one_pass

Release 0.4

Katherine Grayson

Aug 29, 2023

CONTENTS:

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 statistical 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 can make use of the one_pass algorithm library, detailed in this documentation, to help them 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:

$$\begin{tikzcd}
x_1 \arrow[d] & x_2 \arrow[d] & x_3 \arrow[d] & \dots \quad x_n \arrow[d] \\
S_1 = g(x_1) \arrow[r] & S_2 = g(S_1, x_2) \arrow[r] & S_3 = g(S_2, x_3) \arrow[r] & S_n = g(S_{n-1}, x_n).
\end{tikzcd}$$

If we let $X_n = \{x_1, x_2, ..., x_n\}$ represent the data set at time n, 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, and v0.4 can be found on the CSC gitHub. 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. It is not necessary however for the one_pass to work with Climate DT data and be configured by Autosubmit; it is a standalone package that can be used with any streamed climate data. 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 aggregation of data through statistical summaries over different time frequencies.
- 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 data request, located here in the `config.yml` file. How to configure the data request is explained in *The data request*. 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 details of output file naming conventions and structure of the output, refer to *A note on the output*. For detailed examples and tutorials, refer to the *Examples and Tutorials* section which contain example Jupyter notebooks.

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.

CHAPTER FOUR

THE DATA REQUEST

As discussed in *Getting started*, all of the one_pass configuration is set, either by a separate configuration file (that we call config.yml), or passed as a python dictionary. Both contain a set list of specific key:value pairs. Using these pairs 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, 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 are outlined below.

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

4.1 Statistics

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

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

We use the folling definitions in the mathematical descriptions of the algorithms below:

- n is the current number of data samples (time stamps) passed to the statistic
- w is the weight of incoming data chunk (number of time stamps)
- $X_n = \{x_1, x_2, ..., x_n\}$ represents the full data set up to time *n*
- $X_w = \{x_{n-w+1}, . . . , x_n\}$, is the incoming data chunk of weight w
- S_{n-w} is the summary of the statistic before the new chunk at time $n - w$.
- g is a 'one_pass' function that updates the previous summary S_{n-w} with then new incoming data X_w

In the case where the incoming data has only one time step ($w = 1$), X_w, reduces to x_n.

4.1.1 Summation

The summation statistic (written as `"sum"` in the statistic request) is calculated by:

$$\sum_{i=1}^{n} X_n = g(S_{n-w}, X_w) = \sum_{i=1}^{n-w} X_{n-w} + \sum_{i=n-w+1}^{n} X_w,$$

where in the case of $w > 1$, $\sum_{i=n-w+1}^{n} X_w$, is calculated using numpy.

4.1.2 Mean

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

$$\bar{X}_n = g(S_{n-w}, X_w) = \bar{X}_{n-w} + w\left(\frac{\bar{X}_w - \bar{X}_{n-w}}{n}\right),$$

where $\bar{X}_n$ is the updated mean of the full dataset, $\bar{X}_{n-w}$ is the previous rolling mean and, if $w > 1$, $\bar{X}_w$, is the temporal mean over the incoming data computed with numpy. If $w = 1$, $\bar{X}_w = x_n$.

4.1.3 Variance

The variance (written as `"var"`) is calculated for the incoming data stream, over the requested temporal frequency, by updating two estimates iteratively. Let the two-pass summary $M_{2,n}$ be defined as:

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

For the case where $w = 1$, the one_pass definition is given by:

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

where $\bar{X}_n$ and $\bar{X}_{n-1}$ are given by the algorithm for the mean shown above. In the case where the incoming data has more than one time step ($w > 1$), $M_{2,n}$ is updated by:

$$M_{2,n} = g(S_{n-w}, X_w) = M_{2,n-w} + M_{2,w} + \frac{\sqrt{(\bar{X}_{n-w} - \bar{X}_w)}(w(n-w))}{n},$$

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.

At the end of the iterative process (when the last value is given to complete the statistic), the sample variance is computed by:

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

See S. Mastelini for details.

4.1.4 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:

$$\text{std}(X_n) = \sqrt{\text{var}(X_n)}.$$

4.1.5 Minimum

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

$$\min(X_n) = g(S_{n-w}, X_w),$$

$$\text{if } \min(X_w) < \min(S_{n-w}), \text{ then } \min(S_{n-w}) = \min(X_w),$$

where if $w > 1, \min(X_w)$ is calculated using the minimum function in numpy.

4.1.6 Maximum

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

$$\max(X_n) = g(S_{n-w}, X_w)$$

$$\text{if } \max(X_w) > \max(S_{n-w}), \text{ then } \max(S_{n-w}) = \max(X_w).$$

where if $w > 1, \max(X_w)$ is calculated using the maximum function in numpy.

4.1.7 Threshold Exceedance

The threshold exceedance statistic (written as `"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 calculated by:

$$\text{exc}(X_n) = g(S_{n-w}, X_w),$$

$$\text{if } (X_w > \text{thresh exceed}), \text{ then } \text{exc}(X_n) = \text{exc}(S_{n-w}) + s$$

where s is the number of samples in X_w that exceeded the threshold. The variable in the final xr.Dataset output now corresponds to the number of times the data exceeded the threshold.

4.1.8 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.

4.1.9 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.

4.1.10 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 receive 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 updated 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 Januaries of the years the model has covered.

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 Frequencies

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 xr.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 one, corresponding the summary statistic requested by "stat_freq". If, however, if you have requested "stat_freq": "hourly" but you set "output_freq": "daily", you will have a xr.Dataset with a time dimension of length 24, corresponding to 24 hourly statistical summaries in one file. Likewise, if you set "stat_freq":"daily" and "output_freq":"monthly", your final output will have a time dimension of 31 (if there are 31 days in that month), if you started from the first day of the month, or, if you started passing data half way through the month, it will correspond to however many days are left in that month.

The "output_freq" must be the same or greater than the "stat_freq". If you set "stat_freq" = "continuous" you must set "output_freq" to the frequency 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 that this is 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 xr.DataArray, you do not need to set this, however if you provide an xr.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 statistic will only be output in memory and will get overwritten when new data is passed to the Opa class. It is recommended to set this to True and a netCDF file will be written (in the "out_filepath") when the statistic is completed.

If you have requested to save the output, the file name will be timestamp_variable_stat_frequency_statistic.nc. For example, if you asked for a monthly mean of precipitation the file name would be 2070_05_pr_monthly_mean.nc. The one_pass will not differentiate between different experimental runs.

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. Setting to `True` will 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

A NOTE ON TIMESTAMPS

The initialisation of the Opa class is given by the line

```
opa_stat = Opa("config.yml") # initalise some statistic using the config.yml file
```

If checkpointing has been set to `True` in the request, then this line will look for a checkpoint file to start from. If checkpointing has been set to `False` or no checkpoint file exists, it will initialise the class from scratch.

When the next line is run

```
dm = opa_stat.compute(data) # pass some data to compute
```

The Opa class will check the time stamp of the new data and, if the class has been initalised from a checkpoint file, it will compare the time stamp of the new data to the last time stamp in the checkpoint file. It will return one of four options.

- option 1 : the new time stamp is exactly the old time stamp plus the time step. This will pass without error.
- option 2 : the new time stamp is greater than the old time stamp by $<= 2$ x the time step. This will return the phrase: `'Time gap at str(time_stamp) too large, there seems to be data missing, small enough to carry on'` and will continue with the calculation.
- option 3 : the new time stamp is greater than the old time stamp by $> 2x$ the time step. This will cause the one_pass to exit with Value Error : `'Time gap at str(time_stamp) too large, there seems to be some data missing'`.
- option 4 : the new stamp is further back in time than the old time stamp from the checkpoint file. This indicates that the model has crashed and has been re-started or the calculation is starting again. If this new time stamp then corresponds to the beginning of the requested statistic, the one_pass will start the calculation again and overwrite the checkpoint file. If this new time stamp does not correspond to the start of the requested statistic, but rather part way through, the Opa will simply skip the data and wait for the data that corresponds to the next time stamp in the sequence. **In this way, the one_pass does not require to be manually re-set**.

CHAPTER

SIX

A NOTE ON THE OUTPUT

The output `dm` from the one_pass, given by `dm = opa_stat.compute(data)` will only not be `None` when enough data has been passed to the Opa class to complete the statistic. For example, if you have hourly data and have requested a daily statistic, `dm` will return after the 24th time step has been passed. If passing a continuous data stream, `dm` will be overwritten when new data is given to the Opa class. Set `"save" : True` to save `dm` to disk.

The output, `dm` 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 data request* 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 attributes (metadata) included in the Dataset will be present in the final Dataset with a new 'history' attribute corresponding to details of the one_pass algorithm and the time of its creation. 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.

If you have requested to save the output the file name will be `timestamp_variable_stat_frequency_statistic.nc`. For example, if you asked for a monthly mean of precipitation the file name would be output as `2070_05_pr_monthly_mean.nc`. The one_pass will not differentiate between different experimental runs.

CHAPTER

SEVEN

TESTS

The one_pass 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 impelemented in the CI/CD.

7.1 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 statistic request.

CHAPTER

EIGHT

EXAMPLES AND TUTORIALS

8.1 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 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.

8.2 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

NINE

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.

9.1 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.

9.2 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.

9.3 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.

9.4 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

TEN

REFERENCES AND ACKNOWLEDGEMENTS

10.1 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        = {2023},
  publisher   = {GitLab},
  journal     = {Barcelona Supercomputing Center, Earth Sciences GitLab},
  howpublished = {\url{https://earth.bsc.es/gitlab/digital-twins/de_340/one_pass/main}},
}
```

10.2 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