Software paper for submission to the Journal of Open Research Software

To complete this template, please replace the blue text with your own. The paper has three main sections: (1) Overview; (2) Availability; (3) Reuse potential.

Please submit the completed paper to: editor.jors@ubiquitypress.com

---

(1) Overview

Title
Pymrio - a Python based Multi-Regional Input-Output Analysis toolbox

Paper Authors
1. Stadler, Konstantin

Paper Author Roles and Affiliations
Developer of Pymrio, Industrial Ecology Programme, Norwegian University of Science and Technology (NTNU)

Abstract
Pymrio is an open source tool for Environmentally Extended Multi-Regional Input-Output (EE MRIO) analysis developed in Python. It provides a high-level abstraction layer for global EE MRIO databases in order to simplify common EE MRIO data tasks. Among others, Pymrio includes automatic download functions and parsers for several openly available EE MRIO databases (EXIOBASE 1 and 2, WIOD, Eora26) as well as methods for production and consumption based accounts calculation, aggregation, stressor origin estimation and visualization. The use of a consistent storage format including meta data and modification history for MRIOs allows to exchange data with other analysis tools, aiming for an increased interoperability of Industrial Ecology analysis software.

Keywords
Sustainability Analysis; Multi Regional Input Output Analysisi; Footprinting, Consumption Based Accounting

Introduction
Environmentally Extended Multi-Regional Input-Output (EE MRIO) tables describe economic relationships within and between regions and their environmental repercussions.

The analysis of the these tables have become the prevailing methodology in Industrial Ecology and sustainability science to evaluate globally spanning supply chains in order to calculate the environmental and social consequences of trade and consumption [1, 2, 3, 4, 5, 6].

In contrast to other Industrial Ecology methodologies like Life Cycle Assessment (e.g. Brightway [7] or openLCA [8]) or Material Flow Analysis (e.g. STAN [9]), few

generally available analysis packages for (Multi-Regional) Input-Output tables are available [10].

One of the very few well documented and stand alone packages for IO analysis, PyIO [11], has not been updated since 2011.

The MRIOLab suite [12, 13] takes a different approach, providing a virtual lab for the compilation of MRIO tables and thereby streamlining the compilation of MRIO tables. In addition, the MRIOLab also includes functions for MRIO analysis.

However, the MRIOLab suite is not well suited for the analysis of MRIO tables compiled independent of the MRIOLab since these can not fully reproduced within the MRIOLab [14, 15].

As a consequence, MRIO analysis today relies on often ad-hoc produced scripts and analysis functions, hindering reproducibility of results and reuse of previous coding efforts. Here I present the open source tool Pymrio, a Python 3 package, which aims to close this method gap for EE MRIO analysis.

The article proceeds with a description of the architecture of Pymrio, including the mathematical background and implementation details. This is followed by a short tutorial with a simple use case for Pymrio is given. The reuse potential and future development plans are pointed out at the end of the article.

Implementation and architecture

Mathematical Background

This section gives an overview about the mathematical background in Pymrio.

Generally, mathematical routines implemented in Pymrio follow the equations described below. If, however, a more efficient mechanism was available this was prefered. In this cases the original formula remains as comment in the source code. This was generally the case when numpy broadcasting [16] was available for a specific operation, resulting in a substantial speed up of the calculations.

The Input-Output analysis implemented in Pymrio follows the classic Leontief demand-style modeling [17]. To do so, MRIO tables describe the global inter-industries flows within and across countries for k countries with a transaction matrix Z:

$$Z = \begin{pmatrix} Z_{1,1} & Z_{1,2} & \cdots & Z_{1,k} \\ Z_{2,1} & Z_{2,2} & \cdots & Z_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{k,1} & Z_{k,2} & \cdots & Z_{k,k} \end{pmatrix} \tag{1}$$

Each submatrix on the main diagonal ($Z_{i,i}$) represent the domestic interactions for each industry n. The off diagonal matrices ($Z_{i,j}$) describe the trade from region i to region j (with $i, j = 1, \ldots, k$) for each industry. Accordingly, global final demand can be represented by

$$Y = \begin{pmatrix} Y_{1,1} & Y_{1,2} & \cdots & Y_{1,k} \\ Y_{2,1} & Y_{2,2} & \cdots & Y_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{k,1} & Y_{k,2} & \cdots & Y_{k,k} \end{pmatrix} \tag{2}$$

with final demand satisfied by domestic production in the main diagonal ($Y_{i,i}$) and direct import to final demand from country i to j by $Y_{i,j}$.

The global economy can thus be described by:

$$x = Ze + Ye \tag{3}$$

with e representing the summation vector (column vector with 1's of appropriate dimension) and x the gross output.

The direct requirement matrix A is given by multiplication of Z with the diagonalised and inverted gross output x:

$$A = Z\hat{x}^{-1} \tag{4}$$

Based on the linear economy assumption of the IO model, gross output x can than be determined for any arbitrary vector of final demand y by multiplying with the total requirement matrix (Leontief matrix) L.

$$x = (I - A)^{-1}y = Ly \tag{5}$$

IO systems can be extended with various exentions (satellite accounts). Among others these can represent factors of production (e.g. value added, employment) and environmental stressors associated with production. These direct factor F can be normalized to the output per sector x by

$$S = F\hat{x}^{-1} \tag{6}$$

Multipliers for F are obtained by

$$M = SL \tag{7}$$

If parts of the environmental stressors occuring during the final use of product, these can be represented by FY (e.g. household emissions).

Production based accounts (direct territorial requirements) per country are therefore given by:

$$D_{pba} = Fe + FYe \tag{8}$$

Total requirements (footprints in case of environmental requirements) for any given final demand vector y are than given by

$$D_{cba} = My + FYe \tag{9}$$

Total requirements (footprints in case of environmental requirements) for any given final demand vector y are than given by

$$D_{cba} = My \tag{10}$$

Setting the domestically satisfied final demand $Y_{i,i}$ to zero ($Y_t = Y - Y_{i,j} \mid i = j$) allow to calculate the factor of production occurring abroad (embodied in imports)

$$D_{imp} = SMY_t \tag{11}$$

The factors of production occurring domestically to satisfy final demand in other countries is given by:

$$D_{exp} = S M \widehat{Y_t} e \tag{12}$$

Aggregation

$$S_k = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{w,1} & b_{w,2} & \cdots & b_{w,k} \end{pmatrix} S_n = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{x,1} & b_{x,2} & \cdots & b_{x,n} \end{pmatrix} \tag{13}$$

With w and x defining the aggregated number of countries and sectors, respectively. Entries b are set to 1 if the sector/country of the column belong to the aggregated sector/region in the corresponding row and zero otherwise. The complete aggregation matrix S is given by the Kronecker product of S_k and S_n:

$$S = S_k \otimes S_n \tag{14}$$

The aggregated IO system can than be obtained by

$$Z_{agg} = S Z S^{\mathrm{T}} \tag{15}$$

and

$$Y_{agg} = S Y (S_k \otimes \mathrm{I})^{\mathrm{T}} \tag{16}$$

with I defined as the identity matrix with the size the final demand categories per country.

Factor of production are aggregated by

$$F_{agg} = F S^{\mathrm{T}} \tag{17}$$

and stressors occuring during final demand by

$$FY_{agg} = FY (S_k \otimes \mathrm{I})^{\mathrm{T}} \tag{18}$$

Implementation

The main design principle of Pymrio is based on the idea that an EE MRIO systems can be effectively represented as an object in an Object Oriented Programming (OOP) language. In Pymrio, such an EE MRIO object consists of a core component describing the economic relationships grouped with a various number of components describing the environmental and/or social extensions (satellite accounts, see Figure 1). All components of the main object are in turn represented as objects, allowing to implement specific methods for each sub-component.

This architecture described above was implemented in Python 3.5. The various tables of the MRIO system are stored in Pandas DataFrames [18], therefore building upon a well-tested data-science framework. As a consequence, besides the specific
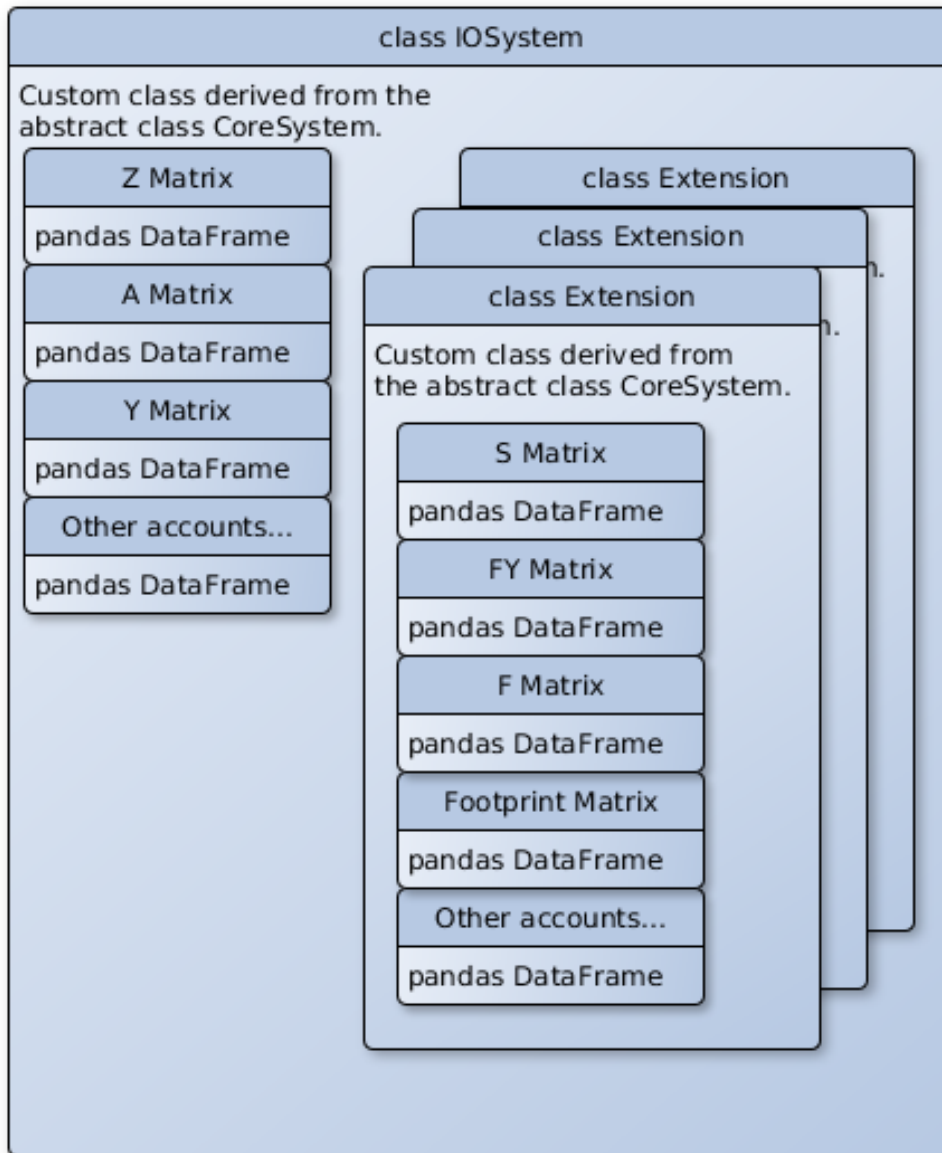
Figure 1: Class diagramm of the core Pymrio class. The composite class IOSystem consists of the economic core with the actual data stored in Pandas DataFrames and a variable number of Extension classes. Each Extension consists of multiple Pandas DataFrames. Both, the IOSystem and Extension class are derived from an abstract CoreSystem class implementing the shared functionality of both classes. Class methods are not depicted here.

methods implemented by Pymrio, the full functionality of Pandas and the underlying numpy framework [16] can be used to modify the MRIO data.

Methods implemented in Pymrio which go beyond basic Pandas functionality are accompanied by a corresponding test harness which ensures the formal correctness of the method. The full source code is hosted on a public code repository [19] together with an extensive documentation and tutorials. Pymrio is openly available under

the GNU General Public License v3.0.

Parsing and Storage

To date, no standard way of storing MRIO databases has been defined. For example, the WIOD database is provided as xlsx tables, whereas Eora and EXIOBASE use (compressed) csv tables. For the two latter, however, the approach differs as Eora26 used pure numerical tables with separate files describing the headers, whereas EXIOBASE use csv tables which include the headers. To ease the use of different MRIO systems, Pymrio include parsers for the different formats. After parsing a MRIO system, Pymrio stores all data in a consistent way. For each component (core system and extension) data is stored in a separate folder. While the storage-format of the actual numerical data can be defined by the user, each storage folder also contains a json file (file_parameters.json) which stored information about the used format. Using the common json file format for storing the file meta data allows to easily and automatically import the data in other programming environments. By default, each tables is stored as a tab-separated text file format, including row and column headers. The folder with the economic core also contains a file metadata.json which stores information on version, name and system (industry by industry or product by product) as well as a record of modifications to the particular MRIO system (including when it was downloaded, any aggregations, removal/addition of extensions, etc.).

Usage

The following section provides a quick start guide for using Pymrio beginning at the installation followed by a basic input-output calculation example. For the example here, the WIOD MRIO database [20] is used. However, after downloading and parsing the database the same methods are available for any EE MRIO system.
The code examples here are included as interactive Jupyter notebook tutorial in the SI. As the code example here only show the code inputs, refer to the notebook to see the output of a specific command. An cloud-based virtual environment with the code example is available at `https://mybinder.org/v2/gh/konstantinstadler/pymrio_article/master?filepath=%2Fnotebook%2Fpymrio-tutorial-for-wiod.ipynb`.
Pymrio is a Python [21] package, Python version $\geq$ 3.5 is required. The Pymrio packages is hosted on PyPI [22] and the Anaconda Cloud [23]. Therefore, you can either use

```
pip install pymrio ——upgrade
```

or

```
conda install −c konstantinstadler pymrio
```

to install Pymrio and all required packages.
Pymrio can than be used in any Python programming environment.
Throughout the code examples below, it is assumed that Pymrio is imported as follows:

import pymrio

First, the Pymrio MRIO download function is used to get the WIOD MRIO database with:

```
raw_wiod_path = '/tmp/wiod/raw'
pymrio.download_wiod2013(storage_folder=raw_wiod_path,
                         years=[2008])
```

This downloads the 2008 MRIO table from WIOD. Omitting the year parameter would result in a download of all years. The function returns a Pymrio meta data object, which gives information about the WIOD version, system (in this case industry by industry) and records about from where the data was received (see SI cell 6).
To parse the database into a Pymrio object use:

```
wiod = pymrio.parse_wiod(raw_wiod_path, year=2008)
```

The available data can be explored by for example

```
wiod.get_sectors()
wiod.get_regions()
```

The transaction matrix can be inspected with

```
wiod.Z
```

which returns a panda DataFrame with the recoreded monetary flows.
WIOD includes several extensions, which are stored as sub-objects (see Figure 1) in Pymrio. For example, in order to see the AIR emissions provided by WIOD:

```
wiod.AIR.F
```

WIOD, however, does neither provide any normalized data (A-matrix, satellite account coefficient data) nor any consumption based accounts (footprints).
In order to calculate them, one could go through all the missing data and compute each account. Pymrio provides the required functions, for example to calculate the A-matrix:

```
x = pymrio.calc_x(Z=wiod.Z, Y=wiod.Y)
A = pymrio.calc_A(Z=wiod.Z, x=x)
```

Alternativly, Pymrio provides a function which finds all missing accounts and calculates them:

```
wiod.calc_all()
```

At this point, a basic EE MRIO analysis is accomplished. For example, the regional consumption based accounts of the AIR emissions are now given by:

```
wiod.AIR.D_cba_reg
```

Units are stored seperately in

```
wiod.AIR.unit
```

Pymrio can be linked with the country converter coco [24] to ease the aggregation of MRIO and results into different classifications. Using the country converter, WIOD can easily be aggregated into EU and non-EU countries with singling out Germany and the UK by:

```
import country_converter as coco
wiod.aggregate(region_agg = coco.agg_conc(
    original_countries='WIOD',
    aggregates=[{'DEU': 'DEU', 'GBR':'GBR'}, 'EU'],
    missing_countries='Other',
    merge_multiple_string=None))
wiod.rename_regions({'EU':'Rest_of_EU'}
```

To visualize the results for example for CH_4 the matplotlib framework [25] can be used (Figure 2):

```
import matplotlib.pyplot as plt
with plt.style.context('ggplot'):
    wiod.AIR.plot_account('CH4')
    plt.savefig('airch4.png', dpi=300)
    plt.show()
```
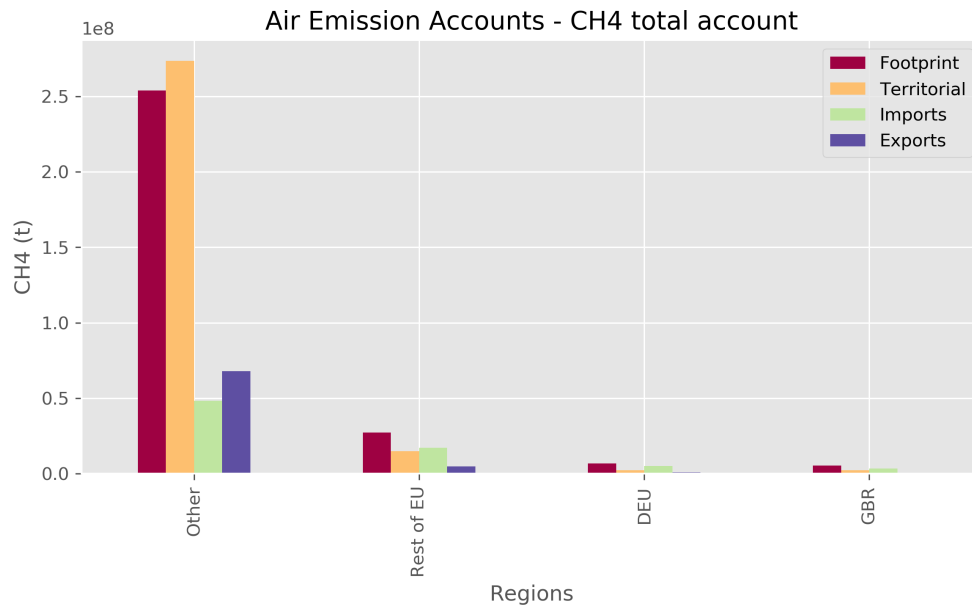

Figure 2: CH_4 emissions of Germany (DEU), the UK (GBR), Rest of the EU and Other countries This figure was produced with Pymrio and matplotlib after aggregating the WIOD countries into the three regions specified above)

To calculate the source (in terms of regions and sectors) of a certain stressor or impact driven by consumption, one needs to diagonalize this stressor/impact. This can be done with Pymrio by:

```
diag_CH4 = wiod.AIR.diag_stressor('CH4')
```

and be reassigned to the aggregated WIOD system:

```
wiod.CH4_source = diag_CH4
```

In the next step the automatic calculation routine of Pymrio is called again to compute the missing accounts in this new extension:

```
wiod.calc_all()
```

The diagonalzied CH4 data now shows the source and destination of the specified stressor (CH_4):

```
wiod.CH4_source.D_cba
```

In this square consumption based accounts matrix, every column represents the amount of stressor occurring in each region - sector driven by the consumption stated in the column header. Conversely, each row states where the stressor impacts occurring in the row are distributed to (from where they are driven).

If only one specific aspect of the source is of interest for the analysis, the footprint matrix can easily be aggregated with the standard Pandas *groupby* function. For example, to aggregate to the source and receiving region of the stressor:

```
CH4_source_reg = wiod.CH4_source.D_cba.groupby(
    level='region', axis=0).sum().groupby(
    level='region', axis=1).sum()
```

Which can than be visualised using the seaborn heatmap [26] with (Figure 3):

```
import seaborn as sns
CH4_source_reg.columns.name = 'Receiving_region'
CH4_source_reg.index.name = 'Souce_region'
sns.heatmap(CH4_source_reg, vmax=5E6,
            annot=True, cmap='YlOrRd', linewidths=0.1,
            cbar_kws={'label': 'CH4_emissions_({})'.format(
                wiod.CH4_source.unit.unit[0])})
plt.show()
```

Storing the MRIO database can be done with

```
storage_path = '/tmp/wiod/aly'
wiod.save_all(storage_path)
```

from where it can be received subsequently by:

```
wiod = pymrio.load_all(storage_path)
```

The meta attribute of Pymrio mentioned at the beginning kept track of all modifications of the system. This can be shown with:

```
wiod.meta
```

Custom notes can be added to the history with:

```
wiod.meta.note("Custom_note")
```

The history of the meta data can be filtered for specific entries like:
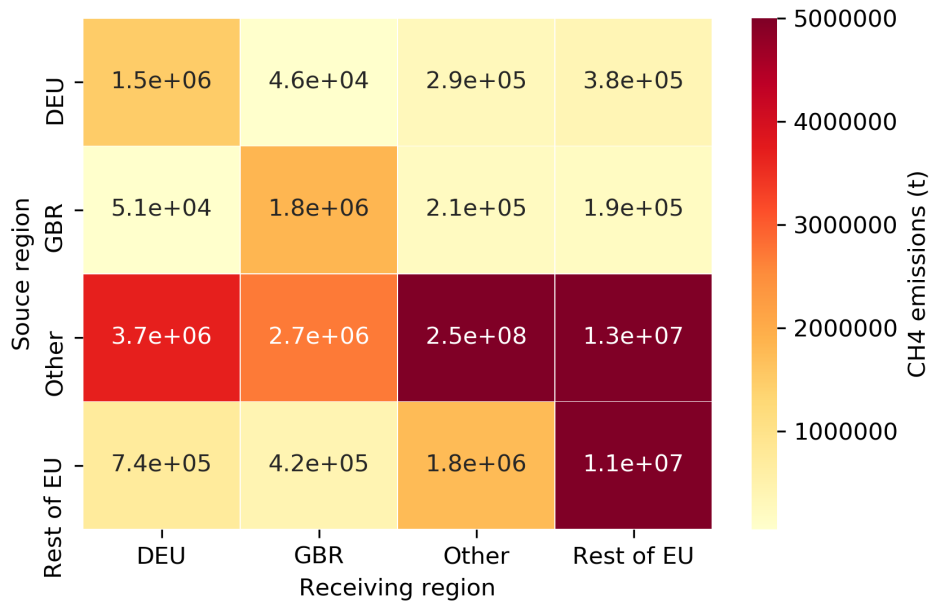
```
wiod.meta.file_io_history
```

Figure 3: CH_4 emissions source and destination A substantial share of CH_4 originating in the Rest of the World region are exported into Germany and the UK. This figure was produced with Pymrio and seaborn.

This tutorial gave a short overview about the basic functionality of Pymrio. For more information about the capabilities of Pymrio check the online documentation at `http://pymrio.readthedocs.io` [27]

Quality control
Detail the level of testing that has been carried out on the code (e.g. unit, functional, load etc.), and in which environments. If not already included in the software documentation, provide details of how a user could quickly understand if the software is working (e.g. providing examples of running the software with sample input and output data).
points
Issue driven development pytest, with automatic travis. Importantly all mathematical functionality is unit tested Travis pep8 (required to pass tests before merging pull requests) numpy style docstrings test mrio system distributed with the source code, allowing to reproduce the tutorials given in the docuementation
AFAP: Since pymrio is already used in research projects, please aim for keeping compatibility with previous versions.
For collaborators, these points are outlined in the Contribution section of the documentation - TODO: readme as reference
text pieces:
To insure a functional master branch of Pymrio, a continuous integration platform run by Travis CI builds the package from source and runs the unit tests upon each commit. These same checks are run on pull requests submitted by contributors. These also check pep8 ... If these tests fail, the developers are notified so that

modifications to the code can be made to fix these issues prior to merging the changes into Pymrio.

Documentation is is built using the Sphinx documentation generator [22] that creates the HTML documentation, hosted on readthedocs. This documentation is autoamtically build (ensure that code and documentation is in sync) and includes a full API reference based on the information provided in the docstrings. The examples and tutorials given the in the documentation are based on Jupyter notebooks which are recalculated for each release, thus also serving as a regression test for documented Pymrio functionality.

(2) Availability

Operating system
GNU/Linux, Mac OSX, Windows and any other operating systems running Python.

Programming language
Please include minimum version compatibility. As tested in Travis, note that the main dependency is pandas and new features are developed against the newest version of pandas

Additional system requirements
E.g. memory, disk space, processor, input devices, output devices., testsystem should run on every computer capabable of running python, further requriments depend on the database to be analysed, for EXIOBASE minimum of 8 GB of RAM

Dependencies
E.g. libraries, frameworks, incl. minimum version compatibility.

List of contributors
Please list anyone who helped to create the software (who may also not be an author of this paper), including their roles and affiliations.

Software location:
Archive (e.g. institutional repository, general repository) (required please see instructions on journal website for depositing archive copy of software in a suitable repository)

> **Name:** The name of the archive.
> **Persistent identifier:** e.g. DOI, handle, PURL, etc.
> **Licence:** Open license under which the software is licensed.
> **Publisher:** Name of the person who deposited the software.
> **Version published:** The version number of the software archived.
> **Date published:** dd/mm/yy

Code repository (e.g. SourceForge, GitHub etc.) (required)

> **Name:** The name of the archive.
> **Persistent identifier:** e.g. DOI, handle, PURL, etc.
> **Licence:** Open license under which the software is licensed.
> **Date published:** dd/mm/yy

Emulation environment (if appropriate)
> **Name:** The name of the archive.
> **Persistent identifier:** e.g. DOI, handle, PURL, etc.
> **Licence:** Open license under which the software is licensed.
> **Date published:** dd/mm/yy

Language
Language of repository, software and supporting files.

(3) Reuse potential

Pymrio contains functionality aimed at professional MRIO analysts and sustainability scientist, but might be useful to anyone doing environmental and/or economic analysis. As such, Pymrio is one key component in the Industrial Ecology analysis software framework [10]. With the other components it shares the ambition to improve usability, interoperability, and collaboration between Industrial Ecology and sustainbility research Python packages. The main motiviation for starting the project was to enable a common interface for handling differrent MRIO databases, but through the years the scope extended to include visualization, reporting and data provenance tracking capabilities. Future development plans include further visualization possibilites, parser for additional MRIO models and extended analysis capabilites like structural decomposition and structural path analysis. Being an open source project, this includes an invitation to fellow researchers to join these coding efforts.
Please describe in as much detail as possible the ways in which the software could be reused by other researchers both within and outside of your field. This should include the use cases for the software, and also details of how the software might be modified or extended (including how contributors should contact you) if appropriate. Also you must include details of what support mechanisms are in place for this software (even if there is no support).

Competing interests
The authors declare that they have no competing interests.

References
TODO: copy paste as numbers before submission

References

[1] Davis, S.J., Caldeira, K., Matthews, H.D.: Future CO2 Emissions and Climate Change from Existing Energy Infrastructure. Science **329**(5997), 1330–1333 (2010). doi:10.1126/science.1188566

[2] Ivanova, D., Vita, G., Steen-Olsen, K., Stadler, K., Melo, P.C., Wood, R., Hertwich, E.G.: Mapping the carbon footprint of EU regions. Environmental Research Letters **12**(5), 054013 (2017). doi:10.1088/1748-9326/aa6da9

[3] Tukker, A., Bulavskaya, T., Giljum, S., de Koning, A., Lutter, S., Simas, M., Stadler, K., Wood, R.: Environmental and resource footprints in a global context: Europe's structural deficit in resource endowments. Global Environmental Change **40**, 171–181 (2016). doi:10.1016/j.gloenvcha.2016.07.002

[4] Verones, F., Huijbregts, M.A.J., Chaudhary, A., de Baan, L., Koellner, T., Hellweg, S.: Harmonizing the assessment of biodiversity effects from land and water use within LCA. Environmental Science & Technology (2015). doi:10.1021/es504995r

[5] Wood, R., Stadler, K., Simas, M., Bulavskaya, T., Giljum, S., Lutter, S., Tukker, A.: Growth in Environmental Footprints and Environmental Impacts Embodied in Trade: Resource Efficiency Indicators from EXIOBASE3: Growth in Environmental Impacts Embodied in Trade. Journal of Industrial Ecology **22**(3), 553–564 (2018). doi:10.1111/jiec.12735

[6] Wood, R., Moran, D., Stadler, K., Ivanova, D., Steen-Olsen, K., Tisserant, A., Hertwich, E.G.: Prioritizing Consumption-Based Carbon Policy Based on the Evaluation of Mitigation Potential Using Input-Output Methods: Prioritizing Consumption-Based Carbon Policies. Journal of Industrial Ecology **22**(3), 540–552 (2018). doi:10.1111/jiec.12702

[7] Mutel, C.: Brightway: An open source framework for Life Cycle Assessment. The Journal of Open Source Software **2**(12), 236 (2017). doi:10.21105/joss.00236

[8] openLCA: openLCA Modeling Suite — openLCA.Org. http://www.openlca.org/openlca/ (2018)

[9] Cencic, O., Rechberger, H.: Material flow analysis with software STAN. Journal of Environmental Engineering and Management **18**(1), 3 (2008)

[10] Pauliuk, S., Majeau-Bettez, G., Müller, D.B.: A General System Structure and Accounting Framework for Socioeconomic Metabolism. Journal of Industrial Ecology **19**(5), 728–741 (2015). doi:10.1111/jiec.12306

[11] Nazara, S., Guo, D., Hewings, G.J.D., Dridi, C.: PyIO - Input-Output Anaysis with Python. Technical Report 03-T-23, The University of Illinois at Urbana-Champaign, Illinois, USA (2003)

[12] Geschke, A., Hadjikakou, M.: Virtual laboratories and MRIO analysis – an introduction. Economic Systems Research **29**(2), 143–157 (2017). doi:10.1080/09535314.2017.1318828

[13] Lenzen, M., Geschke, A., Rahman, M.D.A., Xiao, Y., Fry, J., Reyes, R., Dietzenbacher, E., Inomata, S., Kanemoto, K., Los, B., Moran, D., in den Bäumen, H.S., Tukker, A., Walmsley, T., Wiedmann, T., Wood, R., Yamano, N.: The Global MRIO Lab – charting the world economy. Economic Systems Research **29**(2), 158–186 (2017). doi:10.1080/09535314.2017.1301887

[14] Rahman, M.D.A., Los, B., Geschke, A., Xiao, Y., Kanemoto, K., Lenzen, M.: A flexible adaptation of the WIOD database in a virtual laboratory. Economic Systems Research **29**(2), 187–208 (2017). doi:10.1080/09535314.2017.1318115

[15] Reyes, R.C., Geschke, A., de Koning, A., Wood, R., Bulavskaya, T., Stadler, K., in den Bäumen, H.S., Tukker, A.: The Virtual IELab – an exercise in replicating part of the EXIOBASE V.2 production pipeline in a virtual laboratory. Economic Systems Research **29**(2), 209–233 (2017). doi:10.1080/09535314.2017.1317237

[16] van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering **13**(2), 22–30 (2011). doi:10.1109/MCSE.2011.37

[17] Leontief, W.: Environmental Repercussions and the Economic Structure: An Input-Output Approach. The Review of Economics and Statistics **52**(3), 262–271 (1970). doi:10.2307/1926294. ArticleType: research-article / Full publication date: Aug., 1970 / Copyright © 1970 The MIT Press

[18] McKinney, W.: Data Structures for Statistical Computing in Python, pp. 51–56 (2010)

[19] Stadler, K.: Pymrio: Multi-Regional Input-Output Analysis in Python (2018)

[20] Timmer, M.P., Dietzenbacher, E., Los, B., Stehrer, R., de Vries, G.J.: An Illustrated User Guide to the World Input–Output Database: The Case of Global Automotive Production. Review of International Economics, (2015). doi:10.1111/roie.12178

[21] Python: Official Python Webpage. https://www.python.org/ (2018)

[22] PyPI: PyPI - the Python Package Index : Python Package Index. https://pypi.python.org/pypi (2018)

[23] Inc., A.: Anaconda Cloud. https://anaconda.org/about (2018)

[24] Stadler, K.: The country converter coco - a Python package for converting country names between different classification schemes. The Journal of Open Source Software **2**(16) (2017). doi:10.21105/joss.00332

[25] Hunter, J.D.: Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering **9**(3), 90–95 (2007). doi:10.1109/MCSE.2007.55

[26] Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D.C., Augspurger, T., Halchenko, Y., Cole, J.B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, A., Ram, Y., Yarkoni, T., Williams, M.L., Evans, C., Fitzgerald, C., Brian, Fonnesbeck, C., Lee, A., Qalieh, A.: Mwaskom/seaborn: V0.8.1 (September 2017). Zenodo (2017). doi:10.5281/zenodo.883859

[27] Stadler, K.: Pymrio - Multi Regional Input Output Analysis in Python — Pymrio 0.3.3 Documentation. http://pymrio.readthedocs.io/en/latest/# (2018)

Please enter references in the Harvard style and include a DOI where available, citing them in the text with a number in square brackets, e.g.

[1] Piwowar, H A 2011 Who Shares? Who Doesn't? Factors Associated with Openly Archiving Raw Research Data. PLoS ONE 6(7): e18657. DOI: http://dx.doi.org/10.1371/journal.pone.0018657.