

10. Check https://anaconda.org/conda-forge/pyrcmip to double check that the version has increased (this can take a few minutes to update)

2.3.6 Why is there a `Makefile` in a pure Python repository?

Whilst it may not be standard practice, a `Makefile` is a simple way to automate general setup (environment setup in particular). Hence we have one here which basically acts as a notes file for how to do all those little jobs which we often forget e.g. setting up environments, running tests (and making sure we're in the right environment), building docs, setting up auxillary bits and pieces.

2.3.7 Why did we choose a BSD 2-Clause License?

We want to ensure that our code can be used and shared as easily as possible. Whilst we love transparency, we didn't want to **force** all future users to also comply with a stronger license such as AGPL. Hence the choice we made.

We recommend Morin et al. 2012 for more information for scientists about open-source software licenses.

2.4 Assessed Ranges API

Handling of assessed ranges

class pyrcmip.assessed_ranges.**AssessedRanges**(*db*)

Bases: `object`

Class for handling assessed ranges and performing operations with them.

For example, getting values for specific metrics and plotting results against assessed ranges.

assessed_range_label = 'assessed range'

String used for labelling assessed ranges (in plots, dataframes etc.)

Type str

calculate_metric_from_results(*metric*, *res_calc*, *custom_calculators=None*)

Calculate metric values from results

Parameters

- **metric** (*str*) – Metric for which to calculate results
- **res_calc** (scmdata.ScmRun) – Results to use for the calculation
- **custom_calculators** (tuple(*pyrcmip.metric_calculations.base.Calculator*)) – Custom calculators to use for calculating metrics which require a custom calculation

Returns pd.DataFrame containing the calculated metric values alongside other relevant metadata

Return type pd.DataFrame

Raises ValueError – Data required to calculate the metric is not available

check_norm_period_evaluation_period_against_data(*norm_period*, *evaluation_period*, *data*)

Check the normalisation and evaluation periods against the data

Parameters

- **norm_period** (*None or range(int, int)*) – Normalisation period to check. If None, no check is performed.

- **evaluation_period** (*None or range(int, int)*) – Evaluation period to check. If None, no check is performed.
- **data** (scmdata.ScmRun) – Data to check

Raises ValueError – The data is incompatible with the periods (e.g. the normalisation period begins before the data begins).

get_assessed_range_for_boxplot(*metric*, *n_to_draw=20000*)

Get assessed range for a box plot

This converts the assessed range from IPCC language (very likely, likely, central) into a distribution of values, based on *pyrcmip.stats.get_skewed_normal()*.

Parameters

- **metric** (*str*) – Metric for which to get assessed range distribution
- **n_to_draw** (*int*) – Number of points to include in the returned distribution

Returns pd.DataFrame with n_to_draw rows, each of which contains a drawn value for metric. The returned values are put in a column whose name is equal to the value of metric. We also return a "unit" column and a "Source" column. The "Source" column is filled with self.assessed_range_label. Note that if the central value is nan, the entire distribution will simply be filled with nan.

Return type pd.DataFrame

get_col_for_metric(*metric*, *col*)

Get value of column for a given metric (i.e. RCMIP name)

Parameters

- **metric** (*str*) – Metric whose values we want to look up
- **col** (*str*) – Column whose values we want (e.g. "RCMIP scenario")

Returns The value in the column

Return type str

Raises

- **ValueError** – The metric could not be found in self.db
- **KeyError** – The column could not be found in self.db

get_col_for_metric_list(*metric*, *col*, *delimeter=','*)

Get value of column for a given metric (i.e. RCMIP name), split using a delimeter

Parameters

- **metric** (*str*) – Metric whose values we want to look up
- **col** (*str*) – Column whose values we want (e.g. "RCMIP scenario")
- **delimeter** (*str*) – Delimeter used to split col's values

Returns List of values, derived by splitting

Return type list

Raises TypeError – The found values are not a string (i.e. cannot be split by a delimiter)

get_norm_period_evaluation_period(*metric*)

Get normalisation and evaluation period for a given metric

Parameters **metric** (*str*) – Metric for which to get normalisation and evaluation periods

Returns Normalisation period and evaluation period. Each return value is a range of years which define the relevant period. If there is no period supplied, `None` is returned. For example, if the evaluation period is 1961-1990 and there is no reference period, then `None, range(1961, 1990 + 1)` is returned.

Return type norm_period, evaluation_period

Raises `ValueError` – A period could not be resolved because it is ambiguous i.e. it has nan for the start/end of the period while the other value is not nan.

get_results_summary_table_for_metric(*metric*, *model_results*)
Get results summary table for a given metric

Parameters

- **metric** (*str*) – Metric for which to get the summary table
- **model_results** (`pd.DataFrame`) – `pd.DataFrame` containing the model results. It must have at least the following columns: `"climate_model", "value"`.

Returns `pd.DataFrame` containing a summary of the results. The percentage difference is calculated as `(model_value - assessed_value) / np.abs(assessed_value) * 100`.

Return type `pd.DataFrame`

get_variables_regions_scenarios_for_metric(*metric*, *single_value=True*)
Get variables, regions and scenarios required to calculate a given metric

Parameters **metric** (*str*) – Metric for which to get values

Returns Dictionary containing required variables, regions and scenarios

Return type dict

head(*n=5*)
Get head of `self.db`

Parameters **n** (*int*) – Number of rows to return

Returns Head of `self.db`

Return type `pd.DataFrame`

metric_column = 'RCMIP name'
Name of the column which holds the names of the metrics being assessed

Type str

plot_against_results(*results_database*, *climate_models=['*']*, *custom_calculators=None*, *palette=None*)
Calculate metric values from results, compare and plot against assessed ranges

Parameters

- **metric** (*str*) – Metric for which to calculate results
- **results_database** (`pyrcmip.database.DataBase`) – Database from which to load results
- **climate_models** (*list[str]*) – Climate models to calculate results for
- **custom_calculators** (tuple(*pyrcmip.metric_calculations.base.Calculator*)) – Custom calculators to use for calculating metrics which require a custom calculation
- **palette** (*dict[str, str]*) – Colours to use for the different climate models and assessed ranges when plotting

Returns pd.DataFrame containing a dataframe based on concatenating the results from calling *get_results_summary_table_for_metric()* for each metric.

Return type pd.DataFrame

plot_metric_and_results(*metric*, *model_results*, *axes=None*, *palette=None*)
Plot our parameterisation of the metric's distribution and the model results

This produces a two-panel plot, the top panel has the distributions, the bottom panel has box and whisker plots (with the boxes and whiskers adjusted to match the IPCC calibrated likelihood language).

Parameters

- **metric** (*str*) – Metric to plot
- **model_results** (pd.DataFrame) – pd.DataFrame with the model results. Should be of the form returned by *calculate_metric_from_results()*.
- **axes** ((matplotlib.axes.SubplotBase, matplotlib.axes.SubplotBase)) – Axes on which to make the plots. Must be two-panels.
- **palette** (*dict[str, str]*) – Colours to use for the different climate models and assessed ranges

Returns Axes on which the plot was made

Return type (matplotlib.axes.SubplotBase, matplotlib.axes.SubplotBase)

Raises **AssertionError** – axes doesn't have a length equal to two

plot_metric_and_results_box_only(*metric*, *model_results*, *ax=None*, *palette=None*)
Plot box and whisker plots of the metric's distribution and the model results

The box and whisker plots have the boxes and whiskers adjusted to match the IPCC calibrated likelihood language).

Parameters

- **metric** (*str*) – Metric to plot
- **model_results** (pd.DataFrame) – pd.DataFrame with the model results. Should be of the form returned by *calculate_metric_from_results()*.
- **axes** (matplotlib.axes.SubplotBase) – Axis on which to make the plot
- **palette** (*dict[str, str]*) – Colours to use for the different climate models and assessed ranges

Returns Axes on which the plot was made

Return type matplotlib.axes.SubplotBase

plot_model_reported_against_assessed_ranges(*model_reported*, *palette=None*)
Compare and plot model reported results against assessed ranges

Parameters

- **model_reported** (pd.DataFrame) – pd.DataFrame of the same format as the result of *calculate_metric_from_results()*
- **palette** (*dict[str, str]*) – Colours to use for the different climate models and assessed ranges when plotting

Returns pd.DataFrame containing a dataframe based on concatenating the results from calling *get_results_summary_table_for_metric()* for each metric

Return type pd.DataFrame

`tail`(*n=5*)
Get tail of `self.db`

Parameters `n` (*int*) – Number of rows to return

Returns Tail of `self.db`

Return type `pd.DataFrame`

2.5 Command-line interface

2.5.1 rcmip

Command-line interface for pyrcmip

```
rcmip [OPTIONS] COMMAND [ARGS]...
```

Options

`--log-level` <log_level>

Options DEBUG | INFO | WARNING | ERROR | EXCEPTION | CRITICAL

download

Download submitted files

```
rcmip download [OPTIONS] OUTDIR
```

Options

`--token` <token>
Required Authentication token. Contact zebedee.nicholls@climate-energy-college.org for a token

`--bucket` <bucket>

`--model` <model>
Required

`--version` <version>
Required Version of the data that was uploaded. Must be a valid semver version string (https://semver.org/). For example 2.0.0

Arguments

OUTDIR
Required argument

upload

Validate and upload data to RCMIP's S3 bucket.

All the files for a given version have to be uploaded together.

One or more TIMESERIES files in which the timeseries output is stored. These should be CSV or NetCDF files conforming to the format expected by `scmdata`. Multiple timeseries inputs can be specified, but care must be taken to ensure that all of the individual timeseries have unique metadata. Each timeseries file will be validated and uploaded independently.

MODEL_REPORTED is the CSV file in which the model reported metrics are stored.

METADATA is the CSV file in which the metadata output is stored.

```
rcmip upload [OPTIONS] TIMESERIES... MODEL_REPORTED METADATA
```

Options

--token <token>
Required Authentication token. Contact zebedee.nicholls@climate-energy-college.org for a token

--bucket <bucket>

--model <model>
Required

--version <version>
Required Version of the data being uploaded. Must be a valid semver version string (https://semver.org/). For example 2.0.0

Arguments

TIMESERIES
Required argument(s)

MODEL_REPORTED
Required argument

METADATA
Required argument

validate

Validate submission input

Three different types of input data are required for validation, namely:

One or more `TIMESERIES` files in which the timeseries output is stored. These should be CSV or NetCDF files conforming to the format expected by `scmdata`. Multiple timeseries inputs can be specified, but care must be taken to ensure that all of the individual timeseries have unique metadata.

`MODEL_REPORTED` is the CSV file in which the model reported metrics are stored.

`METADATA` is the CSV file in which the metadata output is stored.

```
rcmip validate [OPTIONS] TIMESERIES... MODEL_REPORTED METADATA
```

Arguments

TIMESERIES
Required argument(s)

MODEL_REPORTED
Required argument

METADATA
Required argument

2.6 Database API

Database of results handling

class pyrcmip.database.**Database**(*root_dir*)
Bases: object

On-disk database handler for outputs from SCMs

get_out_filepath(*climate_model*, *variable*, *region*, *scenario*, *ensemble_member=None*)
Get filepath in which data has been saved

The filepath is the root directory joined with the other information provided. The filepath is also cleaned to remove spaces and special characters.

Parameters

- **climate_model** (*str*) – Climate model to retrieve data for
- **variable** (*str*) – Variable to retrieve data for
- **region** (*str*) – Region to retrieve data for
- **scenario** (*str*) – Scenario to retrieve data for
- **ensemble_member** (*str or None*) – Ensemble member to retrieve data for

Returns Path in which to save the data. If `ensemble_member` is `None` then it is not included in the filename.

Return type str

load_data(*climate_model*, *variable*, *region*, *scenario*)
Load data from the database

Parameters

- **climate_model** (*str*) – Climate model data to load
- **variable** (*str*) – Variable to load
- **region** (*str*) – Region to load
- **scenario** (*str*) – Scenario to load

Returns Loaded data

Return type obj: *scmdata.ScmRun*

load_model_reported()
Load all model reported results

Returns All model reported results

Return type pd.DataFrame

load_summary_tables()
Load all summary tables

Returns All summary tables

Return type pd.DataFrame

save_condensed_file(*scmrun*)
Save results which have multiple ensemble members

Parameters **scmrun** (scmdata.ScmRun) – Results to save in the database

Raises **AssertionError** – ensemble_member is not included in scmrun's metadata

save_model_reported(*res*, *key='all'*)
Save model reported data into the database

Parameters

- **res** (pd.DataFrame) – Model reported results to save. Should be the same format as the result of *pyrcmip.assessed_ranges.AssessedRanges.calculate_metric_from_results()*.
- **key** (*str*) – Identifier to use in the filename

Raises **AssertionError** – The columns of res are not as expected (i.e. {"value", "ensemble_member", "RCMIP name", "unit", "climate_model"}) or more than one climate model is included in res.

save_summary_table(*res*, *file_id*)
Save summary table

Parameters

- **res** (pd.DataFrame) – Summary table to save
- **file_id** (*str*) – Identifier to use in the filename

Raises **AssertionError** – Columns of res are not as expected (i.e. not equal to {"assessed_range_label", "assessed_range_value", "climate_model", "climate_model_value", "metric", "percentage_difference", "unit"})

save_to_database(*scmrun*)
Save a set of results to the database

The results are saved with one file for each ["climate_model", "variable", "region", "scenario", "ensemble_member"] combination.

Parameters **scmrun** (scmdata.ScmRun) – Results to save

2.7 Errors API

Custom errors defined within pyrcmip

exception pyrcmip.errors.**NoDataForMetricError**

Bases: ValueError

No data available to calculate the given metric

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pyrcmip.errors.**ProtocolConsistencyError**

Bases: ValueError

Inconsistency between input data and the RCMIP protocol

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.8 IO API

Input and output handling

pyrcmip.io.**ensure_dir_exists**(*fp*)

Ensure directory exists

Parameters **fp** (*str*) – Filepath of which to ensure the directory exists

pyrcmip.io.**read_results_submission**(*results*)

Read results submission

Parameters **results** (*str or list of str*) – Files to read in. All files to be read should be formatted as csv or xlsx files following the formatting defined in the template spreadsheet.

Returns Results read in from the submission(s)

Return type scmdata.ScmRun

pyrcmip.io.**read_submission_model_metadata**(*fp*)

Read the model metadata component of a submission

Parameters **fp** (*str*) – Filepath to read

Return type pd.DataFrame

pyrcmip.io.**read_submission_model_reported**(*fp*)

Read the model reported component of a submission

Parameters **fp** (*str*) – Filepath to read

Return type pd.DataFrame

pyrcmip.io.**temporary_file_to_upload**(*df*, *max_size=1024*, *compress=False*)
Create a gzipped temporary serialized version of a file to upload

Attempts to keep the file in memory until it exceeds *max_size*. The file is then stored on-disk and cleaned up at the end of the context.

The temporary location can be overriden using the *TMPDIR* environment variable as per https://docs.python.org/3/library/tempfile.html#tempfile.gettempdir

Parameters

- **df** (scmdata.ScmRun or pd.DataFrame) – Run to store
- **max_size** (*int or float*) – Max size in MB before file is temporarily streamed to disk. Defaults to 1GB

Returns Open file object ready to be streamed

Return type tempfile.SpooledTemporaryFile

2.9 Metric Calculations API

Metric calculations used in RCMIP

class pyrcmip.metric_calculations.**CalculatorAirborneFraction18501920**
Bases: *pyrcmip.metric_calculations.base.Calculator*

Calculator of the airborne fraction from 1850 to 1920

classmethod calculate_metric(*assessed_ranges*, *res_calc*, *norm_period*, *evaluation_period*, *unit*)
Calculate metric

Parameters

- **assessed_ranges** (*pyrcmip.assessed_ranges.AssessedRanges*) – Assessed ranges instance
- **res_calc** (scmdata.ScmRun) – Results from which the metric is to be derived
- **norm_period** (*list*) – Years to use for normalising the data before calculating the metric
- **evaluation_period** (*list*) – Years to use when evaluating the metric
- **unit** (*str*) – Unit in which the metric should be returned

Returns Metric values with other relevant model metadata

Return type pd.DataFrame

Raises

- ***NoDataForMetricError*** – No data is available to calculate the given metric
- **DimensionalityError** – The units of the data cannot be converted to the desired units or the units of the data are incompatible with the metric calculation

classmethod can_calculate_metric(*metric*)
Decide whether the input metric can be calculated or not

Parameters **metric** (*str*) – Metric to check

Returns If True, the metric can be calculated. Otherwise, it cannot.

Return type bool

class pyrcmip.metric_calculations.**CalculatorAirborneFraction18501990**
Bases: *pyrcmip.metric_calculations.airborne_fraction.CalculatorAirborneFraction18501920*

Calculator of the airborne fraction from 1850 to 1990

classmethod calculate_metric(*assessed_ranges*, *res_calc*, *norm_period*, *evaluation_period*, *unit*)
Calculate metric

Parameters

- **assessed_ranges** (*pyrcmip.assessed_ranges.AssessedRanges*) – Assessed ranges instance
- **res_calc** (scmdata.ScmRun) – Results from which the metric is to be derived
- **norm_period** (*list*) – Years to use for normalising the data before calculating the metric
- **evaluation_period** (*list*) – Years to use when evaluating the metric
- **unit** (*str*) – Unit in which the metric should be returned

Returns Metric values with other relevant model metadata

Return type pd.DataFrame

Raises

- ***NoDataForMetricError*** – No data is available to calculate the given metric
- **DimensionalityError** – The units of the data cannot be converted to the desired units or the units of the data are incompatible with the metric calculation

classmethod can_calculate_metric(*metric*)
Decide whether the input metric can be calculated or not

Parameters **metric** (*str*) – Metric to check

Returns If True, the metric can be calculated. Otherwise, it cannot.

Return type bool

class pyrcmip.metric_calculations.**CalculatorTCR**
Bases: pyrcmip.metric_calculations.base._CalculatorTCRTCREBase

Calculator of the transient climate response (TCR)

classmethod calculate_metric(*assessed_ranges*, *res_calc*, *norm_period*, *evaluation_period*, *unit*)
Calculate metric

Parameters

- **assessed_ranges** (*pyrcmip.assessed_ranges.AssessedRanges*) – Assessed ranges instance
- **res_calc** (scmdata.ScmRun) – Results from which the metric is to be derived
- **norm_period** (*list*) – Years to use for normalising the data before calculating the metric
- **evaluation_period** (*list*) – Years to use when evaluating the metric
- **unit** (*str*) – Unit in which the metric should be returned

Returns Metric values with other relevant model metadata

Return type pd.DataFrame

Raises

- ***NoDataForMetricError*** – No data is available to calculate the given metric

- **DimensionalityError** – The units of the data cannot be converted to the desired units or the units of the data are incompatible with the metric calculation

classmethod can_calculate_metric(*metric*)
Decide whether the input metric can be calculated or not

Parameters **metric** (*str*) – Metric to check

Returns If True, the metric can be calculated. Otherwise, it cannot.

Return type bool

class pyrcmip.metric_calculations.**CalculatorTCRE**
Bases: pyrcmip.metric_calculations.base._CalculatorTCRTCREBase

Calculator of the transient climate response to emissions (TCRE)

classmethod calculate_metric(*assessed_ranges*, *res_calc*, *norm_period*, *evaluation_period*, *unit*)
Calculate metric

Parameters

- **assessed_ranges** (*pyrcmip.assessed_ranges.AssessedRanges*) – Assessed ranges instance
- **res_calc** (scmdata.ScmRun) – Results from which the metric is to be derived
- **norm_period** (*list*) – Years to use for normalising the data before calculating the metric
- **evaluation_period** (*list*) – Years to use when evaluating the metric
- **unit** (*str*) – Unit in which the metric should be returned

Returns Metric values with other relevant model metadata

Return type pd.DataFrame

Raises

- ***NoDataForMetricError*** – No data is available to calculate the given metric
- **DimensionalityError** – The units of the data cannot be converted to the desired units or the units of the data are incompatible with the metric calculation

classmethod can_calculate_metric(*metric*)
Decide whether the input metric can be calculated or not

Parameters **metric** (*str*) – Metric to check

Returns If True, the metric can be calculated. Otherwise, it cannot.

Return type bool

Base class for metric calculations

class pyrcmip.metric_calculations.base.**Calculator**
Bases: abc.ABC

Base class for metric calculations

classmethod calculate_metric(*assessed_ranges*, *res_calc*, *norm_period*, *evaluation_period*, *unit*)
Calculate metric

Parameters

- **assessed_ranges** (*pyrcmip.assessed_ranges.AssessedRanges*) – Assessed ranges instance

- **res_calc** (scmdata.ScmRun) – Results from which the metric is to be derived
- **norm_period** (*list*) – Years to use for normalising the data before calculating the metric
- **evaluation_period** (*list*) – Years to use when evaluating the metric
- **unit** (*str*) – Unit in which the metric should be returned

Returns Metric values with other relevant model metadata

Return type pd.DataFrame

Raises

- ***NoDataForMetricError*** – No data is available to calculate the given metric
- **DimensionalityError** – The units of the data cannot be converted to the desired units or the units of the data are incompatible with the metric calculation

classmethod can_calculate_metric(*metric*)

Decide whether the input metric can be calculated or not

Parameters **metric** (*str*) – Metric to check

Returns If True, the metric can be calculated. Otherwise, it cannot.

Return type bool

2.10 Plotting API

Helpers and config for plotting

pyrcmip.plotting.**CLIMATE_MODEL_PALETTE = {'AR6 Prelim. FGD': 'tab:gray', 'HadCRUT.5.0.0.0': 'tab:gray', 'HadCRUT.5.0.0.0 (GMST)': 'tab:gray', 'MAGICC7': 'tab:orange', 'Raw CMIP6 multi-model ensemble': 'tab:green', 'assessed range': 'tab:blue', 'two_layer': 'tab:pink', 'von Shuckmann et al. 2020': 'tab:purple'}**

Colour palette used for plots coloured by climate model

Type dict

pyrcmip.plotting.**CMIP6_NAME = 'Raw CMIP6 multi-model ensemble'**

String used to represent the CMIP6 multi-model ensemble in plots

Type str

pyrcmip.plotting.**SCENARIO_PALETTE = {'historical': 'tab:gray', 'ssp119': array([0.1171875, 0.5859375, 0.515625]), 'ssp126': array([0.11328125, 0.19921875, 0.328125]), 'ssp245': array([0.9140625 , 0.86328125, 0.23828125]), 'ssp370': array([0.9453125 , 0.06640625, 0.06640625]), 'ssp370-lowNTCF': array([0.9453125 , 0.06640625, 0.06640625]), 'ssp434': array([0.38671875, 0.73828125, 0.89453125]), 'ssp460': array([0.90625 , 0.921875 , 0.19140625]), 'ssp534-over': array([0.6015625 , 0.42578125, 0.78515625]), 'ssp585': array([0.515625 , 0.04296875, 0.1328125])}**

Colour palette used for plots coloured by scenario

Type dict

2.11 Stats API

Statistics required for RCMIP analysis

pyrcmip.stats.**get_skewed_normal**(*median*, *lower*, *upper*, *conf*, *input_data*)

Get skewed normal distribution matching the inputs

Parameters

- **median** (*float*) – Median of the output distribution
- **lower** (*float*) – Lower bound of the confidence interval
- **upper** (*float*) – Upper bound of the confidence interval
- **conf** (*float*) – Confidence associated with the interval [lower, upper] e.g. 0.66 would mean that [lower, upper] defines the 66% confidence range
- **input_data** (np.ndarray) – Points from the derived distribution to return. For each point, Y, in `input_data`, we determine the value at which a cumulative probability of Y is achieved. As a result, all values in `input_data` must be in the range [0, 1]. Hence if you want a random sample from the derived skewed normal, simply make `input_data` equal to a random sample of the uniform distribution [0, 1]

Returns Points sampled from the derived skewed normal distribution based on `input_data`

Return type np.ndarray

pyrcmip.stats.**sample_multivariate_skewed_normal**(*configuration*, *size*, *cor=None*)

Sample multi-variate skewed normal distribution

Following [Meinshausen et al. (2009)](https://doi.org/10.1038/nature08017), a skewed normal is defined as follows: "A distribution X, is skewed normal, if
$log(X + C)$, where C is a constant, is a normal distribution with variance
$sigma^2$ and mean
mu". The skewed normal allows us to create distributions which match arbitrary median and likely ranges (with associated confidence as a percentage), as is often used by the IPCC. A multivariate skewed normal is constructed such that each marginal distribution is a skewed normal and the overall distribution can have a non-identity correlation matrix.

Parameters

- **configuration** (pd.DataFrame) – Configuration for the sampling. Each column must represent a dimension to be sampled. The rows must be `["median", "upper", "lower", "conf"]`. The rows represent the characteristics of each marginal distribution. The median is the median of each marginal distribution. `"conf"` represents the confidence associated with each of the intervals defined by `"lower"` and `"upper"` (e.g. 0.66 would mean that [lower, upper] defines the 66% confidence range).
- **size** (*int*) – Number of points to sample from the multivariate skewed normal
- **cor** (*array[float]*) – Correlation matrix between different dimensions in *configuration*. *cor* must be a square, symmetric matrix with size n x n, where n is the number of columns in `configuration`. The element in row i and column j represents the correlation between the dimension in column i of `configuration` and the dimension in column j of `configuration`. The correlations must be normalised i.e. the maximum value of each element is 1 and the minimum value is -1. As a result of the normalisation, the diagonals of `cor` must all be equal to 1.

Returns Points sampled from the derived multivariate skewed normal distribution. Each row is a sampled point (so there will be `size` row in the output). The columns match the columns in `configuration`.

Return type `pd.DataFrame`

Raises

- `ValueError` – `configuration` contains any nans, distribution ranges are incorrectly specified (e.g. medians > upper ends of the intervals) or the correlation matrix is incorrectly normalised.
- `KeyError` – `configuration` is missing one of the rows: `["median", "upper", "lower", "conf"]`

2.12 Validate API

Validation of RCMIP submissions

pyrcmip.validate.**convert_units_to_rcmip_units**(*submission*, *protocol_variables*)
Convert units to RCMIP units

Parameters

- **submission** (`scmdata.ScmRun`) – Submission to convert
- **protocol_variables** (`pd.DataFrame`) – Variables and units as defined by the RCMIP protocol

Returns Submission with units converted to RCMIP units

Return type `scmdata.ScmRun`

Raises *ProtocolConsistencyError* – Units could not be converted to RCMIP units

pyrcmip.validate.**validate_regions**(*regions_to_check*, *protocol_regions*)
Validate regions against regions in the RCMIP protocol

Parameters

- **regions_to_check** (*list-like*) – Regions to check
- **protocol_regions** (*list-like*) – Regions in the RCMIP protocol

Raises *ProtocolConsistencyError* – `regions_to_check` contains regions not included in `protocol_regions`

pyrcmip.validate.**validate_scenarios**(*scenarios_to_check*, *protocol_scenarios*)
Validate scenarios against scenarios in the RCMIP protocol

Parameters

- **scenarios_to_check** (*list-like*) – Scenarios to check
- **protocol_scenarios** (*list-like*) – Scenarios in the RCMIP protocol

Raises *ProtocolConsistencyError* – `scenarios_to_check` contains scenarios not included in `protocol_scenarios`

pyrcmip.validate.**validate_submission**(*submission*, *protocol=None*)
Validate that an RCMIP submission complies with the required data format

Parameters

- **submission** (scmdata.ScmRun) – Data to validate
- **protocol** (*str*) – Data file containing the RCMIP protocol against which to validate the data. If None, the submission template will be loaded from pyrcmip/data/rcmip-data-submission-template-v4-0-0.xlsx.

Returns Input data, converted to match RCMIP units

Return type scmdata.ScmRun

Raises *ProtocolConsistencyError* – The data is not consistent with the protocol

pyrcmip.validate.**validate_submission_bundle**(*timeseries*, *model_reported*, *metadata*, *protocol=None*)

Validate that an RCMIP submission bundle complies with the required formats

Parameters

- **timeseries** (scmdata.ScmRun) – Timeseries to validate
- **model_reported** (pd.DataFrame) – Model reported metrics
- **metadata** (pd.DataFrame) – Model metadata
- **protocol** (*str*) – Data file containing the RCMIP protocol against which to validate the timeseries. If None, the submission template will be loaded from pyrcmip/data/rcmip-data-submission-template-v4-0-0.xlsx.

Returns Validated timeseries, model reported metrics and model metadata

Return type (scmdata.ScmRun, pd.DataFrame, pd.DataFrame)

Raises

- *ProtocolConsistencyError* – The submission bundle is not consistent with the RCMIP protocol
- ValueError – A value for climate_model is found in timeseries or model_reported but isn't found in the climate_model column of metadata.

pyrcmip.validate.**validate_submission_model_meta**(*inp*)

Validate a submission's metadata

Parameters **inp** (pd.DataFrame) – Metadata submission to validate

Returns Validated metadata submission

Return type pd.DataFrame

Raises *ProtocolConsistencyError* – The columns of res are not as expected (i.e. {"climate_model", "climate_model_name", "climate_model_version", "climate_model_configuration_label", "climate_model_configuration_description", "project", "name_of_person", "literature_reference"}).

pyrcmip.validate.**validate_submission_model_reported_metrics**(*inp*)

Validate a submission of model reported metrics

Parameters **inp** (pd.DataFrame) – Input to validate

Returns Validated input

Return type pd.DataFrame

Raises *ProtocolConsistencyError* – The columns of res are not as expected (i.e. `{"value", "ensemble_member", "RCMIP name", "unit", "climate_model"}`), more than one climate model is included in `res`, the `ensemble_member` column is not integers, an unrecognised metric is provided or the provided unit is not compatible with RCMIP.

`pyrcmip.validate.`**validate_variables**(*vars_to_check*, *protocol_variables*)

Validate variables against variables in the RCMIP protocol

Parameters

- **vars_to_check** (*list-like*) – Variables to check
- **protocol_variables** (*list-like*) – Variables in the RCMIP protocol

Raises *ProtocolConsistencyError* – `vars_to_check` contains variables not included in `protocol_variables`

2.13 Changelog

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

The changes listed in this file are categorised as follows:

- Added: new features
- Changed: changes in existing functionality
- Deprecated: soon-to-be removed features
- Removed: now removed features
- Fixed: any bug fixes
- Security: in case of vulnerabilities.

2.13.1 master

Added

- (!32) *pyrcmip.stats.sample_multivariate_skewed_normal()* to allow sampling of a multi-variate skewed normal distribution

2.13.2 v0.5.2 - 2022-02-08

Changed

- (!31) Move `docutils` to the `docs` extra requirements as it is only needed for building the documentation

2.13.3 v0.5.1 - 2021-08-18

Added

- (!30) *pyrcmip.metric_calculations.CalculatorAirborneFraction18501920* and *pyrcmip.metric_calculations.CalculatorAirborneFraction18501990* for calculating airboren fraction
- (!28) Scripts for uploading the RCMIP protocol to Zenodo

Changed

- (!29) Compare the ETag value from S3 against the md5sum from zenodo for the protocol data. This requires the protocol data to be uploaded as a single part as for multipart uploads the Etag != md5sum of the uploaded file

2.13.4 v0.5.0 - 2021-02-23

Changed

- (!27) Use `openpyxl` rather than `xlrd` as excel engine
- (!27) Upgrade to `pyjwt>=2`
- (!27) Upgrade to `scmdata>=0.7.3`

Fixed

- (!26) Remove rogue cells in data submission template (new template released as v5-1-0)

2.13.5 v0.4.1 - 2020-09-14

Fixed

- (!25) Usage of old seaborn API in plotting and broken unit check

2.13.6 v0.4.0 - 2020-09-13

Added

- (!23) Documentation and tests for `pyrcmip.assessed_ranges` and `pyrcmip.metric_calculations`
- (!22) Add support for downloading submitted data

Changed

- (!23) *pyrcmip.database.Database.load_data()* now requires a `climate_model` argument
- (!23) *pyrcmip.database.Database.save_summary_table()* now expects an `"RCMIP name"` column, rather than `"metric"`
- (!23) Metric calculations now use the *pyrcmip.metric_calculations.base.Calculator*
- (!24) Pin test dependency `moto==1.3.14`
- (!21) Timeseries submissions must include an `ensemble_member` column

Removed

- (!23) `pyrcmip.database.time_mean()`

2.13.7 v0.3.0 - 2020-09-02

Added

- (!19) Clearer error message if the timeseries submission doesn't contain `climate_model` or `unit` metadata
- (!17) Update create-token script to allow for rotating of tokens

Changed

- (!20) Each input timeseries is now individually validated and uploaded when using the cli

2.13.8 v0.2.1 - 2020-09-01

Added

- (!18) Clarification that pyrcmip only supports Python 3.6+
- (!18) Add support from submission from gzipped csv
- (!16) Add the ability to specify multiple timeseries files via the CLI. Closes (#3)

2.13.9 v0.2.0 - 2020-08-17

Added

- (!14) Check if the templates have changed during CI
- (!12) Add readthedocs configuration
- (!10) Documentation of submission process
- (!6) Skeleton of data processing, including illustrative model submission and processing pipeline
- (!5) Basic docs

Changed

- (!13) Fix broken documentation on readthedocs
- (!8) Upload data, metadata and model reported values together
- (!7) Require validation before uploading
- (!6) Submissions now require three parts: timeseries, model reported and metadata rather than only just one
- (!4) Require scmdata >= 0.6.1

2.13.10 v0.1.1 - 2020-07-09

Changed

- Fixed readme

2.13.11 v0.1.0 - 2020-07-09

Added

- CLI framework
- Basic checks

CHAPTER

THREE

INDEX

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Z. R. J. Nicholls, M. Meinshausen, J. Lewis, R. Gieseke, D. Dommenget, K. Dorheim, C.-S. Fan, J. S. Fuglestvedt, T. Gasser, U. Golüke, P. Goodwin, E. Kriegler, N. J. Leach, D. Marchegiani, Y. Quilcaille, B. H. Samset, M. Sandstad, A. N. Shiklomanov, R. B. Skeie, C. J. Smith, K. Tanaka, J. Tsutsui, and Z. Xie. Reduced complexity model intercomparison project phase 1: protocol, results and initial observations. *Geoscientific Model Development Discussions*, 2020:1–33, 2020. URL: https://gmd.copernicus.org/preprints/gmd-2019-375/, doi:10.5194/gmd-2019-375.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

G

H

L

M

N

O

P

R

S

T

V

W