Software paper for submission to the Journal of Open Research Software
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 parsers for several openly available EE MRIO databases (EXIOBASE v1 - v3, 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 Analysis, Footprinting, Consumption-Based Accounting, EXIOBASE, WIOD, Eora26, OECD-ICIO, Python

Introduction
Environmentally Extended Multi-Regional Input-Output (EE MRIO) tables describe economic relationships within and between regions and their environmental repercussions.
They capture the full life-cycle impacts of goods and services across international supply chains and therefore enable researchers to understand the environmental, social and economic consequences of consumption in todays globalised world. Research questions that can be answered using MRIO tables range from EXAMPLES (carbon, trade, job, biodiversity, human needs, see also que vadis wiedmann).
Given their potential to address this variety of sustainability questions, EE MRIOs analysis has become one of the prevailing methodology in Industrial Ecology and quantitative sustainability science [**davis2010˙Consumptionbased**, **ivanova2017˙Mapping**, **tukker2016˙Environmental**, **verones2015˙Harmonizing**, **wood2018˙Growth**, **wood2018˙Prioritizing**].

Historically, MRIOs developed from single country input-output tables DESCRIBE, THEN REGIONAL, GLOBAL. Note that Pymrio can be used for all of them when a consistent sector classification is used.
Despite their long history, no common framework...
True (full) multi-region models endogenously combine domestic technical coefficient matrices with import matrices from multiple countries or regions into one large coefficient matrix, thus capturing trade supply chains between all trading partners as well as feedback effects. The latter are changes in production in one region that result from changes in intermediate demand in another region, which are in turn brought about by demand changes in the first region (see Miller, 1969, p.41).4 It is worth noting at this point that the term MRIO, for multi- regional input–output model, has its origin in regional economics, interested in differences in production technologies between regions within a nation. The first such subnational applications to the US economy can be traced back to the 1950s and a distinction between MRIO and IRIO, for inter-regional input–output model, was made early on. The differences between the two approaches are grounded in the way regional technology coefficients and inter-regional trade coefficients are calculated (Miller and Blair, 2009, Chapter 3; see also Guo et al., 2009, Box 1, p.7). The MRIO method is a simplification of the IRIO method designed to deal with the most common of data limitations. Inter-regional input–output modelling was first formulated by Isard (1951) who extended the classical national Leontief model to allow for regional analyses. Studies that followed early on were aimed at improving the computability of IRIO analysis but were not aimed at environmental applications (Chenery, 1953; Leontief and Strout, 1963; Leontief, 1953; Miernyk, 1973; Moses, 1955; Riefler and Tiebout, 1970; Riefler, 1973). Polenske (1976, 1980) examines the economic interactions and repercussions between the coal mining, freight transport and electricity generation sectors in nine regions of the USA. Although the input–output community swiftly turned its attention to the world economy — and therefore international or multi-national models (most famously in Leontief's world model; Leontief, 1974) — the term MRIO has only in the last decade begun to be used for models with entire countries or clusters of countries as regions. Particular assumptions, conventions, data sources, and methods have been developed alongside. For the purpose of this paper I use the term MRIO for all international and subnational input–output models with more than one region. It should be born in mind, however, that not all MRIO approaches have the same mathematical form and modelling background.
Tukker Dietzenbacher Similar issues have arisen in the trade literature as succinctly reflected by the titles of some papers, such as "Who produces for whom in the world economy?" (Daudin et al., 2011) or "Give credit where credit is due: tracing value added in global production chains" (Koopman et al., 2010). Production processes have increasingly become sliced up (or fragmented) into ever smaller parts. Many of these parts are outsourced to specialized subcontractors that are more and more located in foreign countries (i.e. offshoring). This has led to an upsurge of trade in intermediate products because the location of the production of intermediate inputs differs from the location of the production of the final products (which corresponds to Baldwin's, 2006, 'second wave of global unbundling'). Today's products and services are no longer produced within a single country. Instead, they are made in

global supply chains, or global value chains. That is, countries import intermediate goods and raw materials, to which they add one or more layers of value after which they sell the product (often to a foreign producer who adds the next layer).
regions within a nation by Isard (1951). MRIO tables have become a widely discussed topic in the regional science literature and a widely used tool for regional policy. The textbook by Miller and Blair (2009) provides an excellent overview and a thorough introduction to MRIO tables and models. At the same time, it should also be emphasized that the regional
FOR DISCUSSION/USE:Need for improvment based on use - make use easy to have a larger userbase and advance the develoment of MRIO models

---

In contrast to other Industrial Ecology methodologies like Life Cycle Assessment (e.g. Brightway [**mutel2017˙Brightway**], openLCA [**openlca2018˙openLCA**]) or Material Flow Analysis (e.g. STAN [**cencic2008˙Material**]), few generally available analysis packages for (Multi-Regional) Input-Output tables are available [**pauliuk2015˙Lifting**]. One of the very few well documented and stand alone packages for IO analysis, PyIO [**nazara2003˙PyIO**], is not available for Python 3 and does not provide parsers for current MRIO databases (the package has not been updated since 2011 but is still available at `http://www.real.illinois.edu/pyio`).
Another framework, the MRIOLab suite [**lenzen2017˙Global**, **geschke2017˙Virtual**] takes a different approach: it provides a virtual lab for the compilation of MRIO tables and thereby streamlining the compilation of MRIO tables. The MRIOLab also includes functions for MRIO analysis. These, however, are not well suited for the analysis of MRIO tables compiled independent of the MRIOLab since the tables can not fully reproduced within the MRIOLab [**reyes2017˙Virtual**, **rahman2017˙flexible**].
As a consequence of the lack of a generic MRIO analysis toolkit, MRIO analysis today relies on often ad-hoc produced scripts and functions. This hinders reproducibility of results and the 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. This tutorial is also available as an IPython notebook. 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 of EE MRIO analysis as used in Pymrio.
TODO: The focus of this section are on the specifics of the multi-regional aspects of input-output analysis. Note, however, that Pymrio can also be used for analysing simple single country input-output tables. Readers unfamiliar with input-output analysis are referred to the standard text-book for input-output analysis by Miller

and Blair (CITATION). Exercises of this book are also used in the online documentation which also highlights how Pymrio can be used for teaching of input-output methods (LINK).
Generally, mathematical routines implemented in Pymrio follow the equations described below. If, however, a more computationally efficient mechanism was available this was preferred. In these cases the original formula remains as comment in the source code. Mostly, this was the case when instead of a matrix multiplication with a diagonalized vector the operation can be efficiently executed using numpy broadcasting mechanisms [**vanderwalt2011˙NumPy**].
The Input-Output analysis implemented in Pymrio follows the classic Leontief demand-style modeling [**leontief1970˙Environmental**]. 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 ($Z_{i,i}$) represents 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 then be determined for any arbitrary vector of final demand y by multiplying with the total requirement matrix (Leontief matrix) L.

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

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

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

Multipliers (total, direct and indirect, requirement factors for one unit of output) are then obtained by

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

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

$$D_{cba} = My \tag{8}$$

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

$$D_{imp} = SLY_t \tag{9}$$

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

$$D_{exp} = S\widehat{LY_t}e \tag{10}$$

where $\widehat{\ }$ indicates diagonalization of the resulting column-vector of the term underneath.
If parts of the environmental stressors occurring during the final use of product, these can be represented by F_Y (e.g. household emissions). These need to be added to the total production- and consumption-based accounts to obtain the total impacts per country. Production-based accounts (direct territorial requirements) per region i are therefore given by summing over the stressors per sector $(0 \dots m)$ plus the stressors occurring due to the final consumption for all final demand categories $(0 \dots w)$ of that region.

$$D^i_{pba} = \sum_{s=0}^{m} F^i_s + \sum_{c=0}^{w} F^{i_c}_Y \tag{11}$$

Similarly, total requirements (footprints in case of environmental requirements) per region i are given by summing the detailed footprint accounts and adding the aggregated final demand stressors.

$$D^i_{cba} = \sum_{s=0}^{m} D^{i_s}_{cba} + \sum_{c=0}^{w} F^{i_c}_Y \tag{12}$$

Internally, the summation are implemented with the *group-by* functionality provided by the pandas package.

Aggregation

For the aggregation of the MRIO system the matrix B_k defines the aggregation matrix for regions and B_n the aggregation matrix for sectors.

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

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 B is given by the Kronecker product of B_k and B_n:

$$B = B_k \otimes B_n \tag{14}$$

This effectively arranges the sector aggregation matrix B_n as defined by the region aggregation matrix B_k. Thus, for each 0 entry in B_k a block $B_n * 0$ is inserted in B and each 1 corresponds to $B_n * 1$ in B.
The aggregated IO system can then be obtained by

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

and

$$Y_{agg} = BY(B_k \otimes \mathrm{I})^{\mathrm{T}} \tag{16}$$

with I defined as the identity matrix with the size equal to the number of final demand categories per country.
Factors of production are aggregated by

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

and stressors occurring during final demand by

$$F_{Y_{agg}} = F_Y(B_k \otimes \mathrm{I})^{\mathrm{T}} \tag{18}$$

Implementation

The main design principle of Pymrio is based on the concept that an EE MRIO system 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 3.7. The various tables of the MRIO system are stored in Pandas DataFrames [**mckinney2010˙Data**],

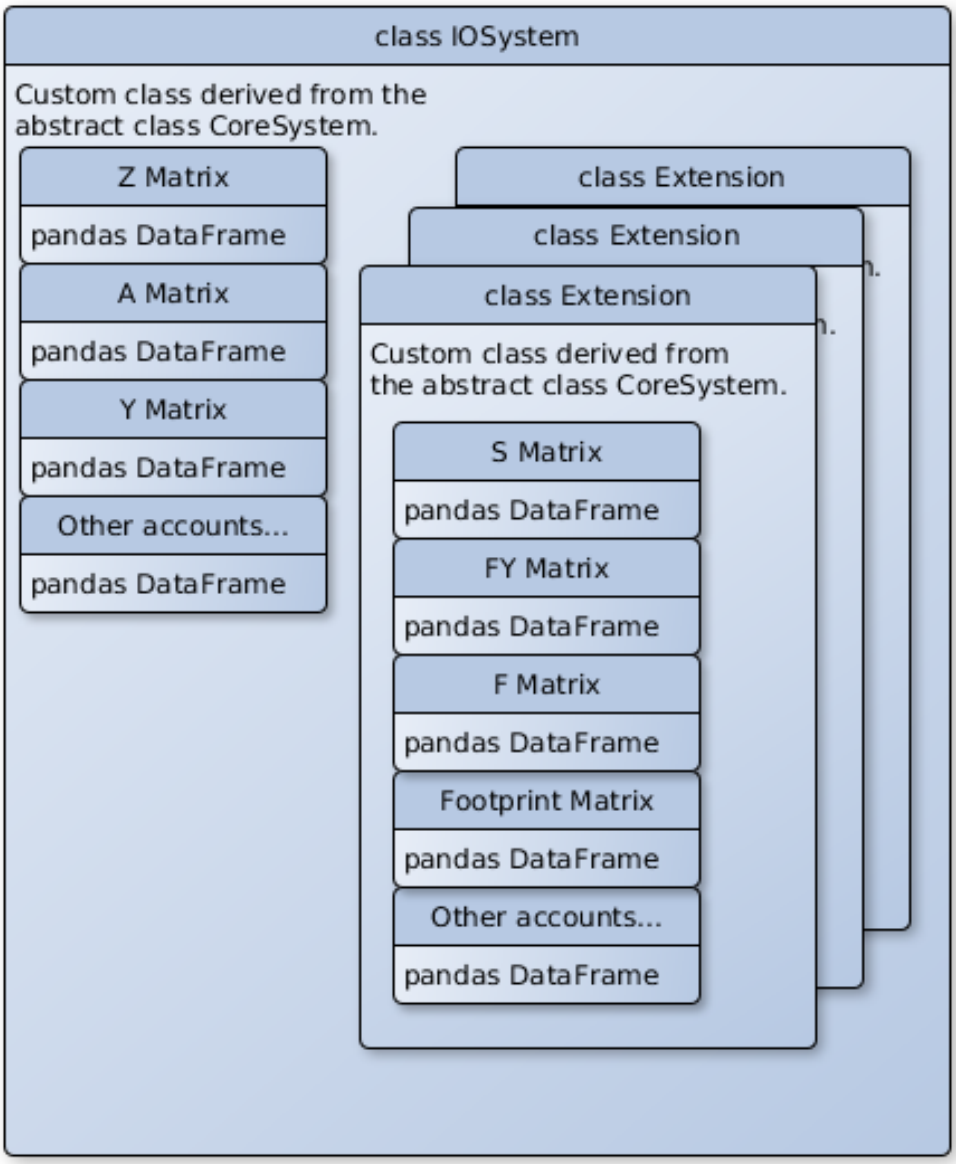

Figure 1: **Class diagramm of the core Pymrio class.** The composite class IOSystem consists of the economic core with the actual data stored in Pandas DataFrames and a variable number of Extension classes. Each Extension consists of multiple Pandas DataFrames. Both, the IOSystem and Extension class are derived from an abstract CoreSystem class implementing the shared functionality of both classes. Class methods are not depicted here.

therefore building upon a well-tested data-science framework. As a consequence, besides the specific methods implemented by Pymrio, the full functionality of Pandas and the underlying numpy framework [**vanderwalt2011˙NumPy**] 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 correct-

ness of the method. The full source code is hosted on a public code repository together with an extensive documentation and tutorials `https://github.com/konstantinstadler/pymrio`. 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 [**timmer2015˙Illustrated**] is provided as xlsx tables, whereas Eora [**lenzen2013˙Building**] and EXIOBASE [**stadler2018˙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 contains 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 table is stored as a tab-separated text file format, including row and column headers. The folder with the economic core also contains a file named "metadata.json" which includes information about 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, applied 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 [**timmer2015˙Illustrated**] 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 `https://git.io/fjjUk`. 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://cutt.ly/vwmOpA3`.

Pymrio is a Python [**python2018˙Official**] package, Python version $\geq$ 3.7 is required. The Pymrio package is hosted on PyPI [**pypi2018˙PyPI**] and the Anaconda Cloud [**anacondainc.2018˙Anaconda**]. Therefore, you can either use

```
pip install pymrio --upgrade
```

or

```
conda install -c konstantinstadler pymrio
```

to install Pymrio and all required packages.
Pymrio can then 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 separetly in

```
wiod.AIR.unit
```

Pymrio can be linked with the country converter coco [**stadler2017˙country**] 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 [**hunter2007˙Matplotlib**] 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()
```

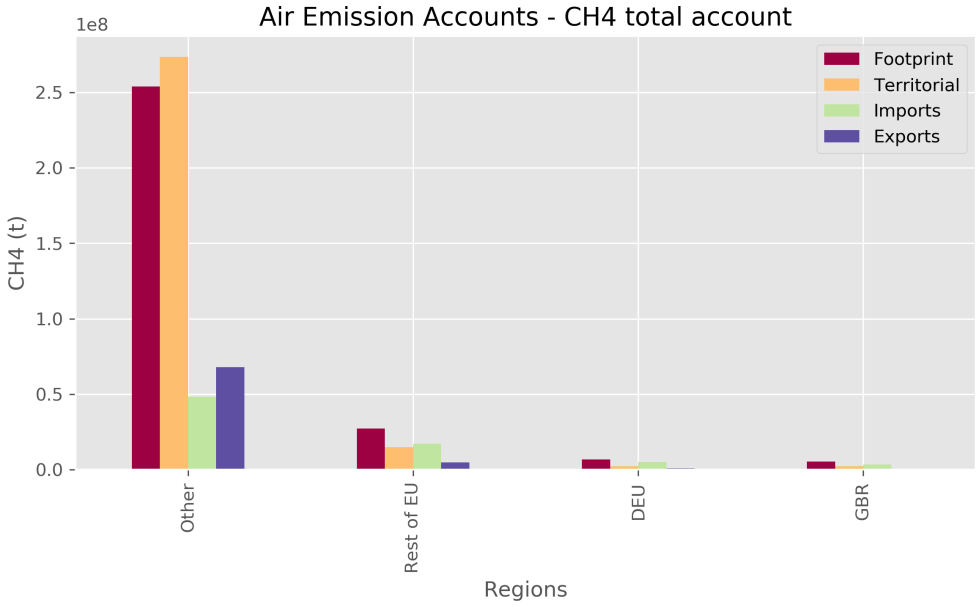

Figure 2: **CH_4 emissions of Germany (DEU), the UK (GBR), Rest of the EU and Other countries** This figure was produced with Pymrio and matplotlib after aggregating the WIOD countries into the three regions specified above)

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 stressors 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 then be visualised using the seaborn heatmap [**waskom2017˙mwaskom**] 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:

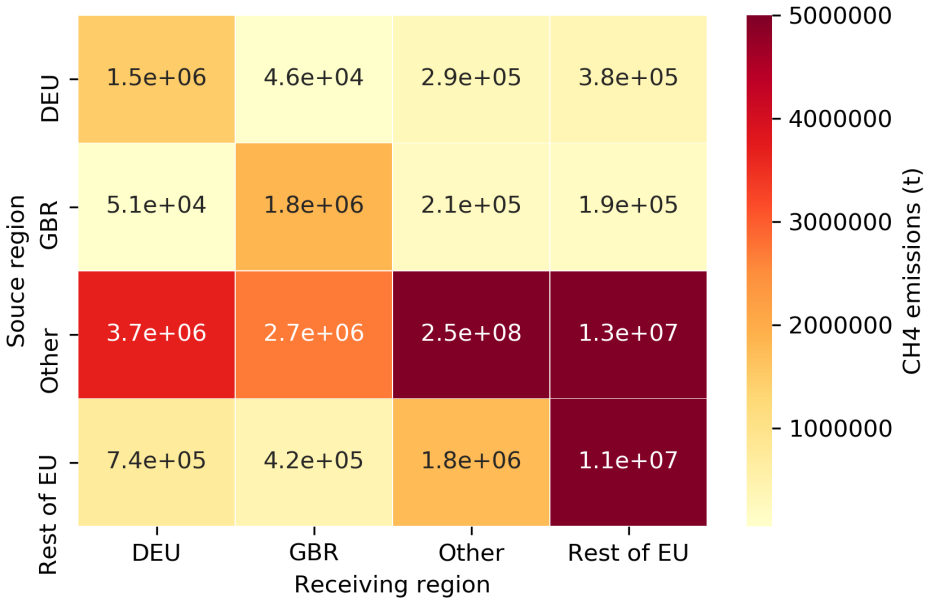

Figure 3: **CH_4 emissions source and destination** A substantial share of CH_4 originating in the Rest of the World region are exported into Germany and the UK. This figure was produced with Pymrio and seaborn.

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

The history of the meta data can be filtered for specific entries like:

```
wiod.meta.file_io_history
```

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` [**stadler2018˙pymrio**]

Quality control
All basic mathematical functions of Pymrio, as described in the section Mathematical Background above, are unit-tested against published results extracted from a classic input-output textbook [**miller2009˙Inputoutput**]. Additional unit tests validate various components of the aggregation, file IO and other utility methods. Beyond the unit tests, Pymrio includes a small pseudo MRIO system which is used for a full regression test of the package. Currently, the test coverage exceeds 90 % `https://cutt.ly/JwPggqV`.
All tests are implemented in pytest and users can verify the correctness of Pymrio, after installing pytest, by

```
py.test -v
```

in the root of the local copy of Pymrio.
In addition, Pymrio uses the continuous integration platform Travis CI for automatic testing after each change of the code base uploaded to the source repository.

After each build, test coverage is automatically calculated using the coveralls platform `https://coveralls.io`.
The Pymrio source code follows the pep8 specifications ; accordance with this code standard is also tested through Travis CI integration. This implies, that any code contribution to Pymrio must follow the pep8 guidelines.
The Pymrio documentation is build using the Sphinx Python Documentation Generator and hosted on readthedocs (`http://pymrio.readthedocs.io`).
After each change to the master branch, the API references in the documentation are automatically updated based on the description provided in the numpy style docstrings, thus keeping the documentation and code base in sync. Furthermore, the code examples and tutorials given the in the documentation are implemented as Jupyter notebooks and are recalculated for each release, thus also serving as regression tests for the documented Pymrio functionality.
Updated results of the Travis CI tests as well as the Sphinx documentation rebuild are indicated at the beginning of the readme file at the source repository.
Contributors to the Pymrio code based are advised to adhere to the testing and code standards established for Pymrio. Further details can be found in the Contribution section of the documentation.

(2) Availability

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

Programming language
Pymrio was built in Python 3 and currently (Pymrio version 0.4.0) tested for Python 3.7.

Additional system requirements
Pymrio runs on every system capable of running the Python SciPy stack. The actual memory requirements depend on the MRIO database to be analysed with Pymrio. For example, for EXIOBASE [**stadler2018˙EXIOBASE**] a minimum of 8 GB RAM are required.

Dependencies
For the current Pymrio version 0.4.0:

- pandas $\geq$ 0.25.0
- numpy $\geq$ 1.13.4
- matplotlib $\geq$ 2.0.0
- requests $\geq$ 2.18
- xlrd $\geq$ 1.1.0
- docutils $\geq$ 0.14

The main dependency of Pymrio is Pandas and future versions of Pymrio will follow the developmental changes in Pandas.
The file requirements.txt in the source repository contains an up-to-date list of all requirements.
For development and unit testing, pytest $\geq$ 3.1 and pytest-pep8 are required.

Software location:
Archive

Name: Zenodo
Persistent identifier: `https://doi.org/10.5281/zenodo.1146054`
Licence: GPL v3
Publisher: Konstantin Stadler
Version published: 0.4.0 and earlier versions. The DOI above always resolves to the latest version, previous versions can be identified with separate DOIs (see versions sections on the Zenodo repository page).
Date published: 12/08/19 (version 0.4.0)

Code repository

Name: Github (Pymrio is also hosted on pypi and anaconda cloud)
Persistent identifier: `https://github.com/konstantinstadler/pymrio`
Licence: GPL v3
Date published: 12/08/19 (version 0.4.0)

Emulation environment

Name: MyBinder Jupyter Notebook of the tutorial included above
Persistent identifier: `https://mybinder.org/v2/gh/konstantinstadler/pymrio_article/master?filepath=%2Fnotebook%2Fpymrio-tutorial-for-wiod.ipynb`
Licence: CC BY 4.0
Date published: 18/01/18

Language
English

(3) Reuse potential
Pymrio contains functionality aimed at professional MRIO analysts and sustainability scientists, 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 [**pauliuk2015˙Lifting**]. With the other components it shares the ambition to improve usability, interoperability, and collaboration between Industrial Ecology and sustainability research Python packages. The main motivation for starting the project was to build a common interface for handling different MRIO databases, but through the years the scope extended to include visualization, reporting and data provenance tracking capabilities. Future development plans include further visualization possibilities, parser for additional MRIO models and extended analysis capabilities like structural decomposition and structural path analysis. Being an open source project, this includes an invitation to fellow researchers to join these coding efforts.

The primary communication channel for Pymrio is the GitHub source repository, in particular the Issue Tracker there. I strongly encouraged to not only use the Issue Tracker for bug reporting but for all questions, comments, and suggestions regarding the project. Pymrio follows an "issue driven development" style. This means that the first step for any modifications or enhancements to Pymrio are to file an issue describing the planned changes. This allows us to discuss changes before the actual programming and gives us the chance to identify synergies across ongoing efforts and avoid potential double work. Finished modification should then be submitted as pull request. Further information about open points and code style can be found in the contributing.rst file at the source repository.

Acknowledgements
Special thanks to Guillaume Majeau-Bettez (Polytechnique Montréal / Norwegian University of Science and Technology), Radek Lonka (Norwegian University of Science and Technology), Richard Wood (Norwegian University of Science and Technology) and Stefan Pauliuk (University of Freiburg) for discussing ideas and the scope of Pymrio as well as implementation details. I am much obliged also to all people already using Pymrio and providing feedback and improvement suggestions.

Funding statement
Parts of the development of Pymrio was funded by the European Commission under the DESIRE Project (grant no.: 308552).

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