one_pass

Release 0.1

Katherine Grayson

Aug 02, 2023

CONTENTS:

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 One-pass method . . . 3

2 Installation **5**
- 2.1 Pre-requisites . . . 5
- 2.2 Pip . . . 5
- 2.3 Git . . . 5
- 2.4 Environments with Conda/Mamba . . . 6
- 2.5 Installation with Docker . . . 6
- 2.6 Installing environment using containers . . . 7

3 Getting started **9**
- 3.1 Basic Concept . . . 9
- 3.2 Running in Python . . . 9
- 3.3 A note on the Output . . . 10

4 The config.yml file **11**
- 4.1 Statistics . . . 12
  - 4.1.1 Mean . . . 12
  - 4.1.2 Variance . . . 12
  - 4.1.3 Standard Deviation . . . 12
  - 4.1.4 Minimum . . . 13
  - 4.1.5 Maximum . . . 13
  - 4.1.6 Threshold Exceedance . . . 13
  - 4.1.7 Percentile . . . 13
  - 4.1.8 Raw . . . 13
  - 4.1.9 Bias-Correction . . . 13
- 4.2 Frequncies . . . 14
  - 4.2.1 Statistic Frequency . . . 14
  - 4.2.2 Output Frequency . . . 14
- 4.3 Time step . . . 15
- 4.4 Variable . . . 15
- 4.5 Save . . . 15
- 4.6 Checkpoint . . . 15
- 4.7 Checkpoint Filepath . . . 15
- 4.8 Save Filepath . . . 15

5 Tests **17**
- 5.1 Running Tests Locally . . . 17

6 Examples and Tutorials **19**
6.1 Jupyter Notebooks . . . 19
6.2 Python script . . . 19

7 Contributing to one_pass **21**
7.1 Reporting Bugs . . . 21
7.2 Suggesting Features . . . 21
7.3 Contributing Code . . . 21
7.4 Documentation and Tutorials . . . 22

8 References and Acknowledgements **23**
8.1 Citing one_pass . . . 23
8.2 Acknowledgments . . . 23

One_pass is a python library developed for the Destination Earth programme as part of the Climate Digital Twin (Climate DT). The library works on streamed climate data, performing intelligent data reduction techniques in order to provide statistics summaries as requested by the user.

Note: This project is under active development

CHAPTER

ONE

INTRODUCTION

1.1 Background

Currently, impact modelling and downstream applications use pre-existing climate model output archives (e.g., CORDEX and CMIP archives) to create information for climate adaptation policies and socio-economic decisions. Climate DT proposes a novel way of running user applications close to the Earth system models by streaming output variables directly to applications instead of storing a limited set of essential climate variables on disk. This novel way of post-processing model outputs as they become available allows an analysis of high-frequency and high-resolution variables that would be too costly to store on disk at the resolutions envisaged by climate DT.

The streaming of climate model outputs will enable applications to access a larger set of variables at unprecedented resolutions for downstream impact modelling at global scales. This approach goes beyond the classical paradigm where adaptation studies have to use variables from archives with only a limited predefined number of variables and at coarse resolutions. However, this streaming of the output variables creates the one-pass problem: the algorithms that compute summaries, diagnostics or derived quantities do not have access anymore to the whole time series, but just receive the values incrementally every time that the model outputs new time steps. The specific computations required by downstream applications can cover a wide range of topics from computing time aggregations or distribution percentiles, bias adjusting, deriving established climate indices or computing sectoral indicators. Any downstream application that is part of the Climate DT workflow will have to use the one-pass algorithm library, detailed in this documentation, to be able to run in streaming mode.

1.2 One-pass method

One-pass algorithms (also known as single-pass algorithms) are streaming algorithms which read the data input only one chunk at a time. This is done by handelling items in order (without buffering); reading and processing a block of data and moving the result into an output buffer before before moving onto the next block. This is described mathematically by:

$$\begin{array}{ccccccc} x_1 & & x_2 & & x_3 & \dots & x_n \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ S_1 = g(x_1) & \rightarrow & S_2 = g(S_1, x_2) & \rightarrow & S_3 = g(S_2, x_3) & \rightarrow & S_n = g(S_{n-1}, x_n) \end{array}$$

If we let $f(X_n) = \{x_1, x_2, ..., x_n\}$ represent the data set, then $S_n = f(X_n) = g(S_{n-1}, x_n)$, where f is a function that acts on the whole data set and computes the summary S_n in one go using all values and g is a 'one pass' function that updates a previous summary S_{n-1} with a new value x_n. The summaries S_n require less memory than the full dataset X_n.

CHAPTER

TWO

INSTALLATION

The one_pass library is hosted in Git, at the BSC GitLab Earth Sciences repository. The package is avalible through pip or git. We recommend using Mamba, a package manager for conda-forge, for setting up the python environment.

2.1 Pre-requisites

Before installing one_pass, ensure you have the following pre-requisites installed:

- Git
- Mamba or Conda
- Python

2.2 Pip

To install one_pass with pip, execute the following:

```
pip install git+https://earth.bsc.es/gitlab/digital-twins/de_340/one_pass.git@main
```

2.3 Git

You can clone the directory directly via Git using:

```
git clone https://earth.bsc.es/gitlab/digital-twins/de_340/one_pass.git
```

This will clone the main branch of the repository.

2.4 Environments with Conda/Mamba

Navigate to the one_pass directory:

```
cd one_pass
```

Create a new mamba environment with the required packages using the environment.yml file:

```
mamba env create -f environment.yml
```

Finally activate the new environment by running:

```
conda activate env_opa
```

2.5 Installation with Docker

When working on some HPC platforms it is not possible to directly use python conda environments. In this case we suggest running the one_pass environment inside a Docker container and copying to the HPC system as a Singularity container. To use this method, you first need both Docker (instructions) and Singularity (instructions) installed on your **local** machine.

1. First clone the git reposity to your local machine and navigate into the root directory.
2. Create the docker image using the command

```
docker build -t one_pass:latest .
```

If you don't have permissions, add `sudo` to the beginning of the line. This will create your docker image.

3. To enter the docker image run

```
docker run --rm -ti one_pass:latest /bin/bash
```

Again, if you don't have permissions, `sudo` to the beginning of the line. You are now running your Docker container with the environment `(env_opa)`. To exit the container use ctrl + d.

4. The next step is to create a singularity container so that you can move this docker image to any HPC system. To create the singularity file (you must have singularity installed on your local machine) run:

```
singularity build one_pass_singularity.sif docker-daemon://one_pass:latest
```

Again, if you don't have permissions add `sudo` at the beginning of the line.

5. To enter the singularity container:

```
singularity shell one_pass_singularity.sif
```

6. Then to activate the environment:

```
source /usr/local/bin/_activate_current_env.sh
```

You now have your singularity container containing the one_pass environment that can be passed to any HPC machine.

7. To copy the singularity container to LUMI you can run:

```
scp -r one_pass_singularity.sif lumi:.
```

This will pass your singularity container to LUMI (or any other machine you want). You can then enter the singularity container using step 5 and 6 above.

2.6 Installing environment using containers

The other option on LUMI is to use conda containers, as described here.

CHAPTER

THREE

GETTING STARTED

3.1 Basic Concept

The one_pass python package is designed to act on streamed climate data, generated from the Climate Digitial Twin's (DT) climate models, and extracted from the GSV interface. By the end of Phase 1, the calls to both the GSV interface (for data retrival) and one_pass (for summary statistics) will be configured by the workflow manager Autosubmit. For information on how to configure the workflow contact work package (WP) 8. This documentation covers how to use the one_pass package and demonstrates its usage with 'fake' streaming.

The one_pass package has been built around a few core concepts:

- The temporal aggreation of data through statistical summaries over different time frequncies.
- The output of raw data as a temporary data storage.
- Passing both raw data and data percentiles for use in the bias-correction layer.

Parallel processing is currently implemented for most statistics through dask to speed up computation.

3.2 Running in Python

As with any python library the first step is to import the package

```
from one_pass.opa import Opa
```

After importing the library only two lines of python code are required to run

```
opa_stat = Opa(config.yml) # initalise some statistic using the config.yml file
dm = opa_stat.compute(data) # pass some data to compute
```

Above, all the details of the requested statistic are given by the config.yml file, explained in *The config.yml file*. The incoming `data` is an xArray object containing some climate data over a given (structured or unstructured) grid and a certain temporal period. The second line will be run multiple times, as the data stream progress, continously providing new data to the Opa class. Once sufficient data has been passed to the Opa class (enough to complete the requested statistic), `dm` will return the output of the summary statistic. For detailed examples and tutorials, refer to the *Examples and Tutorials* section which contain example Jupyter notebooks.

3.3 A note on the Output

The output, `dm` written above, will be an xr.dataSet. The dimensions will the same as the original dimensions of the input data, apart from the time dimension. The length of time dimension will be one, unless you set `output_freq` greater than `stat_freq`. See the *The config.yml file* for details.

The timestamp on the time dimension will correspond to the time stamp of the first piece of data that contributed to that statistic.

All of the original metaData included in the dataSet will be present in the final dataSet with a new attribute corresponding to details of the one pass algorithm. There will only be one variable in the final dataSet, as the one_pass only processes one variable at a time. The name of the variable will be unchanged.

Note: There is currently a problem with GRIB variable names that start with a number, as when saving they are saved with a / in front. This issue is being worked on.

The final output from the one_pass will only be passed when sufficient data has been streamed to the Opa class to complete the statistic. If you do not set `"save" : True` then this will only be output in memory and will be over written as new data is passed.

CHAPTER

FOUR

THE CONFIG.YML FILE

As discussed in *Getting started*, all of the one_pass configuration is set by a seperate configuration file, called config.yml. In this file you provide the details of the statistic you would like. The config.yml looks like:

```
stat: "mean"
percentile_list: None
thresh_exceed: None
stat_freq: "daily"
output_freq: "daily"
time_step: 60
param: "uas"
save: True
checkpoint: True
checkpoint_filepath: "/file/path/to/checkpoint/"
out_filepath: "/file/path/to/save/"
```

All of the above key:value pairs must be present in the config.yml, even if not required for your requested statistic. In this case, keep the output as `None`.

The same information can also be passed directly in python as a dictionary:

```
pass_dic = {"stat" : "mean",
"percentile_list" : None,
"thresh_exceed" : None,
"stat_freq": "daily",
"output_freq": "daily",
"time_step": 60,
"variable": "uas",
"save": True,
"checkpoint": True,
"checkpoint_filepath": "/file/path/to/checkpoint/",
"out_filepath": "/file/path/to/save/"}
```

The functionality of all the keys in the config.yml are outlined below.

Note: Be careful about spelling in the config file, it matters.

4.1 Statistics

The one_pass package supports the following options for `stat` :

```
"mean", "std", "var", "thresh_exceed", "min", "max", "percentile", "raw", "bias_
↪correction"
```

4.1.1 Mean

The mean statistic calculates the arithmatic mean over the requested temporal frequency, using the following:

$$\bar{x}_n = g(S_{n-1}, x_n) = g(\bar{x}_{n-1}, x_n) = \bar{x}_{n-1} + w\left(\frac{x_n - \bar{x}_{n-1}}{n}\right)$$

where n is the current number of data samples or time steps (count) that have been passed, including the incoming data, $\bar{x}_n$ is the updated mean, $\bar{x}_{n-1}$ is the mean of the previous data and w is the weight of the incoming data chunk (the number of time stamps). In the case where the incoming data has more than one time step, so $w > 1$, x_n is the temporal mean over the incoming data computed with numpy.

4.1.2 Variance

The variance (written as `"var"` in the config.yml) is calculated for the incoming data stream, over the requested temporal frequency, by updating two estimates iteratively.

$$\text{Let } M_{2,n} = \sum_{i=1}^{n}(x_i - \bar{x}_n)^2 \text{ then:}$$

$$M_{2,n} = g(S_{n-1}, x_n) = g(M_{2,n-1}, x_n) = M_{2,n-1} + w(x_i - \bar{x}_{n-1})(x_i - \bar{x}_n)$$

where $\bar{x}_n$ is given by the algorithm of the mean shown above. At the end of the iterative process (when the last value is given to complete the statistic)

$$\text{var}(X_n) = \frac{M_{2,n}}{n-1}$$

In the case where the incoming data has more than one time step ($w > 1$), $M_{2,n}$ is updated by

$$M_{2,n} = M_{2,n-w} + M_{2,w} + \frac{\sqrt{(\bar{x}_{n-w} - \bar{x}_w)}(w(n-w))}{n}$$

where $M_{2,n-w}$ is sum of the squared differences of the previously seen data, $M_{2,w}$ is the sum of the squared differences over the incoming data block (of weight w) and $\bar{x}_{n-w}$ and $\bar{x}_w$ are the means over those same periods respectively. See S. Mastelini for details.

4.1.3 Standard Deviation

The standard deviation (written as `"std"`) calculates the standard deviation of the incoming data stream over the requested temporal frequency, by taking the square root of the variance:

$$\sqrt{\text{var}(X_n)}$$

4.1.4 Minimum

The minimum value (written as `"min"`) is given by:

$$\min_n = g(S_{n-1}, x_n) = g(\min_{n-1}, x_n) = \text{ if } (x_n < \min_{n-1}) \text{ then } \min_{n-1} = x_n$$

4.1.5 Maximum

The maximum value (written as `"max"`) is given by:

$$\max_n = g(S_{n-1}, x_n) = g(\max_{n-1}, x_n) = \text{ if } (x_n < \max_{n-1}) \text{ then } \max_{n-1} = x_n$$

4.1.6 Threshold Exceedance

The threshold exceedance statistic (written as `"stat" : "thresh_exceed"`) requires a value for the key:value pair `thresh_exceed: some_value`, where `some_value` is the threshold for your chosen variable. The output of this statistic is the number of times that threshold is exceeded. It is calcuated by:

$$\text{exc}_n = g(S_{n-1}, x_n) = g(\text{exc}_{n-1}, x_n) = \text{ if } (x_n > \text{thresh exceed}) = \text{exc}_{n-1} = \text{exc}_{n-1} + 1$$

The variable in the final xr.dataSet output now corresponds to the number of times the data exceeded the threshold.

4.1.7 Percentile

The `"percentile"` statistic requires a value for the key:value pair `"percentile_list" : [0.2, 0.5]` where the list contains the requested percentiles between the values of `[0,1]`. The list can be as long as you like but must be comma seperated. If you want the whole distribution, so all the percentiles from `[0,1]`, put `["all"]`, including the brackets `[]`. The number of variables in the produced dataSet will correspond to the number of requested percentiles. If you request the full distribution, this will correspond to 101 variables, one for each percentile including 0 and 1. This statistic makes use of the T-Digest algorithm using the python implementation.

Currently for the TDigests we have set a compression parameter at 25 (reduced from the default of 100), as we have to consider memory contraints. This value needs optimising.

4.1.8 Raw

The `"raw"` statistic does not compute any statistical summaries on the incoming data, it simply outputs the raw data as it is passed. The only way it will modify the data is if a dataSet is passed with many climate variables, it will extract the variable requested and produce a dataSet containing only that variable. This option is included to act as a temporary data buffer for some use case applications.

4.1.9 Bias-Correction

Another layer to the one-pass library is the bias-correction. This package is being developed seperately from the one_pass but will make use of the outputs from the one_pass package. Specifically if you set `"stat" : "bias_correction"` you will recieve three outputs, as opposed to just one.

1. Daily aggregations of the incoming data (either daily means or summations depending on the variable) as netCDF
2. The raw daily data as netCDF

3. A pickle file containing tDigest objects. There will be one file for each month, and the digests will be udpated with the daily time aggregations (means or summations) for that month. The months will be accumulated, for example, the month 01 file will contain data from all the years the model has convered.

When using this statistic, make sure to set `"stat_freq" : "daily"` and `"output_freq" : "daily"`.

Note: The bias-correction statistic has been created specifically to pass data to the bias_correction package. It does not provide bias corrected data itself.

4.2 Frequncies

4.2.1 Statistic Frequency

The statistic frequency (written as `"stat_freq"`) can take the following options:

```
"hourly", "3hourly", "6hourly", "12hourly", "daily", "weekly", "monthly", "3monthly",
↪"annually", "continuous"
```

Each option defines the period over which you would like the statistic computed. For the frequencies `"weekly"`, `"monthly"`, `"annually"`, the one_pass package uses the gregorian calendar, e.g. `"annually"` will only start accumulating data if the first piece of data provided corresponds to the 1st January, it will not compute a random 365 days starting on any random date. If the data stream starts half way through the year, the one_pass will simply pass over the incoming data until it reaches the beginning of the new year. For `"monthly"` leap years are included. `"weekly"` will run from Monday - Sunday.

The option of `"continuous"`, will start from the first piece of data that is provided and will continously update the statistic as new data is provided.

4.2.2 Output Frequency

The output frequency option (written as `"output_freq"`) takes the same input options as `"stat_freq"`. This option defines the frequency you want to output (or save) the dataSet containing your statistic. If you set `"output_freq"` the same as `"stat_freq"` (which is the standard output) the dataSet produced by the one_pass will have a time dimension of length 1, corresponding the summary statistic reuqested by `"stat_freq"`. If, however, if you have requested `"stat_freq": "hourly"` but you don't want an output file for every hour, set `"output_freq": "daily"` and you will have a dataSet with a time dimension of length 24, corresponding to 24 hourly statistical summaries in one file.

Note if you set `"stat_freq" = "continuous"` you must set `"output_freq"` to the frequnecy at which the one pass outputs the current status of the statistic. **Do not** also set `"output_freq" = "continuous"`.

4.3 Time step

The option `"time_step"` is the the time step of your incoming data in **minutes**. Currently this is also given in the configuration file for the GSV, we are aware this repeated data. Soon these configuration files will be combined however for now, it needs to be set here. Eventually, this information will be provided by the streamed climate data.

4.4 Variable

The climate variable you want to compute your statistic on. If you provide the one_pass with a dataArray, you do not need to set this, however if you provide a dataSet then this is required.

Note the one_pass can only work with one variable at a time, multiple variables will be handled by different calls in the workflow.

4.5 Save

Either `True` or `False`. If you set this to `False`, the final statistc will only be output in memory and will get overwritten when a new statistic is avaliable. It is recommended to set this to `True` and a netCDF file will be written (in the `"out_filepath"`) when the statistic is completed.

4.6 Checkpoint

Either `True` or `False`. This defines if you want to write intermediate checkpoint files as the one_pass is provided new data. If true, a checkpoint file will be written for every new chunk of incoming data. If set to `False` the rolling statistic will only be stored in memory and will be lost of if the programme crashes. It will also allow for the statistics to be rolled back in time if the model crashes. It is highly recommended to set this to `True`.

4.7 Checkpoint Filepath

This is the file path, **NOT including the file name**, of your checkpoint files. The name of the checkpoint file will be dynamically created.

4.8 Save Filepath

`"out_filepath"` is the file path to where you want the final netCDF files to be written. The name of the file is dynamically created inside the one_pass as it contains the details of the requested statistic.

CHAPTER

FIVE

TESTS

The opa package uses pytest, a common python testing framework, for both writing and running tests. The Gitlab also uses Continuous Integration/Continuous Development (CI/CD) which runs the testing suite whenever changes are pushed to the reposity. The CI/CD pipeline currently uses a small test data file located in the repository for testing. In the future it will use data located on different HPC platforms to test the functionality and accuracy of the code. Code coverage is also impelmented in the CI/CD.

5.1 Running Tests Locally

To run the testing suite locally you must have pytest loaded in your active environment, as well as the dependencies for the one_pass package.

First navigate to the tests folder:

```
cd tests
```

Then simply run the following command:

```
pytest
```

This will run all of the tests. To simply run one of the tests, run:

```
pytest test_accuracy.py
```

The tests cover the accuracy of all the implemented statistics, over different temporal periods and with different time length chunks. They also cover functionality, error handeling and checks on the config file.

CHAPTER

SIX

EXAMPLES AND TUTORIALS

6.1 Jupyter Notebooks

We have provided a Jupyter notebook called "example_notebook_disk_data.ipynb" to show case the functionalities of the one-pass package. This notebook covers the all the different statistics currently avaliable with the one-pass package over different time scales using a small data set saved to disk. It can be found in the examples_and_notebooks folder.

6.2 Python script

In the same "examples and notebooks" folder there is also a 'wrapper.py' script, which provides a short example of how the one-pass package should be cofigured in python.

CHAPTER

SEVEN

CONTRIBUTING TO ONE_PASS

We welcome contributions to the one_pass project! Here you will find instructions for reporting bugs, suggesting new features, or contributing code.

7.1 Reporting Bugs

If you encounter any bugs or issues while using one_pass, the best thing to do is to report them on the project's issue tracker. Make sure to provide a detailed description of the issue, including steps to reproduce and error messages.

7.2 Suggesting Features

We would also love to hear from you if you have an idea for a new feature or enhancement in the one_pass library. To suggest a feature, open an issue and provide a detailed description of the proposed feature, including use cases, benefits, and potential challenges.

7.3 Contributing Code

To contribute code to one_pass, follow these general steps:

1. Fork the one_pass repository on GitLab.
2. Clone the forked repository to your local machine.
3. Create a new branch for your feature or bugfix (e.g., `git checkout -b my-feature`).
4. Make sure your changes pass the tests.
5. Commit your changes and push them to your forked repository.
6. Create a pull request on the one_pass repository, describing your changes and referencing any related issues.

Please note that all contributed codes must be licensed under the same terms as one_pass.

7.4 Documentation and Tutorials

We also welcome contributions to the one_pass documentation and tutorials. If you have suggestions for improvements, want to help maintain the documentation, or have ideas for new tutorials, please create an issue on the GitHub repository or submit a pull request with your proposed changes.

CHAPTER

EIGHT

REFERENCES AND ACKNOWLEDGEMENTS

8.1 Citing one_pass

If you use the one_pass package in your research or publications, please cite using the following format:

```
@software{one_pass,
  author       = {Katherine Grayson, Stephan Thober, et al.},
  title        = {One-pass: intelligent data reduction techniques for streamed climate␣
↪data},
  year         = {20XX},
  publisher    = {GitLab},
  journal      = {Barcelona Supercomputing Center, Earth Sciences GitLab},
  howpublished = {\url{https://earth.bsc.es/gitlab/digital-twins/de_340/one_pass/main}},
}
```

8.2 Acknowledgments

We would like to thank the many contributors who have helped develop, test, and maintain this package, along with their highly valued advice.

Development of one_pass is supported by European Union Contract *DE_340_CSC - Destination Earth Programme Climate Adaptation Digital Twin (Climate DT)*.

- search