Journal of open research software

A Python Package to Calculate the OLR-Based Index of the Madden-Julian-Oscillation (OMI) in Climate Science and Weather Forecasting

SOFTWARE METAPAPER

CHRISTOPH G. HOFFMANN
GEORGE N. KILADIS
MARIA GEHNE
CHRISTIAN VON SAVIGNY
*Author affiliations can be found in the back matter of this article

ubiquity press

ABSTRACT

The Madden-Julian Oscillation (MJO) is a prominent feature of the intraseasonal variability of the atmosphere. The MJO strongly modulates tropical precipitation and has implications around the globe for weather, climate and basic atmospheric research. The time-dependent state of the MJO is described by MJO indices, which are calculated through sometimes complicated statistical approaches from meteorological variables. One of these indices is the OLR-based MJO Index (OMI; OLR stands for outgoing longwave radiation). The Python package mjoindices, which is described in this paper, provides the first open source implementation of the OMI algorithm, to our knowledge. The package meets state-of-the-art criteria for sustainable research software, like automated tests and a persistent archiving to aid the reproducibility of scientific results. The agreement of the OMI values calculated with this package and the original OMI values is also summarized here. There are several reuse scenarios; the most probable one is MJO-related research based on atmospheric models, since the index values have to be recalculated for each model run.

CORRESPONDING AUTHOR:
Christoph G. Hoffmann
Institute of Physics, University of Greifswald, Felix-Hausdorff-Str. 6, 17489 Greifswald, Germany
christoph.hoffmann@uni-greifswald.de

KEYWORDS:
Madden-Julian Oscillation; OLR-based MJO Index; OMI; atmosphere; weather forecasting; climate; intraseasonal variation; Python

TO CITE THIS ARTICLE:
Hoffmann CG, Kiladis GN, Gehne M, von Savigny C 2021 A Python Package to Calculate the OLR-Based Index of the Madden-Julian-Oscillation (OMI) in Climate Science and Weather Forecasting. *Journal of Open Research Software*, 9: 9. DOI: *https://doi.org/10.5334/jors.331*

(1) OVERVIEW

INTRODUCTION

The Madden-Julian Oscillation (MJO) is a prominent feature of the Earth's atmosphere-ocean system with a high relevance for weather, climate and basic atmospheric research. In order to explain the purpose and structure of the software package introduced here, we first mention a few meteorological key points of the MJO.

The Madden-Julian Oscillation

The MJO was described for the first time by [9] and is characterized by a strong tropical convective rainfall anomaly, which is perceptible as enhanced cloudiness and precipitation. This disturbance appears periodically in the Tropics over the Indian Ocean, then travels eastward and decays over the Pacific. It is of major relevance for variability in rainfall and wind in large parts of the Tropics and is therefore one of the most important recurring patterns of variability in the Earth's atmosphere-ocean system. Although the MJO is primarily observable in these tropical regions, its state has implications for many other locations in the atmosphere, which is still a subject of current research (e.g., temperature and dynamical features in the polar regions and in higher atmospheric layers [3, 2, 14] or modifications of extreme precipitation events over northeast Africa, the Middle East, and eastern China [5]. Reviews of the MJO are given in [15] and [16]. One important aspect of the MJO is that it acts on the intra-seasonal timescale (periods of 30–90 days) in contrast to for example the well-known El Niño-Southern Oscillation (ENSO), which acts on time scales of years. A better understanding of the MJO is therefore not only of interest for general atmospheric and climate research but is also thought to be helpful for improving the forecast skill of weather forecasts [16].

The passage of the convective anomaly from the Indian Ocean to the Pacific is conventionally split into 8 temporal phases, roughly defined by the position of the anomaly as shown in ***Figure 1***. For each of these phases, the weather at an individual tropical location tends to have particular characteristics. However, such an ideally clear definition of the phases and the corresponding weather patterns for particular regions is difficult for the real Earth, since the MJO pattern is superimposed on all other kinds of natural variability on various time scales from days to years (e.g., common weather fluctuations, seasonal variations, the state of ENSO). Hence, one recent aim of MJO research was to establish statistical approaches, with which the state of the MJO can be extracted from observed meteorological data as objectively as possible in the form of time dependent index values. Several indices have been formulated in the past with the most important ones being the Real-time Multivariate MJO index (RMM) [12] and the OLR-based MJO index (OMI) [6].

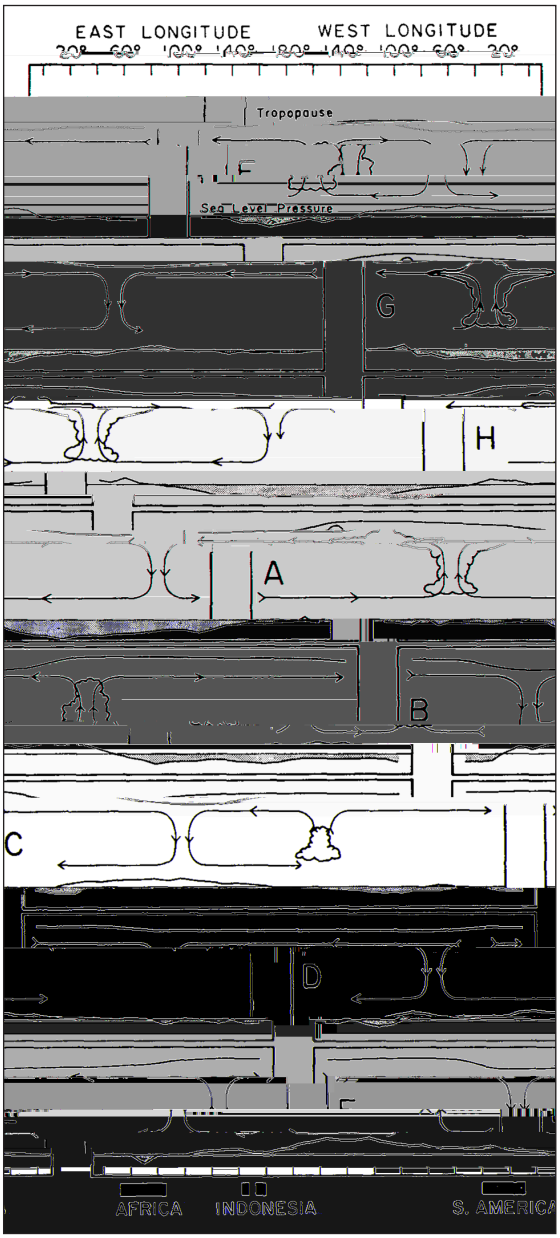

Figure 1 Depiction of the definition of the 8 MJO phases according to the position of the convection anomaly. The figure is taken from the original publication by Madden and Julian [9], where more details can also be found.

MJO index OMI

The software package introduced here provides a modern and sustainable reimplementation of the OMI algorithm and we first outline the general approach of the index calculation, before the software package itself is introduced. For details on the underlying algorithm see Kiladis et al. [6]. As with most of the MJO index algorithms, the OMI calculation is based on a principal component analysis (PCA, also known as empirical orthogonal function analysis). The PCA is basically a linear algebraic basis transformation. The new basis vectors, called empirical orthogonal functions (EOFs), are defined such that the first few of them cover most of the variability in the dataset, which allows for neglecting the other basis vectors and therefore reducing the amount

of data treated in the analysis. In the case of OMI, the data to which the PCA is applied are temporally resolved and filtered maps of OLR. The time dependency of the data is then contained in time dependent coefficients with respect to the EOFs, which are called the principal components (PCs). Wilks [13] or other textbooks offer a detailed description of the PCA in the context of the atmospheric sciences. For OMI and other MJO indices it turns out that only the first two EOFs are needed to describe the MJO sufficiently [12, 6]. Hence, only two PCs are needed to approximately cover the bulk of the temporal and spatial evolution of the MJO. The values of the time dependent PCs together with the associated EOFs form the MJO index. Once the index has been computed, the phase and the strength of the MJO can be calculated from the PCs following simple rules [12]. Note that the calculation is somewhat more complicated for OMI. To better represent the seasonal variations, the PCA is not computed for the whole dataset at once, but in a climatological sense for each day of the year (DOY) separately. Hence, the PCA is executed 366 times (including leap days) resulting in 366 pairs of EOFs. The PCs for each particular date are then calculated using the EOFs of the corresponding DOY [6].

While the RMM index is a good choice to conduct real-time analyses of the MJO, OMI overcomes a few drawbacks of RMM at the expense of the real-time capability [10, 6], although a real-time version of OMI, called ROMI, is also available [6]. Hence, OMI is a good choice for general MJO research that does not depend on real time information of the MJO. This applies for many research goals, e.g., the improvement of the understanding of the MJO based on retrospective analysis of meteorological data or based on data of atmospheric models.

Ideas behind the reimplementation of OMI

In contrast to these widespread possible applications, to our knowledge there is no publicly available software to compute OMI from data. Only available is the official description paper [6], which summarizes the general properties of the algorithm, and a website (*https://www.esrl.noaa.gov/psd/mjo/*, last access on 06/10/2020), which provides the corresponding OMI values for the real-world MJO evolution. The OMI index time series on this website is extended sporadically, which makes this dataset suitable for most MJO research based on retrospective data (note that the values of the real-time version ROMI are updated daily). However, this does not help for model analyses, because the OMI index has to be recalculated based on the modeled OLR instead of the real-world OLR to get a consistent representation of the MJO in modeled data.

Given the importance that OMI has gained in MJO research there are collectively a number of reasons to provide a quality-tested open source code to compute OMI:

- Facilitate MJO research using the OMI index based on modeled data.
- Facilitate MJO research based on real-world data without being dependent on updates of the respective web page (although the dependency on the availability of the observational OLR data remains).
- Enable researchers to easily further understand the characteristics of OMI by modifying an established and tested version of the original OMI calculation (which depends on some choices of thresholds etc.). A recent example of such a case has been brought up by Hoffmann and von Savigny [4], whereas a previous example for the need to assess MJO index properties is given by Wang et al. [11].
- Make OMI also conveniently available to all researchers, whose research questions might involve the MJO and who can be spared from the effort to characterize the MJO themselves.
- Publish the source code as a special kind of technical documentation of the rather complicated calculation approach with 100%-coverage of all its details in addition to the original publication [6].
- Check and demonstrate the reproducibility of the involved and potentially error-prone statistical approach as a contribution to good scientific practice. (The reimplementation actually led to an update [7] of the official description paper [6].)

The reimplementation in Python, which is presented here, was motivated by two of the points listed above, particularly the analysis of modeled data and the further investigation of OMI characteristics. While the implementation approach was at first solely based on the description paper [6], it quickly turned out to be crucial to discuss the implementation details directly with the designers of OMI to speed up the development. Hence, a cooperation with the original authors of [6] was established, who provided not only these details, but also parts of the original code and intermediate calculation results against which the reimplementation could be tested. Overall, we realized that it requires a considerable effort to understand and reimplement the OMI calculation, so that it appears to be worthwhile to share the code with the community. Hence, we compiled the implementation in the Python package `mjoindices` and added quality control mechanisms, examples and documentation. The code includes the complete processing chain: the preprocessing of the OLR data, the calculation of the EOFs, the calculation of the PCs, the post processing of the results, as well as I/O and plotting utilities. The name of the package `mjoindices` indicates that it is intended to include also the calculation of other MJO indices. However, this is a long-term goal and since the OMI calculation is of interest on its own, we decided to publish the OMI calculation now.

Since the package has been released only recently, there is so far no big community using it. However, there

are numerous research papers using OMI and we have already gotten numerous requests regarding the usage of the package prior to the release. In one case we have already shared a preliminary version of the code.

IMPLEMENTATION AND ARCHITECTURE

The package structure is designed to be easily extendable to other MJO indices in the future. Basic modules are therefore placed directly in the `mjoindices` packages, whereas modules specific for OMI are placed in the `mjoindices.omi` sub-package.

Data handling

The basis for the data handling and all numerical operations is the `numpy` package, especially the class `numpy.ndarray`. Dates are treated consistently as `numpy.datetime64` objects.

There are four classes that handle the data exchange between the `mjoindices` package and the calling code developed by the users:

- The class `mjoindices.olr_handling.OLRData` represents the input OLR data. The only code that the users have to develop themselves in order to run the package is actually a method that creates an `OLRData` object filled with the three-dimensional OLR dataset and the corresponding numerical grids for latitude, longitude and time. This class is also internally used to represent intermediate results of the OLR preprocessing.

The other three classes represent the calculation results:

- A calculated pair of EOFs and associated statistical diagnostic quantities are stored in `mjoindices.empirical_orthogonal_functions.EOFData`.
- The list of all 366 pairs of EOFs is stored in `mjoindices.empirical_orthogonal_functions.EOFDataForAllDOYs`.
- The PC time series, which is the basic output that represents the temporal evolution of the MJO, is represented by `mjoindices.principal_components.PCData`

All these classes come with routines for I/O and basic diagnostic plots.

Note that it is in principle possible to provide the OLR data on freely chosen spatial and temporal grids as input for the OMI calculation. However, the original OMI calculation has been performed on spatial grids with a spacing of 2.5° between 20° S and 20° N in latitude and 0° to 360° in longitude as well as daily averages in the time domain [6]. Although it might be expected that the algorithm returns consistent values over a wider range of spatial resolutions (particularly for higher resolutions), this has not been rigorously tested. Hence, it is highly recommended that the users either provide the OLR data on the original grid or at least carefully confirm that the characteristics of the index calculated on a different grid are similar to those of the original OMI values.

Implementation of the OMI algorithm

The calculation of the OMI EOFs and PCs itself is implemented in the module `mjoindices.omi.omi_calculator`. The EOF calculation is more complex than the calculation of the PCs and is separated into a preprocessing of the OLR data, the execution of the PCA and a post processing of the resulting EOFs. These steps are callable separately to evaluate intermediate results, however in most cases all steps will be executed together, which is done by the method `calc_eofs_from_olr()`.

The preprocessing consists of a temporal and spatial filtering of the input data. This is actually a rather involved centerpiece of the OMI calculation and has been implemented in the separate module `mjoindices.omi.wheeler_kiladis_mjo_filter`, which should, however, not be relevant for the end-users.

For the PCA step, two different implementations can be chosen via an argument of the method call: the internal implementation, which follows the description by Kutzbach [8] or the implementation in an external Python package, namely the `eofs` package described by Dawson [1]. There is no noteworthy difference in the results between both variants, as the usage of the external package has originally been included to validate the internal implementation. The internal implementation now remains in the code to have a self-contained and fully understandable package for the OMI calculation, so it is more for purposes of documentation. Using the external general PCA package instead promises perhaps to be a higher performance implementation in terms of computation time. Overall, the differences are marginal and the users will probably simply use the external package without further considerations.

The post processing consists of two pragmatic steps, introduced in the original calculation of OMI. First, the signs of the EOFs calculated by the PCA are arbitrary. This means that the signs may switch from one DOY to another, which is undesirable. Therefore, the signs of all 366 pairs of EOFs are aligned after their computation as the first step, i.e. arbitrary sign reversals of EOFs between neighboring DOYs will be removed. Note that this post processing step might cause problems if the calculation is not performed on the original spatial grids. In this case, the users should call the preprocessing and the PCA calculation separately and then implement individual post processing solutions themselves if needed at all. Second, it was found that reasonable EOFs for some DOYs at the beginning of November were difficult to obtain [6], so that they were replaced by an interpolation between the EOFs for the DOYs 293 and 316 (Note that the range was stated differently in Kiladis et al. [6] and has been corrected as a result of the reimplementation described here). This interpolation

was also included in this package to achieve compatability with the original index. However, it is probably specific to the calculation based on the observational OLR dataset and might either be unnecessary or only be relevant for a different period in the case of other datasets. Hence, this post processing step is configurable and each user has to consider the application depending on the specific case.

Relation to the original OMI implementation

The reimplementation presented here should not be understood as a one-to-one porting of the original code. Instead, it is essentially a new implementation following the statistical steps described in Kiladis et al. [6]. This means that the computation results will slightly differ due to both, numerical artifacts and differences in the implementation details (note that one advantage of a completely independent implementation is that it challenges the reproducibility of the scientific description during the implementation process). The still very high degree of the agreement between the new implementation and the results of Kiladis et al. [6] will be outlined below together with the quality control description. In addition, the complete code for the recalculation of the original OMI values and the validation is included as an example in the package so that it can be executed by the users to evaluate the differences in detail themselves.

One subtle detail, probably responsible for a part of any differences in the results obtained, is the treatment of leap years in the selection of the data samples. We have included two options in the code, which are selectable for the users with the keyword argument `strict_leap_year_treatment` of the respective functions. The first option (`strict_leap_year_treatment=False`), which is the default, leads to results which are closer to the original values, however, the treatment of leap years has a practical aspect (explaining the details would be beyond the scope of this overview, but a respective comment is included in the documentation of the code). The second option (`strict_leap_year_treatment=True`) uses the modern `numpy` date-time routines to treat the leap years more explicitly at the expense of the agreement with the original values. Although we have decided to include both options and leave the choice to the users, we note the second option has to be used with care; the EOFs for DOY 366 may strongly differ from those of DOY 365 and DOY 1, which may produce unwanted jumps in the PC time series. In any case, we emphasize that neither of the implementations is necessarily identical to the original code, but the results of both are very close to the original based on the results of our evaluation (see below).

Conceptual separation of the core package and additional material

For convenience, the package is listed in the Python package index (*https://pypi.org/project/mjoindices/*), so that it can be installed with common Python package managers like `pip`. However, this Python package in the narrower sense only contains the operational code itself, but neither the test suite nor the examples. Those are available together with the complete source code and the documentation in the broader sense of the Python package on GitHub or Zenodo (see below) and can be run with the installed package.

QUALITY CONTROL

Manual evaluation of calculated OMI values using example code

The package includes a basic example, which is available as a common Python script (`recalculate_original_omi.py`) or as a Jupyter Notebook file (`recalculate_original_omi.ipynb`). It recalculates the original OMI values, i.e., executes the reimplemented algorithm with the original input files. Hence, this example not only shows how the package is used, but also provides a direct comparison of the original and the recalculated OMI values. This basic comparison is extended by the script `evaluate_omi_reproduction.py` (or `evaluate_omi_reproduction.ipynb`, respectively), which provides a more detailed comparison. This script does not run the OMI algorithm again, but only analyzes the previously recalculated and saved values in more detail. Overall, the users can use both scripts to become familiar with the usage of the package, but they can also judge the degree of agreement with the original values. As stated before, a perfect agreement is not expected, since the implementation is not a one-to-one port of the original code. Instead, the expected tolerances are described below. This description also allows for a comparison with the individual calculations performed by the users to judge the performance of their local installations. Note that the input data and also the original OMI dataset have to be downloaded from the official websites or Zenodo (see below) before running the example. Respective information and links are given in the upper part of the example code.

Automated testing

The software quality control includes three levels of automated testing routines, which are based on the `pytest` framework. First, many functions are routinely tested using unit tests. These unit tests cover those classes, which provide simple and self-contained functionality apart from the main numerical calculation, i.e. the classes used for the data handling and respective I/O routines. These tests are mainly based on hard-coded simple sample data, for which the expected operation results should be obvious. Second, an integration test validates the results of the complete calculation chain against the original OMI data. Since slight deviations are expected, the test operates with specific tolerances, which are specified below. Third, another integration test validates the complete calculation chain against OMI values, which were previously computed using the `mjoindices.omi` package itself. Hence, equivalence of the results is expected here, so that

this test will guarantee the stability of the results in case of installations on different systems or code changes. Note that both integration tests are implemented in one routine to save computing time during the test execution. Note further that some data files, which are not included in the package, have to be downloaded before the integration tests can be executed. These files are permanently available at Zenodo (*https://doi.org/10.5281/zenodo.3746563*). Details are given in a specific Readme document, which is included in the test suite (basically, this concerns similar external files, as are needed for the examples described above).

Quantitative comparison of recalculated and original OMI values

As stated before, the code in the `mjoindices` package is a new implementation and not a one-to-one port of the original code, so that slight deviations between the results of both implementations have to be expected. As a reference, some basic comparison results are discussed here. This information can also be used to compare it to the local output of the example code, which produces similar figures, among others. In addition, the following figures also describe the tolerances accepted by the respective integration test mentioned before. Note that the comparison results depend slightly on the calculation setup, particularly the treatment of leap years (see above). All figures are based on calculations with the setting `strict_leap_year_treatment=False`, which is closer to the original and is the default setting. In contrast, the ***Tables 1*** and ***2*** show results for both settings. In this case the comparison statistics are shown for both including and excluding DOY 366, which is the DOY that shows the

EOF	INDICATOR	LEAP YEAR TREATMENT	DOY 366	VALUE
1	Correlation	not strict	both	>0.994
2	Correlation	not strict	both	>0.993
1	99% percentile	not strict	both	<0.0084 W/m^2
2	99% percentile	not strict	both	<0.0065 W/m^2
1	Correlation	strict	excluded	>0.994
2	Correlation	strict	excluded	>0.993
1	99% percentile	strict	excluded	<0.0084 W/m^2
2	99% percentile	strict	excluded	<0.0065 W/m^2

Table 1 Comparison of recalculated and original EOFs summarized over all DOYs. Note that we did not include numbers for the setup "strict leap year treatment/DOY 366 included", since these numbers are only determined by the EOFs of DOY 366, which is intentionally different from the original. Hence, no conclusion on the overall agreement can be drawn from these numbers.

PC	INDICATOR	LEAP YEAR TREATMENT	DOY 366	VALUE
1	Correlation	not strict	both	>0.998
2	Correlation	not strict	both	>0.998
1	Std.-Dev. of difference	not strict	both	<0.0458
2	Std.-Dev. of difference	not strict	both	<0.0488
1	99% percentile	not strict	both	<0.157
2	99% percentile	not strict	both	<0.1704
1	Correlation	strict	excluded	>0.998
2	Correlation	strict	excluded	>0.998
1	Std.-Dev. of difference	strict	excluded	<0.0449
2	Std.-Dev. of difference	strict	excluded	<0.0484
1	99% percentile	strict	excluded	<0.1523
2	99% percentile	strict	excluded	<0.1671
1	Correlation	strict	included	>0.998
2	Correlation	strict	included	>0.998
1	Std.-Dev. of difference	strict	included	<0.0509
2	Std.-Dev. of difference	strict	included	<0.0501
1	99% percentile	strict	included	<0.1552
2	99% percentile	strict	included	<0.1708

Table 2 Comparison of recalculated and original PCs considering the complete period of the available original data (01/01/1979 to 28/08/2018).

most pronounced deviation compared to the original data when using `strict_leap_year_treatment=True`.

Figure 2 shows the EOF functions 1 and 2 for two selected DOYs. Shown is one example, which is representative for the best agreement (DOY 23), and one, which is representative for the worst agreement (DOY 218). For the worst case, the visual resemblance of the original and the recalculated EOFs is also obvious and it is seen that the magnitude of the deviation is small for both examples.

Figure 3 provides a quantitative overview of the agreement of the EOFs 1 and 2 for all DOYs based on different statistical values: The correlation of the EOFs (upper panel), the mean and the standard deviation of the differences between the EOFs (the two panels in the middle), and different percentiles of the absolute differences between the EOFs (bottom panel). In particular the correlation, which varies between 0.994 and 0.999, shows that the agreement is nearly optimal for all DOYs, including the previously called *worst* cases. The other numbers have to be seen in the context of the total magnitude of the EOFs (which can be estimated from ***Figure 2***) and also indicate satisfying agreement. Note that the meaning of, e.g., the 99% percentile is that 99% of the absolute differences between the original and the recalculated EOFs are lower than the stated number.

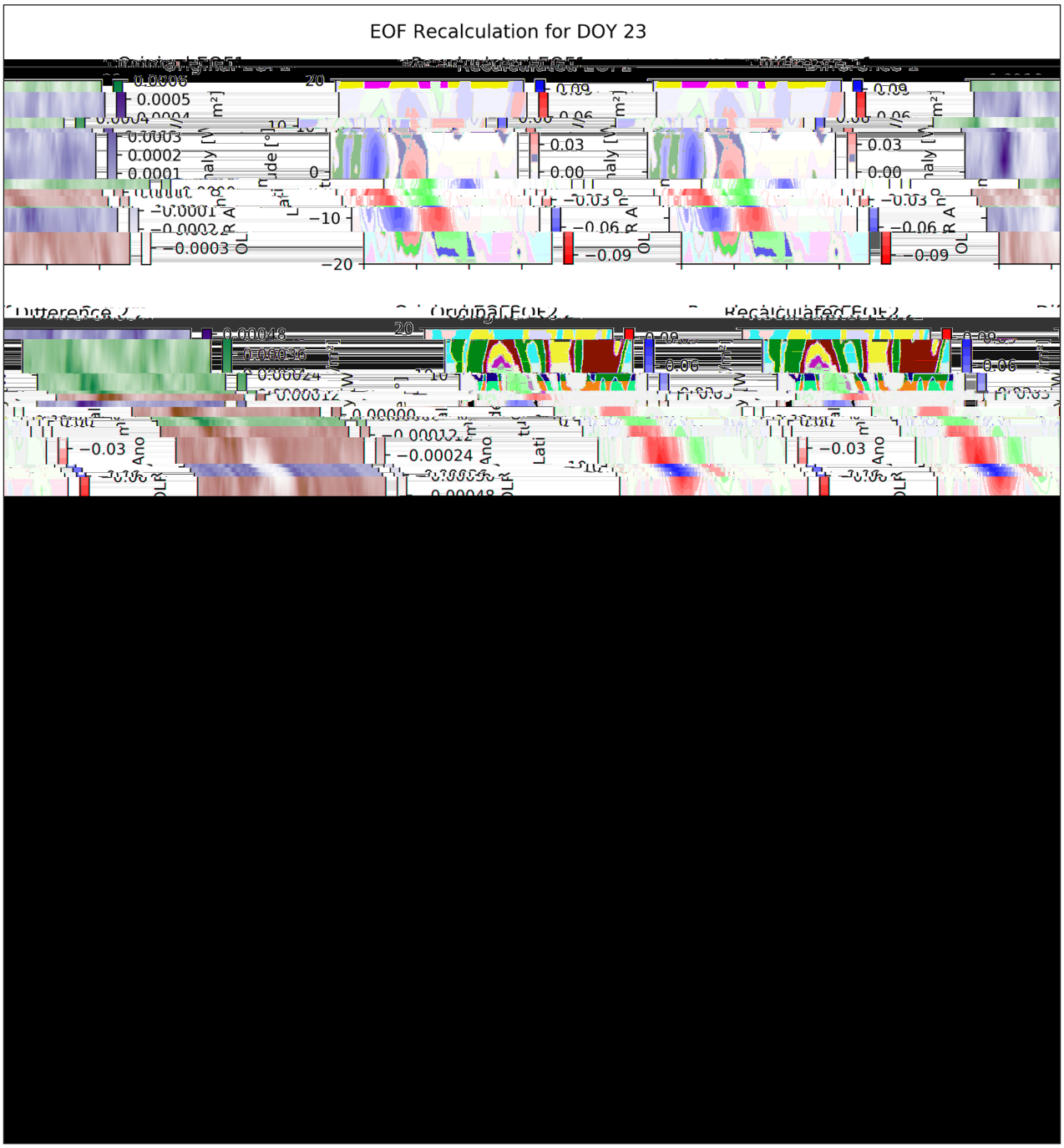

Figure 2 Examples of recalculated EOFs in comparison to the original EOFs for DOY 23, which is among the DOYs with the best agreement, and DOY 218, which has the worst agreement. Note that the color scale of the panels with the differences varies.

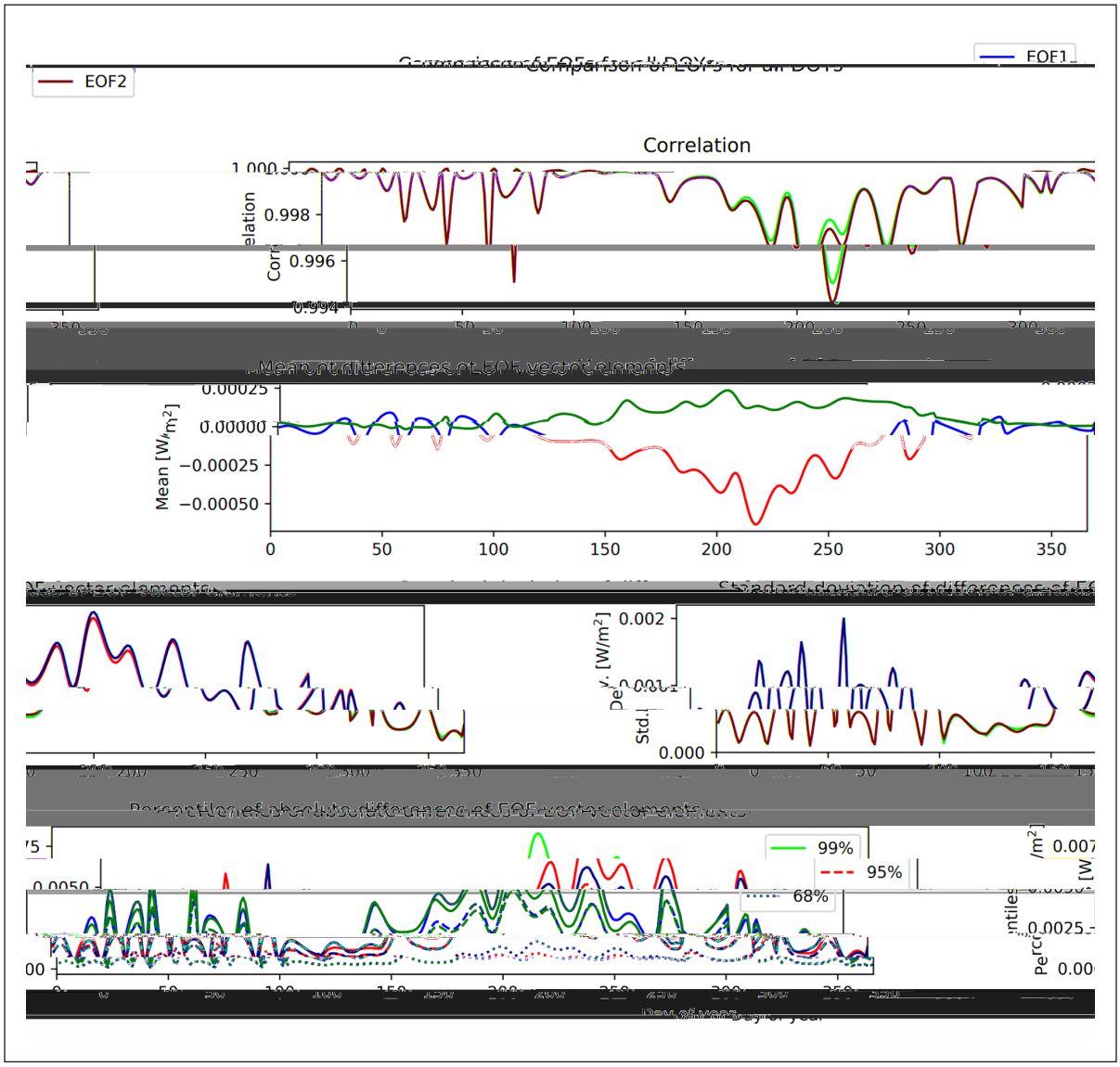

Figure 3 Detailed comparison statistics for the EOFs of all DOYs. See text for details.

Table 1 summarizes the minimum agreement, which can be expected for the individual EOFs, considering all DOYs. In other words, these numbers indicate the maximum tolerance that must conservatively be allowed if the EOFs for each DOY are not assessed individually. This is also done in the respective integration test, so that these numbers state the test tolerances.

Figure 4 shows the recalculated and the original time series of the PCs 1 and 2 for the year 2011, which has been arbitrarily chosen. The PC time series contain the principal information on the temporal evolution of the MJO and are therefore of major interest for the users. Visually, almost no difference between the original and the recalculated values can be seen. This is confirmed by ***Table 2***, which summarizes the agreement between the PC time series for the complete period (01/01/1979 to 28/08/2018), for which the original data was available at the time of the implementation.

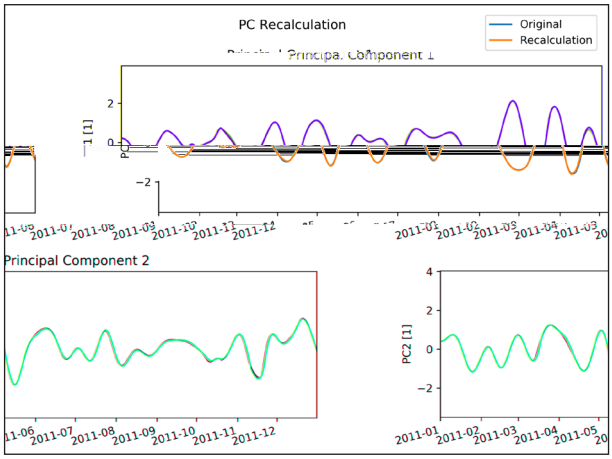

Figure 4 Comparison of the recalculated and original PCs for an arbitrarily chosen sample period (the year 2011).

(2) AVAILABILITY

OPERATING SYSTEM

Tested on Ubuntu 18.04 Linux and Windows 10.

PROGRAMMING LANGUAGE

Python > = 3.6 (tested with Python 3.6, 3.7, and 3.8)

ADDITIONAL SYSTEM REQUIREMENTS

There are no special hardware requirements in addition to a state-of-the-art personal computer system (e.g., 1.5 GHz Processor, 8 GB memory, a few GB free disk space). The complete algorithm will run for a few hours on such a system.

DEPENDENCIES

The package depends on the following standard Python packages. These can be installed using common Python package managers (e.g., `pip`) and will usually also be automatically installed along with `mjoindices`.

- `numpy`, version >= 1.17.0
- `matplotlib`, version >= 3.1.1
- `pandas`, version >= 0.25
- `scipy`, version >= 1.3.0

In order to run the unit and integration tests, the following package is needed:

- `pytest`, version >= 5.0.1

To have the possibility to use an external implementation of the PCA as described before, the following package can be installed:

- `eofs`, version >= 1.4.0

Some of the unit and integration tests depend on external datasets, which serve either as input or as reference for the results. These datasets are also permanently available from Zenodo (*https://doi.org/10.5281/zenodo.3746563*). They have to be downloaded and saved into a particular directory before these tests can be successfully executed. Details are given in a Readme document, which is part of the test suite. Also the examples need these or similar files. In this case, the description of the necessary files is found in the source code documentation of the examples.

LIST OF CONTRIBUTORS

- Christoph G. Hoffmann (University of Greifswald, Germany) has written the code and led the project.
- George N. Kiladis (NOAA/Physical Sciences Laboratory, Boulder, Colorado) has contributed code samples as a reference (which are not included in the package) and discussed several implementation issues.
- Maria Gehne (CIRES, University of Colorado Boulder, and NOAA/Physical Sciences Laboratory, Boulder, Colorado) has tested the package from the perspective of the original designers of OMI.
- Juliana Dias (CIRES, University of Colorado Boulder, and NOAA/Physical Sciences Laboratory, Boulder, Colorado) has provided a file with reference data.
- Christian von Savigny (University of Greifswald, Germany) has contributed to the discussion of the general approach.

SOFTWARE LOCATION

Archive

Name: Zenodo

Persistent identifier: *https://doi.org/10.5281/zenodo.3957857*

Licence: GNU General Public License v3.0

Publisher: Christoph G. Hoffmann

Version published: 1.2.0

Date published: 23/07/2020

Code repository

Name: GitHub

Persistent identifier: *https://github.com/cghoffmann/mjoindices*

Licence: GNU General Public License v3.0

Date published: 23/07/2020

LANGUAGE

The language of the code and the documentation is English.

(3) REUSE POTENTIAL

The reimplementation of the OMI algorithm as open source code can be helpful for climate, weather, and basic atmospheric research in diverse aspects as has been outlined in the introduction. This also includes documentation aspects and good scientific practice.

The primary reuse case of the package in terms of actually running the code to calculate OMI values consists of the analysis of the MJO behavior in complex models of the atmosphere. The major point is that it will be necessary to recalculate OMI for the atmospheric conditions simulated by a specific model to get a consistent representation of the MJO in that particular model. Due to the chaotic nature of the Earth's atmosphere even an ideal numerical model would not able to precisely reproduce the Earth's weather at a particular point in space and time for many days after its initialization. Hence, although state-of-the-art atmospheric models produce realistic weather patterns and realistic climatological conditions (in the sense of large-scale and long-term averaging), the conditions for individual periods and locations cannot be reasonably compared to the real world for long-term runs. This implies that it is impossible to use the original OMI index calculated for the real world based on OLR observations to also describe the MJO in a modeled atmosphere. Put in other words, each MJO-related study based on free running atmospheric models has to recompute the OMI index for each model run based on the modeled OLR data to get a consistent representation. This is easily possible with the presented Python package. Given the various atmospheric models (of which all are run with many

different setups depending on the particular science questions) and the rising awareness of the relevance of the MJO for tropical and extra-tropical meteorology, we expect a high reuse potential, as long as the community becomes aware of the existence of this code.

A more specific reuse case is to understand the characteristics of OMI itself. This knowledge can become useful, when subtle interactions between the MJO and other processes in the earth system are studied. In this case, it must be considered that the particular representation of the MJO (here OMI) influences the results. For this, it can be helpful to be able to recompute OMI with slight modifications, e.g., with different values for the filter constants of the bandpass filter. We do not expect, however, that these individual exploratory variations of the implementation should feed back into the basic source code, as the basic code should unambiguously represent the original documented and scientifically approved OMI algorithm.

CONTRIBUTIONS

We expect that the results, which are produced by the package, will be stable right from the outset, since all major features for the complete reproduction of OMI have already been implemented and tested. Nevertheless, we welcome contributions to the code, such as code optimizations or implementations of other MJO indices. These contributions will also have to meet the high quality standards in terms of automated testing etc. to keep the results stable and scientifically reliable. For contributions and questions, we can be contacted using the project's GitHub page (e.g., "Pull requests" and "Issues") and the author contacts of this manuscript.

ACKNOWLEDGEMENTS

We thank Alejandro Jaramillo Moreno for fruitful discussions on the implementation of the Wheeler-Kiladis-Filter in Python and Rattana Chhin for beta-testing the package. We would like to thank the two reviewers for their valuable comments on the manuscript and the software. We acknowledge support for the Article Processing Charge from the DFG (German Research Foundation, 393148499) and the Open Access Publication Fund of the University of Greifswald.

FUNDING STATEMENT

This work was supported by the University of Greifswald.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Christoph G. Hoffmann *orcid.org/0000-0003-2712-8648*
Institute of Physics, University of Greifswald, Felix-Hausdorff-Str. 6, 17489 Greifswald, Germany

George N. Kiladis *orcid.org/0000-0001-6588-3762*
NOAA/Physical Sciences Laboratory, Boulder, Colorado, USA

Maria Gehne *orcid.org/0000-0002-7267-8676*
CIRES, University of Colorado Boulder, and NOAA/Physical Sciences Laboratory, Boulder, Colorado, USA

Christian von Savigny *orcid.org/0000-0001-7926-3986*
Institute of Physics, University of Greifswald, Felix-Hausdorff-Str. 6, 17489 Greifswald, Germany

REFERENCES

1. **Dawson A.** Eofs: A Library for EOF Analysis of Meteorological, Oceanographic, and Climate Data. *J. Open Res. Softw*. 2016; 4(1): e14. DOI: *https://doi.org/10.5334/jors.122*
2. **Garfinkel CI, Benedict JJ, Maloney ED.** Impact of the MJO on the boreal winter extratropical circulation. *Geophys. Res. Lett*. 2014; 41(16): 6055–6062. DOI: *https://doi.org/10.1002/2014GL061094*
3. **Garfinkel CI, Feldstein SB, Waugh DW, Yoo C, Lee S.** Observed connection between stratospheric sudden warmings and the Madden-Julian Oscillation. *Geophys. Res. Lett*. 2012; 39(18). DOI: *https://doi.org/10.1029/2012GL053144*
4. **Hoffmann CG, von Savigny C.** Indications for a potential synchronization between the phase evolution of the Madden-Julian oscillation and the solar 27-day cycle. *Atmos. Chem. Phys*. 2019; 19(7): 4235–4256. DOI: *https://doi.org/10.5194/acp-19-4235-2019*
5. **Jones C, Waliser DE, Lau KM, Stern W.** Global Occurrences of Extreme Precipitation and the Madden-Julian Oscillation: Observations and Predictability. *J. Climate*. 2004; 17(23): 4575–4589. DOI: *https://doi.org/10.1175/3238.1*
6. **Kiladis GN, Dias J, Straub KH, Wheeler MC, Tulich SN, Kikuchi K, Weickmann KM, Ventrice MJ.** A Comparison of OLR and Circulation-Based Indices for Tracking the MJO. *Mon. Wea. Rev*. 2014; 142(5): 1697–1715. DOI: *https://doi.org/10.1175/MWR-D-13-00301.1*
7. **Kiladis GN, Dias J, Straub KH, Wheeler MC, Tulich SN, Kikuchi K, Weickmann KM, Ventrice MJ.** CORRIGENDUM. *Mon. Wea. Rev*. 2020; 148(2): 875–876. DOI: *https://doi.org/10.1175/MWR-D-19-0385.1*
8. **Kutzbach JE.** Empirical Eigenvectors of Sea-Level Pressure, Surface Temperature and Precipitation Complexes over North America. *J. Appl. Meteor*. 1967; 6(5): 791–802. DOI: *https://doi.org/10.1175/1520-0450(1967)006<0791:EEOSLP>2.0.CO;2*
9. **Madden RA, Julian PR.** Description of Global-Scale Circulation Cells in the Tropics with a 40–50 Day Period. *J. Atmos. Sci*. 1972; 29(6): 1109–1123. DOI: *https://doi.org/10.1175/1520-0469(1972)029<1109:DOGSCC>2.0.CO;2*

10. **Straub KH.** MJO Initiation in the Real-Time Multivariate MJO Index. *J. Climate*. 2013; 26(4): 1130–1151. DOI: *https://doi.org/10.1175/JCLI-D-12-00074.1*
11. **Wang S, Ma D, Sobel AH, Tippett MK.** Propagation Characteristics of BSISO Indices. *Geophys. Res. Lett*. 2018; 45(18): 9934–9943. DOI: *https://doi.org/10.1029/2018GL078321*
12. **Wheeler MC, Hendon HH.** An All-Season Real-Time Multivariate MJO Index: Development of an Index for Monitoring and Prediction. *Mon. Wea. Rev*. 2004; 132(8): 1917–1932. DOI: *https://doi.org/10.1175/1520-0493(2004)132<1917:AARMMI>2.0.CO;2*
13. **Wilks D.** Chapter 12 – Principal Component (EOF) Analysis. In: Wilks DS (ed.), *Statistical Methods in the Atmospheric Sciences, Vol. 100 of International Geophysics*. 2011; 100: 519–562. Academic Press. DOI: *https://doi.org/10.1016/B978-0-12-385022-5.00012-9*
14. **Yang C, Li T, Smith AK, Dou X.** The Response of the Southern Hemisphere Middle Atmosphere to the Madden-Julian Oscillation during Austral Winter Using the Specified-Dynamics Whole Atmosphere Community Climate Model. *J. Climate*. 2017; 30(20): 8317–8333. DOI: *https://doi.org/10.1175/JCLI-D-17-0063.1*
15. **Zhang C.** Madden-Julian Oscillation. *Rev. Geophys*. 2005; 43(2): RG2003. DOI: *https://doi.org/10.1029/2004RG000158*
16. **Zhang C.** Madden-Julian Oscillation: Bridging Weather and Climate. *Bull. Amer. Meteor. Soc*. 2013; 94(12): 1849–1870. DOI: *https://doi.org/10.1175/BAMS-D-12-00026.1*

TO CITE THIS ARTICLE:
Hoffmann CG, Kiladis GN, Gehne M, von Savigny C 2021 A Python Package to Calculate the OLR-Based Index of the Madden-Julian-Oscillation (OMI) in Climate Science and Weather Forecasting. *Journal of Open Research Software*, 9: 9. DOI: *https://doi.org/10.5334/jors.331*

Submitted: 20 April 2020 Accepted: 13 April 2021 Published: 14 May 2021