pymrio Documentation

Release 0.6.dev

Konstantin Stadler

Jan 19, 2024

CONTENTS

1 Pymrio **3**
- 1.1 What is it . . . 3
- 1.2 Where to get it . . . 4
- 1.3 Quickstart . . . 4
- 1.4 Tutorials . . . 5
- 1.5 Citation . . . 5
- 1.6 Contributing . . . 5
- 1.7 Communication, issues, bugs and enhancements . . . 5
  - 1.7.1 Research notice . . . 5

2 Installation **7**

3 Terminology **9**

4 Mathematical background **11**
- 4.1 Basic MRIO calculations . . . 11
- 4.2 Aggregation . . . 15

5 Automatic downloading of MRIO databases **17**
- 5.1 EXIOBASE 3 download . . . 17
- 5.2 WIOD download . . . 19
- 5.3 OECD download . . . 22
- 5.4 Eora26 download . . . 23
- 5.5 EXIOBASE download (previous version 1 and 2) . . . 25
- 5.6 GLORIA download . . . 25

6 Handling MRIO data **27**
- 6.1 Exploring MRIOs with Pymrio . . . 27
- 6.2 Searching through the MRIO . . . 28
  - 6.2.1 Specific search methods: contains, match, fullmatch, . . . 29
  - 6.2.2 Search through all extensions . . . 31
  - 6.2.3 Generic search method for any dataframe index . . . 31
- 6.3 Using pymrio without a parser (small IO example) . . . 37
  - 6.3.1 Preperation . . . 37
  - 6.3.2 Include tables in the IOSystem object . . . 39
  - 6.3.3 Calculate the missing parts . . . 40
  - 6.3.4 Update to system for a new final demand . . . 40
- 6.4 Handling the WIOD EE MRIO database . . . 41
  - 6.4.1 Getting the database . . . 41

6.4.2 Parsing . . . 42
6.5 Working with the EXIOBASE EE MRIO database . . . 52
6.5.1 Getting EXIOBASE . . . 52
6.5.2 Parsing . . . 54
6.5.3 Exploring EXIOBASE . . . 55
6.5.4 Calculating the system and extension results . . . 57
6.5.5 Exploring the results . . . 57
6.5.6 Visualizing the data . . . 62
6.6 Parsing the Eora26 EE MRIO database . . . 63
6.6.1 Getting Eora26 . . . 63
6.6.2 Parse . . . 63
6.6.3 Explore . . . 64
6.7 Working with the OECD - ICIO database . . . 65
6.7.1 Parsing . . . 66
6.8 Loading, saving and exporting data . . . 67
6.8.1 Basic save and read . . . 67
6.8.2 Storage format . . . 68
6.8.3 Archiving MRIOs databases . . . 69
6.8.4 Storing or exporting a specific table or extension . . . 71
6.9 Adjusting, Renaming and Restructuring . . . 72
6.10 Using the aggregation functionality of pymrio . . . 75
6.10.1 Loading the test mrio . . . 75
6.10.2 Aggregation using a numerical concordance matrix . . . 75
6.10.3 Aggregation using a numerical vector . . . 77
6.10.4 Regional aggregation using the country converter coco . . . 77
6.10.5 Aggregation by renaming . . . 79
6.10.6 Aggregation to one total sector / region . . . 80
6.10.7 Pre- vs post-aggregation account calculations . . . 81
6.11 Analysing the source of stressors (flow matrix) . . . 82
6.11.1 Basic example . . . 82
6.11.2 Aggregation of source footprints . . . 90
6.12 Characterization of stressors . . . 94
6.12.1 Example . . . 94
6.12.2 Inspecting the used characterization table . . . 104
6.13 Advanced functionality - pandas groupby with pymrio satellite accounts . . . 106
6.13.1 WIOD material extension aggregation - stressor w/o compartment info . . . 106
6.13.2 Use with stressors including compartment information: . . . 110

7 Metadata and change recording **113**

8 Contributing **119**
8.1 Working on the documentation . . . 119
8.2 Changing the code base . . . 119
8.2.1 Running and extending the tests . . . 120
8.2.2 Debugging and logging . . . 120
8.3 Versioning . . . 121
8.4 Documentation and docstrings . . . 121
8.5 Open points . . . 121

9 Changelog **123**
9.1 v0.6dev . . . 123
9.1.1 Breaking changes . . . 123

9.1.2 Depracted . . . 123
9.1.3 Misceallaneous . . . 123
9.2 v0.5.2 - 20230815 . . . 123
9.2.1 New features . . . 123
9.2.2 Development . . . 124
9.3 v0.5.1 - 20230615 . . . 124
9.4 v0.5.0 - 20230615 . . . 124
9.4.1 Development . . . 124
9.4.2 Breaking changes . . . 124
9.4.3 New features . . . 124
9.4.4 Bugfixes . . . 124
9.5 v0.4.8 - 20221116 . . . 124
9.5.1 Bugfixes . . . 125
9.6 v0.4.7 - 20220428 . . . 125
9.7 v0.4.6 - 20211118 . . . 125
9.7.1 Breaking changes . . . 125
9.8 v0.4.5 (March 03, 2021) . . . 125
9.8.1 Bugfixes . . . 125
9.9 v0.4.4 (February 26, 2021) . . . 125
9.9.1 Bugfixes . . . 125
9.10 v0.4.3 (February 24, 2021) . . . 126
9.10.1 New features . . . 126
9.10.2 Bugfixes . . . 126
9.10.3 Development . . . 126
9.11 v0.4.2 (November 19, 2020) . . . 126
9.11.1 Bugfixes . . . 126
9.11.2 Development . . . 126
9.12 v0.4.1 (October 08, 2019) . . . 126
9.12.1 Bugfixes . . . 126
9.12.2 New features . . . 127
9.13 v0.4.0 (August 12, 2019) . . . 127
9.13.1 New features . . . 127
9.13.2 Bugfixes . . . 127
9.13.3 Backward incompatible changes . . . 127
9.14 v0.3.8 (November 06, 2018) . . . 127
9.15 v0.3.7 (October 10, 2018) . . . 127
9.15.1 New features . . . 127
9.15.2 Bugfixes . . . 128
9.15.3 Removed functionality . . . 128
9.15.4 Misc . . . 128
9.16 v0.3.6 (March 12, 2018) . . . 128
9.17 v0.3.5 (Jan 17, 2018) . . . 128
9.18 v0.3.4 (Jan 12, 2018) . . . 128
9.18.1 API breaking changes . . . 128
9.19 v0.3.3 (Jan 11, 2018) . . . 129
9.19.1 Dependencies . . . 129
9.19.2 API breaking changes . . . 129
9.20 v0.2.2 (May 27, 2016) . . . 129
9.20.1 Dependencies . . . 129
9.20.2 Misc . . . 129
9.21 v0.2.1 (Nov 17, 2014) . . . 130

9.21.1 Dependencies . . . 130
9.21.2 Misc . . . 130
9.22 v0.2.0 (Sept 11, 2014) . . . 130
9.22.1 API changes . . . 130
9.22.2 Misc . . . 130
9.23 v0.1.0 (June 20, 2014) . . . 130

10 API Reference **131**
10.1 Data input and output . . . 131
10.1.1 Test system . . . 131
10.1.2 Download MRIO databases . . . 132
10.1.3 Raw data . . . 134
10.1.4 Save data . . . 139
10.1.5 Load processed data . . . 141
10.1.6 Accessing . . . 142
10.2 Exploring and Search . . . 144
10.2.1 Structure . . . 144
10.2.2 Finding accounts . . . 147
10.3 Calculations . . . 153
10.3.1 Top level methods . . . 153
10.3.2 Low level matrix calculations . . . 155
10.4 Metadata and history recording . . . 159
10.4.1 pymrio.MRIOMetaData . . . 160
10.4.2 pymrio.MRIOMetaData.note . . . 161
10.4.3 pymrio.MRIOMetaData.history . . . 161
10.4.4 pymrio.MRIOMetaData.modification_history . . . 161
10.4.5 pymrio.MRIOMetaData.note_history . . . 162
10.4.6 pymrio.MRIOMetaData.file_io_history . . . 162
10.4.7 pymrio.MRIOMetaData.save . . . 162
10.5 Modifiying the IO System and its Extensions . . . 162
10.5.1 Aggregation . . . 162
10.5.2 Characterizing stressors . . . 164
10.5.3 Analysing the source of impacts . . . 166
10.5.4 Changing extensions . . . 166
10.5.5 Adjusting and Renaming . . . 168
10.5.6 Getting predefined Classifications of MRIOs . . . 169
10.6 Report . . . 170
10.6.1 pymrio.IOSystem.report_accounts . . . 171
10.7 Visualization . . . 171
10.7.1 pymrio.Extension.plot_account . . . 171
10.8 Utility . . . 172
10.8.1 pymrio.index_match . . . 173
10.8.2 pymrio.index_fullmatch . . . 173
10.8.3 pymrio.build_agg_matrix . . . 174
10.8.4 pymrio.convert_to_long . . . 175
10.9 Miscellaneous . . . 175
10.9.1 pymrio.IOSystem.reset_to_flows . . . 175
10.9.2 pymrio.IOSystem.reset_to_coefficients . . . 176
10.9.3 pymrio.IOSystem.copy . . . 176

11 Indices and tables **177**

Index 179

Contents:

CHAPTER

ONE

PYMRIO

Pymrio: Multi-Regional Input-Output Analysis in Python.

1.1 What is it

Pymrio is an open source tool for analysing global environmentally extended multi-regional input-output tables (EE MRIOs). Pymrio aims to provide a high-level abstraction layer for global EE MRIO databases in order to simplify common EE MRIO data tasks. Pymrio includes automatic download functions and parsers for available EE MRIO databases like EXIOBASE, WIOD and EORA26. It automatically checks parsed EE MRIOs for missing data necessary for calculating standard EE MRIO accounts (such as footprint, territorial, impacts embodied in trade) and calculates all missing tables. Various data report and visualization methods help to explore the dataset by comparing the different accounts across countries.

Further functions include:

- analysis methods to identify where certain impacts occur
- modifying region/sector classification
- restructuring extensions
- export to various formats
- visualization routines and
- automated report generation

1.2 Where to get it

The full source code is available on Github at: https://github.com/IndEcol/pymrio

Pymrio is registered at PyPI and on the Anaconda Cloud. Install it by:

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

or when using conda install it by

```
conda install -c conda-forge pymrio
```

or update to the latest version by

```
conda update -c conda-forge pymrio
```

The source-code of Pymrio available at the GitHub repo: https://github.com/IndEcol/pymrio

The master branch in that repo is supposed to be ready for use and might be ahead of the official releases. To install directly from the master branch use:

```
pip install git+https://github.com/IndEcol/pymrio@master
```

1.3 Quickstart

A small test mrio is included in the package.

To use it call

```
import pymrio
test_mrio = pymrio.load_test()
```

The test mrio consists of six regions and eight sectors:

```
print(test_mrio.get_sectors())
print(test_mrio.get_regions())
```

The test mrio includes tables flow tables and some satellite accounts. To show these:

```
test_mrio.Z
test_mrio.emissions.F
```

However, some tables necessary for calculating footprints (like test_mrio.A or test_mrio.emissions.S) are missing. pymrio automatically identifies which tables are missing and calculates them:

```
test_mrio.calc_all()
```

Now, all accounts are calculated, including footprints and emissions embodied in trade:

```
test_mrio.A
test_mrio.emissions.D_cba
test_mrio.emissions.D_exp
```

To visualize the accounts:

```
import matplotlib as plt
test_mrio.emissions.plot_account('emission_type1')
plt.show()
```

Everything can be saved with

```
test_mrio.save_all('some/folder')
```

See the documentation , tutorials and Stadler 2021 for further examples.

1.4 Tutorials

The documentation includes information about how to use pymrio for automatic downloading and parsing of the EE MRIOs EXIOBASE, WIOD, OECD and EORA26 as well as tutorials for the handling, aggregating and analysis of these databases.

1.5 Citation

If you use Pymrio in your research, citing the article describing the package (Stadler 2021) is very much appreciated.

For the full bibtex key see CITATION file.

1.6 Contributing

Want to contribute? Great! Please check CONTRIBUTING.rst if you want to help to improve Pymrio.

1.7 Communication, issues, bugs and enhancements

Please use the issue tracker for documenting bugs, proposing enhancements and all other communication related to pymrio.

You can follow me on twitter to get the latest news about all my open-source and research projects (and occasionally some random retweets).

1.7.1 Research notice

Please note that this repository is participating in a study into sustainability of open source projects. Data will be gathered about this repository for approximately the next 12 months, starting from June 2021.

Data collected will include number of contributors, number of PRs, time taken to close/merge these PRs, and issues closed.

For more information, please visit the informational page or download the participant information sheet.

CHAPTER

TWO

INSTALLATION

Pymrio is registered at PyPI and on the Anaconda Cloud. Install it by:

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

or when using conda install it by

```
conda install -c conda-forge pymrio
```

or update to the latest version by

```
conda update -c conda-forge pymrio
```

The source-code is available at the GitHub repo: https://github.com/IndEcol/pymrio

The master branch in that repo is supposed to be ready for use and might be ahead of the official releases. To install directly from the master branch use:

```
pip install git+https://github.com/IndEcol/pymrio@master
```

CHAPTER

THREE

TERMINOLOGY

So far, there is no consistent terminology for MRIO systems and parameters in the scientific community. For pymrio, the following variable names (= attributes of the IOSystem and Extensions) are used (the alias columns are other names and abbreviations often found in the literature):

variable name	formal name	alias variable names	alias formal names	description
Z	transaction matrix	T	flow matrix	transactions matrix, inter industry flows
A	A matrix			inter-industry coefficients (direct requirements matrix)
L	Leontief inverse	B		leontief inverse (total requirements matrix) (multi regional approach)
Y	final demand matrix	H, y		final demand matrix (sectors x region/countries and final demand categories)
x	gross output	q	industry output	total output, defined as column vector
F	factor production		extensions, stressors	Factors of productions: extensions plus value added block
F_Y	factor production final demand	yF, Fhh		Factor inputs/emissions of the final demand, eg the emissions from housholds
S	factor production coefficients	D	stressor coefficients	
S_Y	factor production coefficients final demand	DY, yD, Shh		Factor inputs/emissions coefficients of the final demand, eg the emissions from housholds
D_cba	consumption-based accounts	fp, con	footprints, consumption footprints	footprint of consumption, further specification with reg (per region) or cap (per capita) possible
D_pba	production-based accounts	terr	territorial accoutns	territorial or domestic accounts, further specification with reg (per region) or cap (per capita) possible
D_imp	import accounts	imp		import accounts, further specification with reg (per region) or cap (per capita) possible
D_exp	export accounts	exp		export accounts, further specification with reg (per region) or cap (per capita) possible
M	multipliers	m		total requirement factors
pxp	iosystem products x products	mxm		
ixi	iosystem industries x industries	nxn		

CHAPTER

FOUR

MATHEMATICAL BACKGROUND

This section gives a general overview about the mathematical background of Input-Output calculations. For a full detail account of this matter please see Miller and Blair 2009

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

4.1 Basic MRIO calculations

MRIO tables describe the global inter-industries flows within and across countries for k countries with a transaction matrix Z:

$$Z = \begin{pmatrix} Z_{1,1} & Z_{1,2} & \cdots & Z_{1,k} \\ Z_{2,1} & Z_{2,2} & \cdots & Z_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{k,1} & Z_{k,2} & \cdots & Z_{k,k} \end{pmatrix} \quad (4.1)$$

$\cdots$
$Z_{2,2}$
$Z_{2,k}\vdots$
$\ddots$
$Z_{k,2}$
$Z_{k,k}$

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

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

$\cdots$
$Y_{2,2}$
$Y_{2,k}\vdots$
$\ddots$
$Y_{k,2}$
$Y_{k,k}$

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:

$$x = Ze + Ye$$

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

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

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

Based on the linear economy assumption of the IO model and the classic Leontief demand-style modeling (see Leontief 1970), total industry output x can be calculated for any arbitrary vector of final demand y by multiplying with the total requirement matrix (Leontief matrix) L.

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

with I defined as the identity matrix with the size of A.

The global multi regional IO system can be extended with various factors of production $f_{h,i}$. These can represent among others value added, employment and social factors (h, with $h = 1, \ldots, r$) per country. The row vectors of factors can be summarised in a factor of production matrix F:

$$F = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,k} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ f_{r,1} & f_{r,2} & \cdots & f_{r,k} \end{pmatrix}$$

$\ldots$
$f_{2,2}$
$f_{2,k}\vdots$
$\ddots$
$f_{r,2}$
$f_{r,k}$

with the factor of production coefficients S given by

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

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

$$M = SL$$

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

$$D_{cba} = My$$

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)

$$D_{imp} = SLY_t$$

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

$$D_{exp} = S\widehat{LY_te}$$

with the hat indicating diagonalization of the resulting column-vector of the term underneath.

If the factor of production represent required environmental impacts, these can also occur during the final use phase. In that case F_Y describe the impacts associated with final demand (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 ($0 \ldots m$) plus the stressors occurring due to the final consumption for all final demand categories ($0 \ldots w$) of that region.

$$D^i_{pba} = \sum_{s=0}^{m} F^i_s + \sum_{c=0}^{w} F^i_{Yc}$$

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.

$$D^i_{cba} = \sum_{s=0}^{m} D^i_{cba,s} + \sum_{c=0}^{w} F^i_{Yc}$$

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

4.2 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.

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

$\cdots$
$b_{2,2}$
$b_{2,k}\vdots$
$\ddots$
$b_{w,2}$
$b_{w,k}$

$$\mathrm{B}_n = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{x,1} & b_{x,2} & \cdots & b_{x,n} \end{pmatrix}$$

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 $\otimes$ of B_k and B_n:

$$B = B_k \otimes B_n$$

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

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

and

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

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

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

and final demand impacts by

$$F_{Y,agg} = F_Y(B_k \otimes \mathrm{I})^{\mathrm{T}}$$

CHAPTER

FIVE

AUTOMATIC DOWNLOADING OF MRIO DATABASES

Pymrio includes functions to automatically download some of the publicly available global EE MRIO databases. This is currently implemented for EXIOBASE 3, OECD and WIOD.

The functions described here download the raw data files. Thus, they can also be used for post processing by other tools.

5.1 EXIOBASE 3 download

EXIOBASE 3 is licensed under the Creative Commons Attribution ShareAlike 4.0 International-license. Thus you can use EXIOBASE 3 for analyses as well as remix, tweak, and build uponit, even commercially, as long as you give credit to the EXIOBASE compilers and share your results/new databases under the same licence. The suggested citation for EXIOBASE 3 is Stadler et al 2018. You can find more information, links to documentation as well as concordance matrices on the EXIOBASE 3 Zenodo repository. The download function of pymrio also downloads the files from this repository.

To download, start with:

[1]:
```
import pymrio
```

and define a folder for storing the data:

[2]:
```
exio3_folder = "/tmp/mrios/autodownload/EXIO3"
```

With that we can start the download with (this might take a moment):

[3]:
```
exio_downloadlog = pymrio.download_exiobase3(
    storage_folder=exio3_folder, system="pxp", years=[2011, 2012]
)
```

The command above will download the latest EXIOBASE 3 tables in the product by product classification (system='pxp') for the years 2011 and 2012. Both parameters (system and years) are optional and when omitted the function will download all available files.

The function returns a log of the download data (which is stored in `download_log.json` in the download folder). You can inspect the meta data by:

[4]:
```
print(exio_downloadlog)
```

```
Description: Download log of EXIOBASE3
MRIO Name: EXIO3
```

(continues on next page)

(continued from previous page)

```
System: pxp
Version: 10.5281/zenodo.3583070
File: /tmp/mrios/autodownload/EXIO3/download_log.json
History:
20230323 09:32:39 - FILEIO -  Downloaded https://zenodo.org/record/5589597/
↪files/IOT_2012_pxp.zip to IOT_2012_pxp.zip
20230323 09:26:03 - FILEIO -  Downloaded https://zenodo.org/record/5589597/
↪files/IOT_2011_pxp.zip to IOT_2011_pxp.zip
20230323 09:19:45 - NOTE -  Download log created
20230323 09:19:45 - NOTE -  python_version: 3.9.16
20230323 09:19:45 - NOTE -  pymrio_version: 0.5.0dev
20230323 09:19:45 - NOTE -  os: Linux
20230323 09:19:45 - NOTE -  hostname: NTNU09417
20230323 09:19:45 - NOTE -  username: konstans
```

By default, the download_exiobase3 fetches the latest version of EXIOBASE3 available at the EXIOBASE 3 Zenodo repository. To download one of the previous versions specify the DOI with the doi parameter:

[5]:
```
prev_version_storage = "/tmp/mrios/autodownload/EXIO3_7"
exio_downlog_37 = pymrio.download_exiobase3(
    storage_folder=prev_version_storage,
    system="ixi",
    years=2004,
    doi="10.5281/zenodo.3583071",
)
```

[6]:
```
print(exio_downlog_37)
```

```
Description: Download log of EXIOBASE3
MRIO Name: EXIO3
System: ixi
Version: 10.5281/zenodo.3583071
File: /tmp/mrios/autodownload/EXIO3_7/download_log.json
History:
20230323 09:44:54 - FILEIO -  Downloaded https://zenodo.org/record/3583071/
↪files/IOT_2004_ixi.zip to IOT_2004_ixi.zip
20230323 09:43:49 - NOTE -  Download log created
20230323 09:43:49 - NOTE -  python_version: 3.9.16
20230323 09:43:49 - NOTE -  pymrio_version: 0.5.0dev
20230323 09:43:49 - NOTE -  os: Linux
20230323 09:43:49 - NOTE -  hostname: NTNU09417
20230323 09:43:49 - NOTE -  username: konstans
```

We also recommend to specifiy a specific DOI version even when using the latest version of EXIOBASE. In that way the used version is documented in the code and can be reproduced in case a newer EXIOBASE version becomes available.

5.2 WIOD download

DUE TO A RESTRUCTERING OF THE WIOD WEBPAGE THIS IS CURRENTLY BROKEN.

WIOD is licensed under the Creative Commons Attribution 4.0 International-license. Thus you can remix, tweak, and build upon WIOD, even commercially, as long as you give credit to WIOD. The WIOD web-page suggest to cite Timmer et al. 2015 when you use the database. You can find more information on the WIOD webpage.

The download function for WIOD currently processes the 2013 release version of WIOD.

To download, start with:

[7]:
```
import pymrio
```

Define a folder for storing the data

[8]:
```
wiod_folder = "/tmp/mrios/autodownload/WIOD2013"
```

And start the download with (this will take a couple of minutes):

[9]:
```
wiod_meta = pymrio.download_wiod2013(storage_folder=wiod_folder)
```

The function returns the meta data for the release (which is stored in `metadata.json` in the download folder). You can inspect the meta data by:

[10]:
```
print(wiod_meta)
```

```
Description: WIOD metadata file for pymrio
MRIO Name: WIOD
System: IxI
Version: data13
File: /tmp/mrios/autodownload/WIOD2013/metadata.json
History:
20210223 15:46:26 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/water/wat_may12.zip to wat_may12.zip
20210223 15:46:25 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/materials/mat_may12.zip to mat_may12.zip
20210223 15:46:25 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/land/lan_may12.zip to lan_may12.zip
20210223 15:46:24 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/AIR/AIR_may12.zip to AIR_may12.zip
20210223 15:46:24 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/CO2/CO2_may12.zip to CO2_may12.zip
20210223 15:46:23 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/EM/EM_may12.zip to EM_may12.zip
20210223 15:46:22 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/EU/EU_may12.zip to EU_may12.zip
20210223 15:46:21 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/SEA/WIOD_SEA_July14.xlsx to WIOD_SEA_July14.xlsx
20210223 15:46:20 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/update_sep12/wiot/wiot09_row_sep12.xlsx to wiot09_row_sep12.xlsx
20210223 15:46:15 - FILEIO -  Downloaded http://www.wiod.org/protected3/
```

(continues on next page)

(continued from previous page)

```
↪data13/wiot_analytic/wiot04_row_apr12.xlsx to wiot04_row_apr12.xlsx
 ... (more lines in history)
```

The WIOD database provide data for several years and satellite accounts. In the default case, all of them are downloaded. You can, however, specify years and satellite account.

You can specify the years as either int or string (2 or 4 digits):

[11]:
```
res_years = [97, 2004, "2005"]
```

The available satellite accounts for WIOD are listed in the `WIOD_CONFIG`. To get them import this dict by:

[12]:
```
from pymrio.tools.iodownloader import WIOD_CONFIG
```

[13]:
```
WIOD_CONFIG
```

[13]:
```
{'url_db_view': 'http://www.wiod.org/database/wiots13',
 'url_db_content': 'http://www.wiod.org/',
 'mrio_regex': 'protected.*?wiot\\d\\d.*?xlsx',
 'satellite_urls': ['http://www.wiod.org/protected3/data13/SEA/WIOD_SEA_
↪July14.xlsx',
  'http://www.wiod.org/protected3/data13/EU/EU_may12.zip',
  'http://www.wiod.org/protected3/data13/EM/EM_may12.zip',
  'http://www.wiod.org/protected3/data13/CO2/CO2_may12.zip',
  'http://www.wiod.org/protected3/data13/AIR/AIR_may12.zip',
  'http://www.wiod.org/protected3/data13/land/lan_may12.zip',
  'http://www.wiod.org/protected3/data13/materials/mat_may12.zip',
  'http://www.wiod.org/protected3/data13/water/wat_may12.zip']}
```

To restrict this list, you can either copy paste the urls or automatically select the accounts:

[14]:
```
sat_accounts = ["EU", "CO2"]
res_satellite = [
    sat
    for sat in WIOD_CONFIG["satellite_urls"]
    if any(acc in sat for acc in sat_accounts)
]
```

[15]:
```
res_satellite
```

[15]:
```
['http://www.wiod.org/protected3/data13/EU/EU_may12.zip',
 'http://www.wiod.org/protected3/data13/CO2/CO2_may12.zip']
```

[16]:
```
wiod_meta_res = pymrio.download_wiod2013(
    storage_folder="/tmp/foo_folder/WIOD2013_res",
    years=res_years,
    satellite_urls=res_satellite,
)
```

[17]:
```
print(wiod_meta_res)
```

```
Description: WIOD metadata file for pymrio
MRIO Name: WIOD
System: IxI
Version: data13
File: /tmp/foo_folder/WIOD2013_res/metadata.json
History:
20210218 15:29:34 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot01_row_apr12.xlsx to wiot01_row_apr12.xlsx
20210218 15:29:33 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot00_row_apr12.xlsx to wiot00_row_apr12.xlsx
20210218 15:29:32 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/CO2/CO2_may12.zip to CO2_may12.zip
20210218 15:29:31 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/EU/EU_may12.zip to EU_may12.zip
20210218 15:29:30 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot04_row_apr12.xlsx to wiot04_row_apr12.xlsx
20210218 15:29:27 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot97_row_apr12.xlsx to wiot97_row_apr12.xlsx
20210218 15:29:26 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot05_row_apr12.xlsx to wiot05_row_apr12.xlsx
```

Subsequent download will only catch files currently not present in the folder, e.g.:

[18]:
```
additional_years = [2000, 2001]
wiod_meta_res = pymrio.download_wiod2013(
    storage_folder="/tmp/foo_folder/WIOD2013_res",
    years=res_years + additional_years,
    satellite_urls=res_satellite,
)
```

only downloads the years given in `additional_years`, appending these downloads to the meta data file.

[19]:
```
print(wiod_meta_res)
```

```
Description: WIOD metadata file for pymrio
MRIO Name: WIOD
System: IxI
Version: data13
File: /tmp/foo_folder/WIOD2013_res/metadata.json
History:
20210218 15:29:34 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot01_row_apr12.xlsx to wiot01_row_apr12.xlsx
20210218 15:29:33 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot00_row_apr12.xlsx to wiot00_row_apr12.xlsx
20210218 15:29:32 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/CO2/CO2_may12.zip to CO2_may12.zip
20210218 15:29:31 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/EU/EU_may12.zip to EU_may12.zip
20210218 15:29:30 - FILEIO -  Downloaded http://www.wiod.org/protected3/
```

(continues on next page)

(continued from previous page)

```
↪data13/wiot_analytic/wiot04_row_apr12.xlsx to wiot04_row_apr12.xlsx
20210218 15:29:27 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot97_row_apr12.xlsx to wiot97_row_apr12.xlsx
20210218 15:29:26 - FILEIO -  Downloaded http://www.wiod.org/protected3/
↪data13/wiot_analytic/wiot05_row_apr12.xlsx to wiot05_row_apr12.xlsx
```

To catch all files, irrespective if present in the storage_folder or not pass `overwrite_existing=True`

5.3 OECD download

The OECD Inter-Country Input-Output tables (ICIO) are available on the OECD webpage. There is no specific license given for the these tables, but the webpage state that "Data can be downloaded for free" (per June 2023) and to cite this database "OECD (2021), OECD Inter-Country Input-Output Database, http://oe.cd/icio".

Currently OECD provides three versions 2016, 2018, and 2021, which are available through pymrio

To download the data, we first define the folder for storing the data (these will be created if they do not exist yet):

[20]:
```
oecd_folder_v2021 = "/tmp/mrios/autodownload/OECD_2021"
oecd_folder_v2018 = "/tmp/mrios/autodownload/OECD_2018"
```

Than we can start the download with

[21]:
```
log_2021 = pymrio.download_oecd(storage_folder=oecd_folder_v2021)
```

By default, the 2021 release of the OECD - ICIO tables are downloaded. To retrieve other versions, simply pass "version='v<version date>'", for example to get the 2018 release, pass "version='v2018'" .

As for WIOD, specific years can be specified by passing a list of years:

[22]:
```
log_2018 = pymrio.download_oecd(
    storage_folder=oecd_folder_v2018, version="v2016", years=[2003, 2008]
)
```

However, the 2021 release of OECD is only available in 5-year bundles starting from 1995 to 2018 1995-1999, 2000-2004, 2005-2009, 2010-2014, 2015-2018

Therefore whenever 2021 release is used, it is recommended to pass years as a list of the wanted bundles in string forms:

[ ]:
```
log_2021 = pymrio.download_oecd(
    storage_folder=oecd_folder_v2021, years=["1995-1999", "2015-2018"]
)
```

Otherwise the corresponding bundles for the entered years would be downloaded, for example if year 2003 is requested, the bundle 2000-2004 would be downloaded

[ ]:
```
log_2021 = pymrio.download_oecd(storage_folder=oecd_folder_v2021, years=[2003,
↪ 2012])
```

The bundles 2000-2004 and 2010-2014 would be downloaded

The function returns a log for the download progress and MRIO info:

[23]:
```
print(log_2021)
```

```
Description: OECD-ICIO download
MRIO Name: OECD-ICIO
System: IxI
Version: v2021
File: \tmp\mrios\autodownload\OECD_2021\download_log.json
History:
20230706 15:35:10 - FILEIO -  Downloaded https://stats.oecd.org/wbos/
↪fileview2.aspx?IDFile=2c2f499f-5703-4034-9457-2f7518e8f2fc to ICIO2021_2010-
↪2014.zip
20230706 15:31:47 - FILEIO -  Downloaded https://stats.oecd.org/wbos/
↪fileview2.aspx?IDFile=8adf89dd-18b4-40fe-bc7f-c822052eb961 to ICIO2021_2000-
↪2004.zip
20230706 15:29:43 - NOTE -  Download log created
20230706 15:29:43 - NOTE -  python_version: 3.11.2
20230706 15:29:43 - NOTE -  pymrio_version: 0.5.1
20230706 15:29:43 - NOTE -  os: Windows
20230706 15:29:43 - NOTE -  hostname: NTNU09702
20230706 15:29:43 - NOTE -  username: hazimh
```

5.4 Eora26 download

Eora26 provides a simplified, symmetric version of the full Eora database.

Downloading the Eora data requires registration through the Eora website (worldmrio) . Currently (August 2023), open Eora26 data are only offered for the years 1990 - 2016.

Therefore, you have to have the Eora account email and password in advance to use the downloader.

Setup the download with

[14]:
```
import pymrio

eora_folder = "/tmp/mrios/eora26"
```

Start the download with (passing the email and password of the eora account) If you pass invalid email or password, you get a prompt about that and asked to enter them again, otherwise the function will download the data (this can take some minutes)

This will download all available years 1990 - 2016

[ ]:
```
eora_log = pymrio.download_eora26(
    storage_folder=eora_folder,
    email="<Your Eora account email>",
    password="<Your Eora account password>",
)
```

```
Eora account with this email was not found

            Please try again or register a new user at the following website:

            https://worldmrio.com/login.jsp
The password for this email account is incorrect

            Please try again
```

[16]:
```
print(eora_log)
```

```
Description: Eora metadata file for pymrio
MRIO Name: Eora
System: ixi
Version: v199.82
File: /tmp/mrios/eora26/metadata.json
History:
20180111 10:26:35 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_1995_bp.zip to Eora26_1995_bp.zip
20180111 10:26:26 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_1996_bp.zip to Eora26_1996_bp.zip
20180111 10:26:17 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_1997_bp.zip to Eora26_1997_bp.zip
20180111 10:25:57 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_1998_bp.zip to Eora26_1998_bp.zip
20180111 10:25:47 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_1999_bp.zip to Eora26_1999_bp.zip
20180111 10:25:37 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_2000_bp.zip to Eora26_2000_bp.zip
20180111 10:25:21 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_2001_bp.zip to Eora26_2001_bp.zip
20180111 10:25:08 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_2002_bp.zip to Eora26_2002_bp.zip
20180111 10:24:58 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_2003_bp.zip to Eora26_2003_bp.zip
20180111 10:24:46 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/
↪Phase199/Loop082/simplified/Eora26_2004_bp.zip to Eora26_2004_bp.zip
 ... (more lines in history)
```

As in the case of the WIOD downloader, you can restrict the

1) years to download by passing `years=[list of int or str - 4 digits]`
2) force the overwriting of existing files by passing `overwrite_existing=True`

Satellite accounts, however, can not be restricted since they are included in one file.

The tables are in basic prices as it is the only price system available to download for Eora26.

5.5 EXIOBASE download (previous version 1 and 2)

Previous EXIOBASE version requires registration prior to download and therefore an automatic download has not been implemented. For further information check the download instruction at the *EXIOBASE example notebook*.

5.6 GLORIA download

The Global Resource Input Output Assessment (GLORIA) database are available to download through a dropbox folder GLORIA database. There is no specific licence given for the this database.

Currently (as per April 2023), there are four available versions 53, 54, 55, and 57 (Release 056 was only distributed to a limited number of users for feedback).

The download function can (as per April 2023) download all four versions

To download, start with:

```
[ ]: import pymrio
```

Define a folder for storing the data

```
[ ]: gloria_folder = "/tmp/mrios/autodownload/GLORIA2014"
```

And start the download with (this will take a couple of minutes):

```
[ ]: gloria_log_2014 = pymrio.download_gloria(storage_folder=gloria_folder)
```

The function returns the download log data for the release (which is stored in `download_log.json` in the download folder). You can inspect the log data by:

```
[ ]: print(gloria_log_2014)
```

```
Description: Download log of Gloria
MRIO Name: GLORIA
System: IxI
Version: 57
File: /tmp/mrios/autodownload/GLORIA2014/download_log.json
History:
20230418 07:51:15 - NOTE -  Download log created
20230418 07:51:15 - NOTE -  python_version: 3.11.3
20230418 07:51:15 - NOTE -  pymrio_version: 0.5.0dev
20230418 07:51:15 - NOTE -  os: Linux
20230418 07:51:15 - NOTE -  hostname: instance-exio-svr
20230418 07:51:15 - NOTE -  username: hazimh
20230418 07:50:38 - FILEIO -  Downloaded https://dl.dropboxusercontent.com/sh/
↪o4fxq94n7grvdbk/AACposoILlF8OETb20yh4st7a/latest_release/057/GLORIA_MRIO_
↪Loop057_part_I_MRIOdatabase/GLORIA_MRIOs_57_2014.zip?dl=0 to GLORIA_MRIOs_
↪57_2014.zip
20230418 07:46:30 - NOTE -  Download log created
20230418 07:46:30 - NOTE -  python_version: 3.11.3
```

(continues on next page)

(continued from previous page)

```
20230418 07:46:30 - NOTE -  pymrio_version: 0.5.0dev
 ... (more lines in history)
```

By default the function downloads all years for the final release (57 as per April 2023), but the year and version can be specified by passing them to the function

[ ]:
```
gloria_log_v053_2012 = pymrio.download_gloria(gloria_folder, year=2012,␣
↪version=53)
```

CHAPTER

SIX

HANDLING MRIO DATA

6.1 Exploring MRIOs with Pymrio

The first step when working with a new MRIO data set is to familiarize yourself with the data. This notebook shows how to use the `pymrio` package to explore the data. We use the test data set that is included in the `pymrio` package. This is a completely artificial, very small MRIO. It is not meant to be realistic, but it is useful for developing, testing and learning.

First we import the required packages:

[1]:
```
import pymrio
```

We can now load the test data set with the `load_test` function. We can call the MRIO whatever we want, here we use mrio.

[2]:
```
mrio = pymrio.load_test()
```

We can get some first information about the MRIO by printing it.

[3]:
```
print(mrio)
```

```
IO System with parameters: Z, Y, unit, population, meta, factor_inputs,␣
↪emissions
```

This tells us what the MRIO data we just loaded contains. We find a Z and Y matrix, some unit information and two satellite accounts, factor_inputs and emissions.

To get more specific data we can ask pymrio for regions, sectors, products, etc.

[4]:
```
mrio.name
```

[4]:
```
'testmrio'
```

[5]:
```
mrio.get_regions()
```

[5]:
```
Index(['reg1', 'reg2', 'reg3', 'reg4', 'reg5', 'reg6'], dtype='object', name=
↪'region')
```

[6]:
```
mrio.get_sectors()
```

[6]:
```
Index(['food', 'mining', 'manufactoring', 'electricity', 'construction',
       'trade', 'transport', 'other'],
      dtype='object', name='sector')
```

```
[7]: mrio.get_Y_categories()
```

```
[7]: Index(['Final consumption expenditure by households',
       'Final consumption expenditure by non-profit organisations serving␣
↪households (NPISH)',
       'Final consumption expenditure by government',
       'Gross fixed capital formation', 'Changes in inventories',
       'Changes in valuables', 'Export'],
      dtype='object', name='category')
```

The same methods can be used to explore one of the satellite accounts.

```
[8]: print(mrio.emissions)
```

```
Extension Emissions with parameters: name, F, F_Y, unit
```

```
[9]: mrio.emissions.name
```

```
[9]: 'Emissions'
```

```
[10]: mrio.emissions.get_regions()
```

```
[10]: Index(['reg1', 'reg2', 'reg3', 'reg4', 'reg5', 'reg6'], dtype='object', name=
↪'region')
```

The satellite accounts also have a special method to get index (rows) of the acccounts.

```
[11]: mrio.emissions.get_rows()
```

```
[11]: MultiIndex([('emission_type1',   'air'),
            ('emission_type2', 'water')],
           names=['stressor', 'compartment'])
```

6.2 Searching through the MRIO

Several methods are available to search through the whole MRIO. These generally accept regular expressions as search terms.

The most general method is 'find'. This can be used for a quick overview where a specific term appears in the MRIO.

```
[12]: mrio.find('air')
```

```
[12]: {'emissions_index': MultiIndex([('emission_type1', 'air')],
            names=['stressor', 'compartment'])}
```

```
[13]: mrio.find("trade")
```

```
[13]: {'index': MultiIndex([('reg1', 'trade'),
             ('reg2', 'trade'),
             ('reg3', 'trade'),
             ('reg4', 'trade'),
```

(continues on next page)

(continued from previous page)

```
             ('reg5', 'trade'),
             ('reg6', 'trade')],
            names=['region', 'sector']),
 'sectors': Index(['trade'], dtype='object', name='sector')}
```

Not that 'find' (and all other search methods) a case sensitive. Do make a case insensitive search, add the regular expression flag `(?i)` to the search term.

[14]:
```
mrio.find('value')
```

[14]:
```
{}
```

[15]:
```
mrio.find('(?i)value')
```

[15]:
```
{'factor_inputs_index': Index(['Value Added'], dtype='object', name='inputtype
↪')}
```

6.2.1 Specific search methods: contains, match, fullmatch,

The MRIO class also contains a set of specific regular expresion search methods, mirroring the 'contains', 'match' and 'fullmatch' methods of the pandas DataFrame str column type. See the pandas documentation for details, in short:

-. 'contains' looks for a match anywhere in the string -. 'match' looks for a match at the beginning of the string -. 'fullmatch' looks for a match of the whole string

These methods are available for all index columns of the MRIO and have a similar signature:

1. As for 'find_all', the search term is case sensitive. To make it case insensitive, add the regular expression flag `(?i)` to the search term.
2. The search term can be passed to the keyword argument 'find_all' or as the first positional argument to search in all index levels.
3. Alternativels, the search term can be passed to the keyword argument with the level name to search only in that index level.

The following examples show how to use these methods.

[16]:
```
mrio.contains(find_all = 'ad')
mrio.contains('ad')
```

[16]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

[17]:
```
mrio.match('ad')
```

[17]:
```
MultiIndex([], names=['region', 'sector'])
```

[18]:
```
mrio.match('trad')
```

[18]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

[19]:
```
mrio.fullmatch('trad')
```

[19]:
```
MultiIndex([], names=['region', 'sector'])
```

[20]:
```
mrio.fullmatch('trade')
```

[20]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

[21]:
```
mrio.fullmatch('(?i).*AD.*')
```

[21]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

For the rest of the notebook, we will do the examples with the 'contains' method, but the same applies to the other methods.

To search only at one specific level, pass the search term to the keyword argument with the level name.

[22]:
```
mrio.contains(region='trade')
```

[22]:
```
MultiIndex([], names=['region', 'sector'])
```

[23]:
```
mrio.contains(sector='trade')
```

[23]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

And of course, the method are also available for the satellite accounts.

```
[24]: mrio.emissions.contains(compartment='air')
```

```
[24]: MultiIndex([('emission_type1', 'air')],
                 names=['stressor', 'compartment'])
```

Passing a non-existing level to the keyword argument is silently ignored.

```
[25]: mrio.factor_inputs.contains(compartment='trade')
```

```
[25]: Index([], dtype='object')
```

This allows to search for terms that are only in some index levels. Locially, this is an 'or' search.

```
[26]: mrio.factor_inputs.contains(compartment='air', inputtype="Value")
```

```
[26]: Index(['Value Added'], dtype='object', name='inputtype')
```

But note, that if both levels exist, both must match (so it becomes a logical 'and').

```
[27]: mrio.emissions.contains(stressor='emission', compartment='air')
```

```
[27]: MultiIndex([('emission_type1', 'air')],
                 names=['stressor', 'compartment'])
```

6.2.2 Search through all extensions

All three search methods are also available to loop through all extensions of the MRIO.

```
[28]: mrio.extension_contains(stressor='emission', compartment='air')
```

```
[28]: {'Factor Inputs': Index([], dtype='object'),
 'Emissions': MultiIndex([('emission_type1', 'air')],
                  names=['stressor', 'compartment'])}
```

If only a subset of extensions should be searched, pass the extension names to the keyword argument 'extensions'.

6.2.3 Generic search method for any dataframe index

Internally, the class methods 'contains', 'match' and 'fullmatch' all the 'index_contains', 'index_match' and 'index_fullmatch' methods of ioutil module. This function can be used to search through index of any pandas DataFrame.

```
[29]: df = mrio.Y
```

Depending if a dataframe or an index is passed, the return is either the dataframe or the index.

```
[30]: pymrio.index_contains(df, 'trade')
```

```
[30]: region                                                reg1  \
      category      Final consumption expenditure by households
      region sector
      reg1   trade                                   769535.93000
```

(continues on next page)

(continued from previous page)

```
reg2   trade                                  5678.26740
reg3   trade                                  2753.86080
reg4   trade                                   373.28393
reg5   trade                                  4287.40670
reg6   trade                                  4772.75750

region                                                                    ␣
↪                      \
category      Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
region sector
reg1   trade                                          16.638920
reg2   trade                                           0.075424
reg3   trade                                           0.111540
reg4   trade                                           0.009382
reg5   trade                                           0.038941
reg6   trade                                           0.113112

region                                                    \
category      Final consumption expenditure by government
region sector
reg1   trade                                2.408807e+07
reg2   trade                                2.312962e+02
reg3   trade                                1.956911e+00
reg4   trade                                3.585011e-01
reg5   trade                                7.014679e+00
reg6   trade                                2.321101e+01

region                                                              \
category      Gross fixed capital formation Changes in inventories
region sector
reg1   trade                   6.727345e+07            1230.218200
reg2   trade                   6.339521e+02              35.607157
reg3   trade                   3.598675e+02              23.391120
reg4   trade                   2.514957e-02               0.002016
reg5   trade                   1.955479e+02               6.675656
reg6   trade                   2.417571e+02              16.267049

region                                    \
category      Changes in valuables Export
region sector
reg1   trade            216.211080      0
reg2   trade              3.192694      0
reg3   trade              0.000455      0
reg4   trade              0.000144      0
reg5   trade              0.524015      0
reg6   trade              1.488818      0

region                                             reg2  \
```

(continues on next page)

(continued from previous page)

```
category      Final consumption expenditure by households
region sector
reg1   trade                                    8063.52380
reg2   trade                                  385664.01000
reg3   trade                                    2072.24890
reg4   trade                                     192.21539
reg5   trade                                    3633.68750
reg6   trade                                    2031.49640

region                                                                 ␣
↪                      \
category      Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
region sector
reg1   trade                                       12.738233
reg2   trade                                      178.501140
reg3   trade                                        0.044811
reg4   trade                                        0.019666
reg5   trade                                        2.536312
reg6   trade                                        1.864492

region                                                    ...  \
category      Final consumption expenditure by government  ...
region sector                                               ...
reg1   trade                                     163.205380  ...
reg2   trade                                    8160.881200  ...
reg3   trade                                       1.242613  ...
reg4   trade                                       0.537107  ...
reg5   trade                                      50.624916  ...
reg6   trade                                      18.787893  ...

region                           reg5                                \
category      Changes in inventories Changes in valuables Export
region sector
reg1   trade              204.383170         1.684295e+00      0
reg2   trade              358.249320         2.271962e+01      0
reg3   trade              309.984270         6.283278e+00      0
reg4   trade               73.859706         7.199126e-02      0
reg5   trade             9177.081800         1.330591e+06      0
reg6   trade               91.040319         2.122217e+00      0

region                                              reg6  \
category      Final consumption expenditure by households
region sector
reg1   trade                                   49782414.00
reg2   trade                                   26592464.00
reg3   trade                                     114505.89
reg4   trade                                   40152651.00
reg5   trade                                   60992225.00
```

(continues on next page)

(continued from previous page)

```
reg6   trade                                    851864.04

region                                                                    ␣
↪                        \
category      Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
region sector
reg1   trade                                          0.224933
reg2   trade                                          0.139745
reg3   trade                                          0.630098
reg4   trade                                          0.255523
reg5   trade                                          0.823823
reg6   trade                                         23.371306

region                                                   \
category      Final consumption expenditure by government
region sector
reg1   trade                                    14.445660
reg2   trade                                    10.623962
reg3   trade                                    37.095549
reg4   trade                                    17.253634
reg5   trade                                    34.208026
reg6   trade                                  1966.030900

region                                                            \
category      Gross fixed capital formation Changes in inventories
region sector
reg1   trade                    16739029.00              12.145465
reg2   trade                    11775351.00              20.572534
reg3   trade                    31317361.00             212.707510
reg4   trade                    14011134.00               4.052444
reg5   trade                    27870911.00              85.191511
reg6   trade                      131182.13            1549.410400

region
category      Changes in valuables Export
region sector
reg1   trade              0.013888      0
reg2   trade              0.005433      0
reg3   trade              0.014929      0
reg4   trade              0.001935      0
reg5   trade              0.008929      0
reg6   trade              0.266033      0

[6 rows x 42 columns]
```

[31]:
```
pymrio.index_contains(df.index, 'trade')
```

[31]:
```
MultiIndex([('reg1', 'trade'),
            ('reg2', 'trade'),
```

(continues on next page)

(continued from previous page)

```
            ('reg3', 'trade'),
            ('reg4', 'trade'),
            ('reg5', 'trade'),
            ('reg6', 'trade')],
           names=['region', 'sector'])
```

[32]:
```
pymrio.index_fullmatch(df, region='reg[2,4]', sector='m.*')
```

[32]:
```
region                                                    reg1  \
category             Final consumption expenditure by households
region sector
reg2   mining                                       1.653997e+02
       manufactoring                                9.928459e+07
reg4   mining                                       1.072728e+02
       manufactoring                                4.086352e+07

region                                                                 ␣
↪                              \
category             Final consumption expenditure by non-profit␣
↪organisations serving households (NPISH)
region sector
reg2   mining                                             1.817989e-05
       manufactoring                                      4.187143e+00
reg4   mining                                             9.421644e-09
       manufactoring                                      1.170611e+00

region                                                           \
category             Final consumption expenditure by government
region sector
reg2   mining                                           0.334824
       manufactoring                                 1373.370200
reg4   mining                                           0.209851
       manufactoring                                  377.322810

region                                                                  \
category             Gross fixed capital formation Changes in inventories
region sector
reg2   mining                         3.283238e+01           2.910648e+01
       manufactoring                  4.237878e+07           4.415752e+03
reg4   mining                         1.055704e+00           2.697312e+01
       manufactoring                  3.032655e+07           2.263532e+06

region                                            \
category             Changes in valuables Export
region sector
reg2   mining                3.970468e-05      0
       manufactoring         1.637658e+02      0
reg4   mining                3.112643e-09      0
       manufactoring         4.696135e+01      0
```

(continues on next page)

(continued from previous page)

```
region                                                        reg2  \
category             Final consumption expenditure by households
region sector
reg2   mining                                       1.091126e+03
       manufactoring                                3.210316e+05
reg4   mining                                       2.940734e+02
       manufactoring                                1.510445e+07

region                                                                ␣
↪                                \
category             Final consumption expenditure by non-profit␣
↪organisations serving households (NPISH)
region sector
reg2   mining                                                2.751312
       manufactoring                                       125.729110
reg4   mining                                                0.000001
       manufactoring                                         7.598440

region                                                          ...  \
category             Final consumption expenditure by government  ...
region sector                                                    ...
reg2   mining                                       1.544777e+01  ...
       manufactoring                                1.603833e+07  ...
reg4   mining                                       1.061796e-01  ...
       manufactoring                                1.285316e+02  ...

region                                  reg5                                \
category             Changes in inventories Changes in valuables Export
region sector
reg2   mining                      0.040217             0.000420      0
       manufactoring             951.809210            21.641280      0
reg4   mining                     45.388457             0.000015      0
       manufactoring            1004.074200             1.476200      0

region                                                        reg6  \
category             Final consumption expenditure by households
region sector
reg2   mining                                       6.127299e-01
       manufactoring                                1.074192e+07
reg4   mining                                       7.705800e+04
       manufactoring                                2.190604e+07

region                                                                ␣
↪                                \
category             Final consumption expenditure by non-profit␣
↪organisations serving households (NPISH)
region sector
reg2   mining                                                0.008212
       manufactoring                                        62.832488
```

(continues on next page)

(continued from previous page)

```
reg4   mining                                                0.348367
       manufactoring                                        94.282910

region                                                                \
category            Final consumption expenditure by government
region sector
reg2   mining                                              0.013061
       manufactoring                                    6363.192800
reg4   mining                                              2.326858
       manufactoring                                    6712.972800

region                                                                 \
category            Gross fixed capital formation Changes in inventories
region sector
reg2   mining                        1.167170e+01             166.539580
       manufactoring                 1.170497e+07            1060.321500
reg4   mining                        9.917552e-02              68.266815
       manufactoring                 3.167496e+07             825.256940

region
category            Changes in valuables Export
region sector
reg2   mining                    0.000002      0
       manufactoring             0.145903      0
reg4   mining                    0.000789      0
       manufactoring             0.036510      0

[4 rows x 42 columns]
```

6.3 Using pymrio without a parser (small IO example)

Pymrio provides parsing function to load existing MRIO databases. However, it is also possible to assign data directly to the attributes of an IOSystem instance.

This tutorial exemplify this functionality. The tables used here are taken from *Miller and Blair (2009)*: Miller, Ronald E, and Peter D Blair. Input-Output Analysis: Foundations and Extensions. Cambridge (England); New York: Cambridge University Press, 2009. ISBN: 978-0-521-51713-3

6.3.1 Preperation

Import pymrio

First import the pymrio module and other packages needed:

[1]:
```
import pymrio

import pandas as pd
import numpy as np
```

Get external IO table

For this example we use the IO table given in *Miller and Blair (2009)*: Table 2.3 (on page 22 in the 2009 edition).

This table contains an interindustry trade flow matrix, final demand columns for household demand and exports and a value added row. The latter we consider as an extensions (factor inputs). To assign these values to the IOSystem attributes, the tables must be pandas DataFrames with multiindex for columns and index.

First we set up the Z matrix by defining the index of rows and columns. The example IO tables contains only domestic tables, but since pymrio was designed with multi regions IO tables in mind, also a region index is needed.

[2]:
```
_sectors = ["sector1", "sector2"]
_regions = ["reg1"]
_Z_multiindex = pd.MultiIndex.from_product(
    [_regions, _sectors], names=["region", "sector"]
)
```

Next we setup the total Z matrix. Here we just put in the name the values manually. However, pandas provides several possibility to ease the data input.

[3]:
```
Z = pd.DataFrame(
    data=np.array([[150, 500], [200, 100]]), index=_Z_multiindex, columns=_Z_
↪multiindex
)
```

[4]:
```
Z
```

[4]:
```
region            reg1
sector         sector1 sector2
region sector
reg1   sector1     150     500
       sector2     200     100
```

Final demand is treated in the same way:

[5]:
```
_categories = ["final demand"]
_fd_multiindex = pd.MultiIndex.from_product(
    [_regions, _categories], names=["region", "category"]
)
```

[6]:
```
Y = pd.DataFrame(
    data=np.array([[350], [1700]]), index=_Z_multiindex, columns=_fd_
↪multiindex
)
```

[7]:
```
Y
```

[7]:
```
region                 reg1
category       final demand
region sector
```

(continues on next page)

(continued from previous page)

```
reg1   sector1          350
       sector2         1700
```

Factor inputs are given as 'Payment sectors' in the table:

[8]:
```
F = pd.DataFrame(
    data=np.array([[650, 1400]]), index=["Payments_sectors"], columns=_Z_
↪multiindex
)
```

[9]:
```
F
```

[9]:
```
region              reg1
sector           sector1 sector2
Payments_sectors     650    1400
```

6.3.2 Include tables in the IOSystem object

In the next step, an empty instance of an IOSYstem has to be set up.

[10]:
```
io = pymrio.IOSystem()
```

Now we can add the tables to the IOSystem instance:

[11]:
```
io.Z = Z
io.Y = Y
```

Extension are defined as objects within the IOSystem. The Extension instance can be instanced independently (the parameter 'name' is required):

[12]:
```
factor_input = pymrio.Extension(name="Factor Input", F=F)
```

[13]:
```
io.factor_input = factor_input
```

For consistency and plotting we can add a DataFrame containg the units per row:

[14]:
```
io.factor_input.unit = pd.DataFrame(data=["USD"], index=F.index, columns=[
↪"unit"])
```

We can check whats in the system:

[15]:
```
str(io)
```

[15]:
```
'IO System with parameters: Z, Y, meta, factor_input'
```

At this point we have everything to calculate the full IO system.

6.3.3 Calculate the missing parts

[16]:
```
io.calc_all()
```

[16]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e16bde8b0>
```

This gives, among others, the A and L matrix which we can compare with the tables given in *Miller and Blair (2009)* (Table 2.4 and L given on the next page afterwards):

[17]:
```
io.A
```

[17]:
```
region            reg1
sector         sector1 sector2
region sector
reg1   sector1    0.15    0.25
       sector2    0.20    0.05
```

[18]:
```
io.L
```

[18]:
```
region              reg1
sector           sector1   sector2
region sector
reg1   sector1  1.254125  0.330033
       sector2  0.264026  1.122112
```

6.3.4 Update to system for a new final demand

The example in *Miller and Blair (2009)* goes on with using the L matrix to calculate the new industry output x for a changing finald demand Y. This step can easly reproduced with the pymrio module.

To do so we first have to set up the new final demand:

[19]:
```
Ynew = Y.copy()
Ynew[("reg1", "final_demand")] = np.array([[600], [1500]])
```

We copy the original IOSystem:

[20]:
```
io_new_fd = io.copy()
```

To calculate for the new final demand we have to remove everything from the system except for the coefficients (A,L,S,M)

[21]:
```
io_new_fd.reset_all_to_coefficients()
```

[21]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e16b9c280>
```

Now we can assign the new final demand and recalculate the system:

[22]:
```
io_new_fd.Y = Ynew
```

[23]:
```
io_new_fd.calc_all()
```

[23]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e16b9c280>
```

The new x equalls the xnew values given in *Miller and Blair (2009)* at formula 2.13:

[24]:
```
io_new_fd.x
```

[24]:
```
                     indout
region sector
reg1   sector1  2247.524752
       sector2  3841.584158
```

As for all IO System, we can have a look at the modification history:

[25]:
```
io_new_fd.meta
```

[25]:
```
Description: Metadata for pymrio
MRIO Name: IO_copy
System: None
Version: None
File: None
History:
20210224 10:42:06 - MODIFICATION -  Calculating accounts for extension factor_
↪input
20210224 10:42:06 - MODIFICATION -  Flow matrix Z calculated
20210224 10:42:06 - MODIFICATION -  Industry Output x calculated
20210224 10:42:06 - MODIFICATION -  Reset full system to coefficients
20210224 10:42:06 - NOTE -  IOSystem copy IO_copy based on IO
20210224 10:42:06 - MODIFICATION -  Calculating accounts for extension factor_
↪input
20210224 10:42:06 - MODIFICATION -  Leontief matrix L calculated
20210224 10:42:06 - MODIFICATION -  Coefficient matrix A calculated
20210224 10:42:06 - MODIFICATION -  Industry output x calculated
```

6.4 Handling the WIOD EE MRIO database

6.4.1 Getting the database

The WIOD database is available at http://www.wiod.org. You can download these files with the pymrio automatic downloader as described at *WIOD download*.

In the most simple case you get the full WIOD database with:

[1]:
```
import pymrio
```

[2]:
```
wiod_storage = "/tmp/mrios/WIOD2013"
```

[3]:
```
wiod_meta = pymrio.download_wiod2013(storage_folder=wiod_storage)
```

This download the whole 2013 release of WIOD including all extensions.

The extension (satellite accounts) are provided as zip files. You can use them directly in pymrio (without extracting them). If you want to have them extracted, create a folder with the name of each extension (without the ending ".zip") and extract the zip file there.

6.4.2 Parsing

Parsing a single year

A single year of the WIOD database can be parse by:

[4]:
```
wiod2007 = pymrio.parse_wiod(year=2007, path=wiod_storage)
```

Which loads the specific year and extension data:

[5]:
```
wiod2007.Z.head()
```

[5]:
```
region                 AUS                                                  ␣
↪ \
sector                 AtB            C         15t16       17t18          19
region sector
AUS    AtB     3784.749470    33.520510  13821.807920  474.033810  136.300060
       C         26.253436  6671.832980    324.993193   20.785395    5.976473
       15t16    929.958296    81.490230   6201.543062   60.054879   17.267720
       17t18     31.971488    33.970751     95.871008  295.911795   85.084218
       19         8.244949     8.760528     24.723640   76.311042   21.941895

region                                                                    \
sector                 20       21t22           23          24         25
region sector
AUS    AtB     810.877200  234.948790     0.000000  185.784570  81.938000
       C        26.981388  105.029472  6659.692127  352.992634  34.737431
       15t16    13.588882   45.246115    24.007404  754.675350  50.919526
       17t18    16.340238   43.536115     9.525953   39.101829  38.656918
       19        4.213893   11.227285     2.456595   10.083753   9.969017

region         ...        RoW                                          \
sector         ...         63        64         J        70      71t74
region sector  ...
AUS    AtB     ...  10.481874  0.000808  0.031735  0.023280   5.861476
       C       ...   0.220334  0.028363  0.004541  0.081185   5.442078
       15t16   ...   0.936095  1.495263  3.829824  2.434498  13.989495
       17t18   ...   0.560207  0.367082  0.651173  0.779275   3.215393
       19      ...   0.050365  0.033002  0.058544  0.070061   0.289079

region
sector                 L           M          N          O         P
region sector
AUS    AtB     1.123848   25.333019   4.291562   4.874767  0.000791
       C       0.292077    0.232191   0.310890   0.443509  0.001000
       15t16   8.138864  133.900410  48.045408  62.537153  0.001805
       17t18   5.515491    0.854378   2.059234   3.027687  0.001752
```

(continues on next page)

(continued from previous page)

```
       19      0.495869    0.076813   0.185135   0.272204  0.000157

[5 rows x 1435 columns]
```

[6]:
```
wiod2007.AIR.F
```

[6]:
```
region             AUS                                                \
sector             AtB             C          15t16         17t18
stressor
CO2       6.367691e+03  2.365858e+04    3126.525484    409.176104
CH4       3.221551e+06  1.368899e+06    1213.871992     43.759118
N2O       6.460006e+04  1.209250e+02     519.404359     11.081322
NOX       1.755811e+05  1.722916e+05   68672.002050   4040.886651
SOX       1.658225e+04  4.307541e+04   46636.439902   1010.280269
CO        1.512935e+06  8.801313e+05  247496.408649  20642.389652
NMVOC     3.999910e+05  2.800153e+05  148660.535247   6567.410709
NH3       5.199660e+05  4.613353e+02     108.576568      4.412711

region                                                          \
sector             19            20         21t22             23
stressor
CO2         96.323715    152.732847   2170.361889    8058.147997
CH4          6.252038     64.497627    196.132563   33393.021316
N2O          1.358277     14.745548    111.627792     146.815518
NOX       1012.797200   9028.943174  36118.410462   20162.149026
SOX        253.213989   2257.366743  17059.702285  108904.020068
CO        5173.754239  46123.284133  90805.126903   65080.776447
NMVOC     1646.038543  14674.199801  31944.464014  127366.360161
NH3          0.462632     13.142829     47.789610       4.104788

region                                ...           RoW                \
sector               24           25  ...            63             64
stressor                              ...
CO2         9119.700257    82.396753  ...  4.519067e+04   22917.441324
CH4          778.097344    24.219940  ...  2.066574e+04    2367.344909
N2O         9240.259497     6.956387  ...  7.461038e+02     320.047984
NOX        32405.126521   643.318443  ...  1.590800e+05   93897.479993
SOX        79131.591852   160.838941  ...  5.841325e+04   34478.601877
CO        406113.713306  3286.315879  ...  1.124907e+06  663979.652027
NMVOC     111694.012236  1045.546880  ...  3.455864e+05  203983.475230
NH3          358.482070     4.786318  ...  4.530937e+02     316.828348

region                                                          \
sector                J             70         71t74             L
stressor
CO2         17723.690229   13910.567504  5.601616e+04  1.098584e+05
CH4          4044.853398    6769.992126  1.631978e+04  8.902853e+04
N2O           356.817406     269.634242  1.250597e+03  3.028098e+03
NOX         77084.038042   81937.152824  2.207988e+05  4.710649e+05
SOX         28304.804973   30086.840151  8.107600e+04  1.729722e+05
```

(continues on next page)

(continued from previous page)

```
CO        545086.329897  579404.284591  1.561340e+06  3.331053e+06
NMVOC     167457.848344  178000.785375  4.796646e+05  1.023344e+06
NH3          233.201236     313.192459  1.620282e+03  1.332001e+03

region
sector                M             N             O    P
stressor
CO2         23671.960753  4.305903e+04  4.631917e+04  0.0
CH4          4809.795338  1.357472e+04  1.341403e+07  0.0
N2O           266.198531  7.628004e+03  9.028870e+04  0.0
NOX        105912.353518  1.790062e+05  1.693474e+05  0.0
SOX         38890.392703  6.573001e+04  6.218337e+04  0.0
CO         748940.734503  1.265811e+06  1.197511e+06  0.0
NMVOC      230084.661930  3.888741e+05  3.678914e+05  0.0
NH3           107.031855  5.911219e+02  4.808552e+03  0.0

[8 rows x 1435 columns]
```

If a WIOD SEA file is present (at the root of path or in a folder named 'SEA' - only one file!), the labor data of this file gets included in the factor_input extension (calculated for the the three skill levels available). The monetary data in this file is not added because it is only given in national currency:

[7]:
```
wiod2007.SEA.F
```

[7]:
```
region            AUS                                                  \
sector            AtB           C       15t16      17t18        19
inputtype
EMP        349.906604  147.955799  189.229541  49.152618  4.174751
EMPE       177.972976  145.153389  183.691303  40.240016  3.168873
H_EMP      743.950259  326.446887  365.287608  89.988634  8.279106
H_EMPE     367.588214  321.341746  351.470490  74.647046  6.197284

region                                                          ...  RoW ␣
↪\
sector             20       21t22         23         24         25  ...   63
inputtype                                                       ...
EMP         52.350134  124.076825   5.886915  46.400859  37.624869  ...  0.0
EMPE        44.836024  117.458029   5.859025  45.338232  36.800479  ...  0.0
H_EMP      108.256975  229.727320  11.710741  89.721636  73.069805  ...  0.0
H_EMPE      92.854499  218.041008  11.684120  88.045761  71.235855  ...  0.0

region
sector       64    J   70 71t74    L    M    N    O    P
inputtype
EMP         0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
EMPE        0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
H_EMP       0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
H_EMPE      0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0

[4 rows x 1435 columns]
```

Provenance tracking and additional meta data is availabe in the field `meta`:

[8]:
```
print(wiod2007.meta)
```

```
Description: WIOD metadata file for pymrio
MRIO Name: WIOD
System: industry-by-industry
Version: data13
File: /tmp/mrios/WIOD2013/metadata.json
History:
20210224 11:49:16 - FILEIO -  Extension wat parsed from /tmp/mrios/WIOD2013
20210224 11:49:15 - FILEIO -  Extension mat parsed from /tmp/mrios/WIOD2013
20210224 11:49:14 - FILEIO -  Extension lan parsed from /tmp/mrios/WIOD2013
20210224 11:49:13 - FILEIO -  Extension EU parsed from /tmp/mrios/WIOD2013
20210224 11:49:11 - FILEIO -  Extension EM parsed from /tmp/mrios/WIOD2013
20210224 11:49:09 - FILEIO -  Extension CO2 parsed from /tmp/mrios/WIOD2013
20210224 11:49:08 - FILEIO -  Extension AIR parsed from /tmp/mrios/WIOD2013
20210224 11:49:06 - FILEIO -  SEA file extension parsed from /tmp/mrios/
↪WIOD2013
20210224 11:48:52 - METADATA_CHANGE -  Changed parameter "system" from "IxI"␣
↪to "industry-by-industry"
20210224 11:48:52 - FILEIO -  WIOD data parsed from /tmp/mrios/WIOD2013/
↪wiot07_row_apr12.xlsx
 ... (more lines in history)
```

WIOD provides three different sector/final demand categories naming schemes. The one to use for pymrio can specified by passing a tuple `names=` with:

1) 'isic': ISIC rev 3 Codes - available for interindustry flows and final demand rows.
2) 'full': Full names - available for final demand rows and final demand columns (categories) and interindustry flows.
3) 'c_codes' : WIOD specific sector numbers, available for final demand rows and columns (categories) and interindustry flows.

Internally, the parser relies on 1) for the interindustry flows and 3) for the final demand categories. This is the default and will also be used if just 'isic' gets passed ('c_codes' also replace 'isic' if this was passed for final demand categories). To specify different finial consumption category names, pass a tuple with (sectors/interindustry classification, fd categories), eg ('isic', 'full'). Names are case insensitive and passing the first character is sufficient.

For example, for loading wiod with full sector names:

[9]:
```
wiod2007_full = pymrio.parse_wiod(year=2007, path=wiod_storage, names=("full",
↪ "full"))
wiod2007_full.Y.head()
```

[9]:
```
region                                                                  ␣
↪            AUS  \
category                                          Final consumption␣
↪expenditure by households
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                       ␣
↪    8222.798980
```

(continues on next page)

(continued from previous page)

```
       Mining and Quarrying                                                    ␣
↪    2525.696909
       Food, Beverages and Tobacco                                             ␣
↪   28619.069479
       Textiles and Textile Products                                           ␣
↪    1837.921033
       Leather, Leather and Footwear                                           ␣
↪     473.971219

region                                                                         ␣
↪                                                        \
category                                          Final consumption␣
↪expenditure by non-profit organisations serving households (NPISH)
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                              ␣
↪                    0.0
       Mining and Quarrying                                                    ␣
↪                    0.0
       Food, Beverages and Tobacco                                             ␣
↪                    0.0
       Textiles and Textile Products                                           ␣
↪                    0.0
       Leather, Leather and Footwear                                           ␣
↪                    0.0

region                                                                         ␣
↪                \
category                                          Final consumption␣
↪expenditure by government
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                              ␣
↪     184.205180
       Mining and Quarrying                                                    ␣
↪     137.230459
       Food, Beverages and Tobacco                                             ␣
↪      54.444946
       Textiles and Textile Products                                           ␣
↪       8.595108
       Leather, Leather and Footwear                                           ␣
↪       2.216545

region                                                                         ␣
↪   \
category                                          Gross fixed capital␣
↪formation
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                  2924.
↪034910
       Mining and Quarrying                                        4150.
```

(continues on next page)

(continued from previous page)

```
↪190757
       Food, Beverages and Tobacco                                    457.
↪899386
       Textiles and Textile Products                                  453.
↪941827
       Leather, Leather and Footwear                                  117.
↪064525

region                                                                  ␣
↪          \
category                                        Changes in inventories and␣
↪valuables
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                       ␣
↪1280.356810
       Mining and Quarrying                                             -
↪292.042008
       Food, Beverages and Tobacco                                      ␣
↪404.590962
       Textiles and Textile Products                                     -
↪42.196861
       Leather, Leather and Footwear                                     -
↪10.881914

region                                                                  ␣
↪            AUT  \
category                                        Final consumption␣
↪expenditure by households
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                       ␣
↪       0.422485
       Mining and Quarrying                                             ␣
↪       0.666800
       Food, Beverages and Tobacco                                      ␣
↪       5.606114
       Textiles and Textile Products                                    ␣
↪       1.522250
       Leather, Leather and Footwear                                    ␣
↪       0.476768

region                                                                  ␣
↪                                                    \
category                                        Final consumption␣
↪expenditure by non-profit organisations serving households (NPISH)
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                       ␣
↪                   0.0
       Mining and Quarrying                                             ␣
↪                   0.0
```

(continues on next page)

(continued from previous page)

```
       Food, Beverages and Tobacco                                          ␣
↪                  0.0
       Textiles and Textile Products                                        ␣
↪                  0.0
       Leather, Leather and Footwear                                        ␣
↪                  0.0

region                                                                      ␣
↪                \
category                                          Final consumption␣
↪expenditure by government
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                           ␣
↪       0.025177
       Mining and Quarrying                                                 ␣
↪       0.000000
       Food, Beverages and Tobacco                                          ␣
↪       0.037221
       Textiles and Textile Products                                        ␣
↪       0.006089
       Leather, Leather and Footwear                                        ␣
↪       0.001907

region                                                                      ␣
↪   \
category                                          Gross fixed capital␣
↪formation
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                    0.
↪000000
       Mining and Quarrying                                          0.
↪012719
       Food, Beverages and Tobacco                                   0.
↪031606
       Textiles and Textile Products                                 0.
↪050338
       Leather, Leather and Footwear                                 0.
↪015766

region                                                                      ␣
↪          \
category                                          Changes in inventories and␣
↪valuables
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                           ␣
↪      0.0
       Mining and Quarrying                                                 ␣
↪      0.0
       Food, Beverages and Tobacco                                          ␣
```

(continues on next page)

(continued from previous page)

```
↪      0.0
       Textiles and Textile Products                                          ␣
↪      0.0
       Leather, Leather and Footwear                                          ␣
↪      0.0

region                                                 ...  \
category                                               ...
region sector                                          ...
AUS    Agriculture, Hunting, Forestry and Fishing      ...
       Mining and Quarrying                            ...
       Food, Beverages and Tobacco                     ...
       Textiles and Textile Products                   ...
       Leather, Leather and Footwear                   ...

region                                                                        ␣
↪             USA  \
category                                          Final consumption␣
↪expenditure by households
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                             ␣
↪       69.083262
       Mining and Quarrying                                                   ␣
↪         0.490308
       Food, Beverages and Tobacco                                            ␣
↪     1631.773339
       Textiles and Textile Products                                          ␣
↪      158.781552
       Leather, Leather and Footwear                                          ␣
↪       49.730261

region                                                                        ␣
↪                                                        \
category                                          Final consumption␣
↪expenditure by non-profit organisations serving households (NPISH)
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                             ␣
↪                    0.0
       Mining and Quarrying                                                   ␣
↪                    0.0
       Food, Beverages and Tobacco                                            ␣
↪                    0.0
       Textiles and Textile Products                                          ␣
↪                    0.0
       Leather, Leather and Footwear                                          ␣
↪                    0.0

region                                                                        ␣
↪                  \
```

(continues on next page)

(continued from previous page)

```
category                                          Final consumption␣
↪expenditure by government
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                      ␣
↪              0.0
       Mining and Quarrying                                            ␣
↪              0.0
       Food, Beverages and Tobacco                                     ␣
↪              0.0
       Textiles and Textile Products                                   ␣
↪              0.0
       Leather, Leather and Footwear                                   ␣
↪              0.0

region                                                                 ␣
↪    \
category                                          Gross fixed capital␣
↪formation
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                    0.
↪000000
       Mining and Quarrying                                          0.
↪764753
       Food, Beverages and Tobacco                                   0.
↪554414
       Textiles and Textile Products                                 4.
↪737164
       Leather, Leather and Footwear                                 1.
↪483677

region                                                                 ␣
↪           \
category                                          Changes in inventories and␣
↪valuables
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                      ␣
↪      0.0
       Mining and Quarrying                                            ␣
↪      0.0
       Food, Beverages and Tobacco                                     ␣
↪      0.0
       Textiles and Textile Products                                   ␣
↪      0.0
       Leather, Leather and Footwear                                   ␣
↪      0.0

region                                                                 ␣
↪              RoW  \
category                                          Final consumption␣
```

(continues on next page)

(continued from previous page)

```
↪expenditure by households
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                          ␣
↪      107.088905
       Mining and Quarrying                                                ␣
↪        0.088067
       Food, Beverages and Tobacco                                         ␣
↪     2918.131643
       Textiles and Textile Products                                       ␣
↪       86.189090
       Leather, Leather and Footwear                                       ␣
↪        7.748815

region                                                                     ␣
↪                                                          \
category                                          Final consumption␣
↪expenditure by non-profit organisations serving households (NPISH)
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                          ␣
↪                    0.0
       Mining and Quarrying                                                ␣
↪                    0.0
       Food, Beverages and Tobacco                                         ␣
↪                    0.0
       Textiles and Textile Products                                       ␣
↪                    0.0
       Leather, Leather and Footwear                                       ␣
↪                    0.0

region                                                                     ␣
↪                  \
category                                          Final consumption␣
↪expenditure by government
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                          ␣
↪        1.798976
       Mining and Quarrying                                                ␣
↪        0.004956
       Food, Beverages and Tobacco                                         ␣
↪        0.969600
       Textiles and Textile Products                                       ␣
↪        0.969294
       Leather, Leather and Footwear                                       ␣
↪        0.087144

region                                                                     ␣
↪    \
category                                          Gross fixed capital␣
↪formation
```

(continues on next page)

(continued from previous page)

```
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                    10.
↪713377
       Mining and Quarrying                                           0.
↪202258
       Food, Beverages and Tobacco                                    3.
↪599341
       Textiles and Textile Products                                  2.
↪892659
       Leather, Leather and Footwear                                  0.
↪260064

region
category                                          Changes in inventories and␣
↪valuables
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                      ␣
↪0.000770
       Mining and Quarrying                                           -
↪0.004381
       Food, Beverages and Tobacco                                     ␣
↪0.001523
       Textiles and Textile Products                                   ␣
↪0.000035
       Leather, Leather and Footwear                                  -
↪0.000005

[5 rows x 205 columns]
```

The wiod parsing routine provides some more options - for a full specification see *the API reference*

Parsing multiple years

Multiple years can be passed by running the parser in a for loop.

6.5 Working with the EXIOBASE EE MRIO database

6.5.1 Getting EXIOBASE

EXIOBASE 1 (developed in the fp6 project EXIOPOL), EXIOBASE 2 (outcome of the fp7 project CREEA) and EXIOBASE 3 (outcome of the fp7 project DESIRE) are available on the EXIOBASE webpage.

You need to register before you can download the full dataset.

Further information on the different EXIOBASE versions can be found in corresponding method papers.

- EXIOBASE 1: Tukker et al. 2013. Exiopol – Development and Illustrative Analyses of a Detailed Global MR EE SUT/IOT. Economic Systems Research 25(1), 50-70

- EXIOBASE 2: Wood et al. 2015. Global Sustainability Accounting—Developing EXIOBASE for Multi-Regional Footprint Analysis. Sustainability 7(1), 138-163
- EXIOBASE 3: Stadler et al. 2018. EXIOBASE 3: Developing a Time Series of Detailed Environmentally Extended Multi-Regional Input-Output Tables. Journal of Industrial Ecology 22(3), 502-515

EXIOBASE 1

To download EXIOBASE 1 for the use with pymrio, navigate to the EXIOBASE webpage - section(tab) "Data Download" - "EXIOBASE 1 - full dataset" and download either

- pxp_ita_44_regions_coeff_txt for the product by product (pxp) MRIO system or
- ixi_fpa_44_regions_coeff_txt for the industry by industry (ixi) MRIO system or
- pxp_ita_44_regions_coeff_src_txt for the product by product (pxp) MRIO system with emission data per source or
- ixi_fpa_44_regions_coeff_src_txt for the industry by industry (ixi) wMRIO system with emission data per source.

The links above directly lead to the required file(s), but remember that you need to be logged in to access them.

The Pymrio parser works with the compressed (zip) files as well as the unpacked files. If you want to unpack the files, make sure that you store them in different folders since they unpack in the current directory.

EXIOBASE 2

EXIOBASE 3 is available at the EXIOBASE webpage at the section (tab) tab "Data Download" - "EXIOBASE 2 - full dataset".

You can download either

- MrIOT PxP ita coefficient version2 2 2 for the product by product (pxp) MRIO system or
- MrIOT IxI fpa coefficient version2 2 2 for the industry by industry (ixi) MRIO system.

The links above directly lead to the required file(s), but remember that you need to be logged in to access them.

The pymrio parser works with the compressed (zip) files as well as the unpacked files. You can unpack the files together in one directory (unpacking creates a separate folder for each EXIOBASE 2 version). The unpacking of the PxP version also creates a folder "__MACOSX" - you can delete this folder.

EXIOBASE 3

EXIOBASE 3 is available at the EXIOBASE webpage at the section (tab) tab "Data Download" - "EXIOBASE 3 - monetary". The EXIOBASE 3 parser works with both, the compressed zip archives and the extracted database.

6.5.2 Parsing

[1]:
```
import pymrio
```

For each publically available version of EXIOBASE pymrio provides a specific parser. All exiobase parser work with the zip archive (as downloaded from the exiobase webpage) or the extracted data.

To parse **EXIOBASE 1** use:

[2]:
```
exio1 = pymrio.parse_exiobase1(
    path="/tmp/mrios/exio1/zip/121016_EXIOBASE_pxp_ita_44_regions_coeff_txt.
↪zip"
)
```

```
/home/konstans/proj/pymrio/pymrio/tools/ioparser.py:373: FutureWarning: The␣
↪default value of regex will change from True to False in a future version.
  _new_unit = _unit.unit.str.replace("/" + mon_unit, "")
```

The parameter 'path' needs to point to either the folder with the extracted EXIOBASE1 files for the downloaded zip archive.

Similarly, **EXIOBASE 2** can be parsed by:

[3]:
```
exio2 = pymrio.parse_exiobase2(
    path="/tmp/mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip",
    charact=True,
    popvector="exio2",
)
```

The additional parameter 'charact' specifies if the characterization matrix provided with EXIOBASE 2 should be used. This can be specified with True or False; in addition, a custom one can be provided. In the latter case, pass the full path to the custom characterisatio file to 'charact'.

The parameter 'popvector' allows to pass information about the population per EXIOBASE2 country. This can either be a custom vector of, if 'exio2' is passed, the one provided with pymrio.

EXIOBASE 3 can be parsed by:

[4]:
```
exio3 = pymrio.parse_exiobase3(path="/tmp/mrios/exio3/zip/exiobase3.4_iot_
↪2009_pxp.zip")
```

Currently, no characterization or population vectors are provided for EXIOBASE 3.

For the rest of the tutorial, we use *exio2*; deleting *exio1* and *exio3* to free some memory:

[5]:
```
del exio1
del exio3
```

6.5.3 Exploring EXIOBASE

After parsing a EXIOBASE version, the handling of the database is the same as for any IO. Here we use the parsed EXIOBASE2 to explore some characteristics of the EXIBOASE system.

After reading the raw files, metadata about EXIOBASE can be accessed within the meta field:

```
[6]: exio2.meta
```

```
[6]: Description: Metadata for pymrio
MRIO Name: EXIOBASE
System: pxp
Version: 2.2.2
File: None
History:
20210224 11:48:56 - FILEIO -  EXIOBASE data F_Y_materials parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFDMaterials_version2.2.2.txt
20210224 11:48:56 - FILEIO -  EXIOBASE data F_Y_emissions parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFDEmissions_version2.2.2.txt
20210224 11:48:56 - FILEIO -  EXIOBASE data S_resources parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrResources_version2.2.2.txt
20210224 11:48:55 - FILEIO -  EXIOBASE data S_materials parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrMaterials_version2.2.2.txt
20210224 11:48:54 - FILEIO -  EXIOBASE data S_emissions parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrEmissions_version2.2.2.txt
20210224 11:48:52 - FILEIO -  EXIOBASE data S_factor_inputs parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFactorInputs_version2.2.2.txt
20210224 11:48:51 - FILEIO -  EXIOBASE data Y parsed from /tmp/mrios/exio2/
↪zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_coefficient_
↪version2.2.2/mrFinalDemand_version2.2.2.txt
20210224 11:48:51 - FILEIO -  EXIOBASE data A parsed from /tmp/mrios/exio2/
↪zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_coefficient_
↪version2.2.2/mrIot_version2.2.2.txt
```

Custom points can be added to the history in the meta record. For example:

```
[7]: exio2.meta.note("First test run of EXIOBASE 2")
exio2.meta
```

```
[7]: Description: Metadata for pymrio
MRIO Name: EXIOBASE
System: pxp
Version: 2.2.2
File: None
History:
20210224 11:50:36 - NOTE -  First test run of EXIOBASE 2
20210224 11:48:56 - FILEIO -  EXIOBASE data F_Y_materials parsed from /tmp/
```

(continues on next page)

(continued from previous page)

```
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFDMaterials_version2.2.2.txt
20210224 11:48:56 - FILEIO -  EXIOBASE data F_Y_emissions parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFDEmissions_version2.2.2.txt
20210224 11:48:56 - FILEIO -  EXIOBASE data S_resources parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrResources_version2.2.2.txt
20210224 11:48:55 - FILEIO -  EXIOBASE data S_materials parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrMaterials_version2.2.2.txt
20210224 11:48:54 - FILEIO -  EXIOBASE data S_emissions parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrEmissions_version2.2.2.txt
20210224 11:48:52 - FILEIO -  EXIOBASE data S_factor_inputs parsed from /tmp/
↪mrios/exio2/zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_
↪coefficient_version2.2.2/mrFactorInputs_version2.2.2.txt
20210224 11:48:51 - FILEIO -  EXIOBASE data Y parsed from /tmp/mrios/exio2/
↪zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_coefficient_
↪version2.2.2/mrFinalDemand_version2.2.2.txt
20210224 11:48:51 - FILEIO -  EXIOBASE data A parsed from /tmp/mrios/exio2/
↪zip/mrIOT_PxP_ita_coefficient_version2.2.2.zip/mrIOT_PxP_ita_coefficient_
↪version2.2.2/mrIot_version2.2.2.txt
```

To check for sectors, regions and extensions:

[8]:
```
exio2.get_sectors()
```

[8]:
```
Index(['Paddy rice', 'Wheat', 'Cereal grains nec', 'Vegetables, fruit, nuts',
       'Oil seeds', 'Sugar cane, sugar beet', 'Plant-based fibers',
       'Crops nec', 'Cattle', 'Pigs',
       ...
       'Paper for treatment: landfill',
       'Plastic waste for treatment: landfill',
       'Inert/metal/hazardous waste for treatment: landfill',
       'Textiles waste for treatment: landfill',
       'Wood waste for treatment: landfill',
       'Membership organisation services n.e.c.',
       'Recreational, cultural and sporting services', 'Other services',
       'Private households with employed persons',
       'Extra-territorial organizations and bodies'],
      dtype='object', name='sector', length=200)
```

[9]:
```
exio2.get_regions()
```

[9]:
```
Index(['AT', 'BE', 'BG', 'CY', 'CZ', 'DE', 'DK', 'EE', 'ES', 'FI', 'FR', 'GR',
       'HU', 'IE', 'IT', 'LT', 'LU', 'LV', 'MT', 'NL', 'PL', 'PT', 'RO', 'SE',
       'SI', 'SK', 'GB', 'US', 'JP', 'CN', 'CA', 'KR', 'BR', 'IN', 'MX', 'RU',
       'AU', 'CH', 'TR', 'TW', 'NO', 'ID', 'ZA', 'WA', 'WL', 'WE', 'WF', 'WM
↪'],
      dtype='object', name='region')
```

[10]:
```
list(exio2.get_extensions())
```

[10]:
```
['factor_inputs', 'emissions', 'materials', 'resources', 'impact']
```

6.5.4 Calculating the system and extension results

The following command checks for missing parts in the system and calculates them. In case of the parsed EXIOBASE this includes A, L, multipliers M, footprint accounts, ..

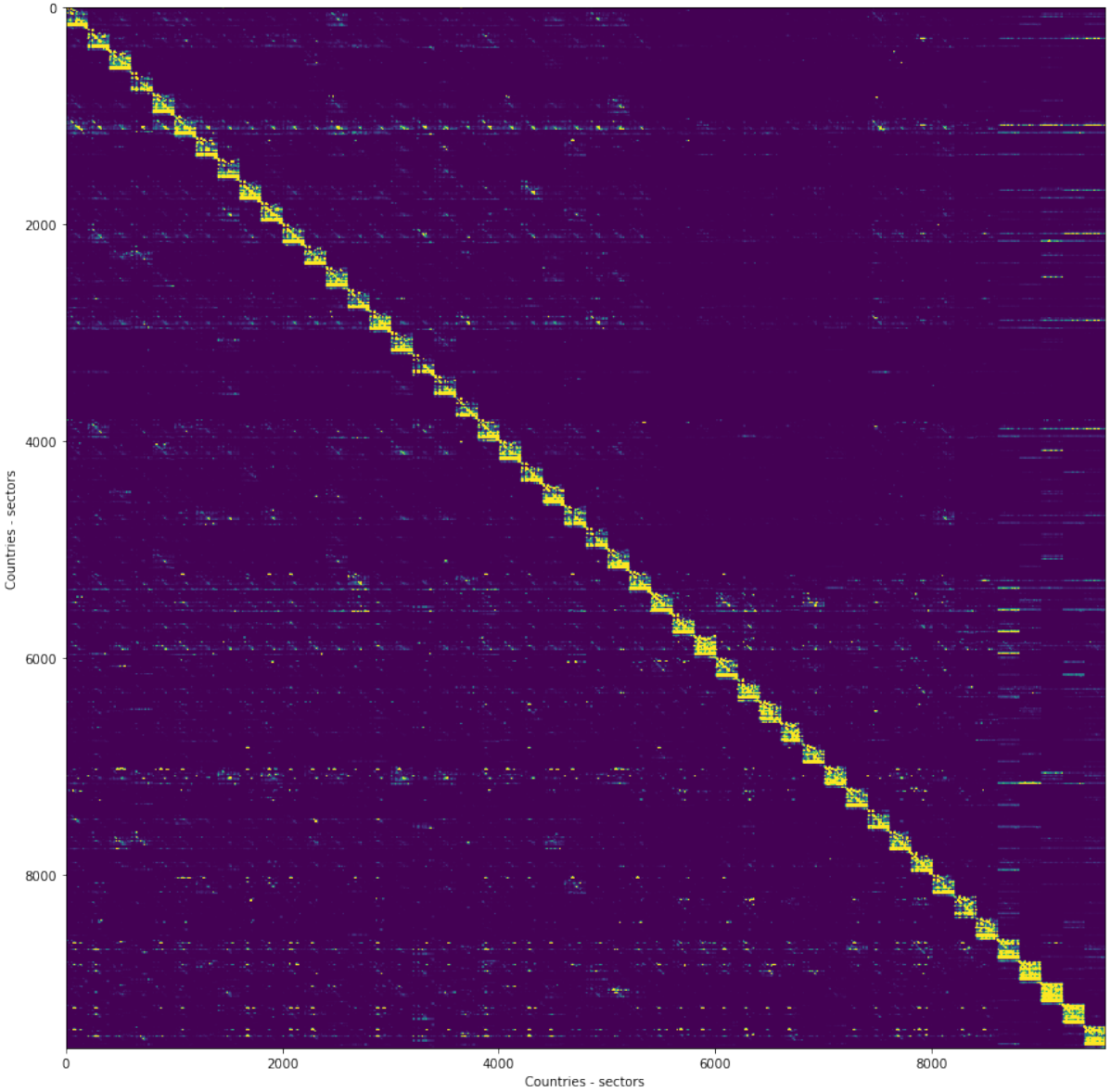
[11]:
```
exio2.calc_all()
```

[11]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f337397a520>
```

6.5.5 Exploring the results

[12]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 15))
plt.imshow(exio2.A, vmax=1e-3)
plt.xlabel("Countries - sectors")
plt.ylabel("Countries - sectors")
plt.show()
```



The available impact data can be checked with:

[13]:
```
list(exio2.impact.get_rows())
```

[13]:
```
['Value Added',
 'Employment',
 'Employment hour',
 'abiotic depletion (elements, ultimate ultimate reserves)',
 'abiotic depletion (fossil fuels)',
 'abiotic depletion (elements, reserve base)',
 'abiotic depletion (elements, economic reserve)',
 'Landuse increase of land competition',
 'global warming (GWP100)',
 'global warming net (GWP100 min)',
 'global warming net (GWP100 max)',
 'global warming (GWP20)',
 'global warming (GWP500)',
```

(continues on next page)

(continued from previous page)

```
'ozone layer depletion (ODP steady state)',
'ozone layer depletion (ODP5)',
'ozone layer depletion (ODP10)',
'ozone layer depletion (ODP15)',
'ozone layer depletion (ODP20)',
'ozone layer depletion (ODP25)',
'ozone layer depletion (ODP30)',
'ozone layer depletion (ODP40)',
'human toxicity (HTP inf)',
'Freshwater aquatic ecotoxicity (FAETP inf)',
'Marine aquatic ecotoxicity (MAETP inf)',
'Freshwater sedimental ecotoxicity (FSETP inf)',
'Marine sedimental ecotoxicity (MSETP inf)',
'Terrestrial ecotoxicity (TETP inf)',
'human toxicity (HTP20)',
'Freshwater aquatic ecotoxicity (FAETP20)',
'Marine aquatic ecotoxicity (MAETP20)',
'Freshwater sedimental ecotoxicity (FSETP20)',
'Marine sedimental ecotoxicity (MSETP20)',
'Terrestrial ecotoxicity (TETP20)',
'human toxicity (HTP100)',
'Freshwater aquatic ecotoxicity (FAETP100)',
'Marine aquatic ecotoxicity (MAETP100)',
'Freshwater sedimental ecotoxicity (FSETP100)',
'Marine sedimental ecotoxicity (MSETP100)',
'Terrestrial ecotoxicity (TETP100)',
'human toxicity (HTP500)',
'Freshwater aquatic ecotoxicity (FAETP500)',
'Marine aquatic ecotoxicity (MAETP500)',
'Freshwater sedimental ecotoxicity (FSETP500)',
'Marine sedimental ecotoxicity (MSETP500)',
'Terrestrial ecotoxicity (TETP500)',
'Human toxicity (USEtox) 2008',
'Fresh water Ecotoxicity (USEtox) 2008',
'Human toxicity (USEtox) 2010',
'Fresh water Ecotoxicity (USEtox) 2010',
'photochemical oxidation (high NOx)',
'photochemical oxidation (low NOx)',
'photochemical oxidation (MIR; very high NOx)',
'photochemical oxidation (MOIR; high NOx)',
'photochemical oxidation (EBIR; low NOx)',
'acidification (incl. fate, average Europe total, A&B)',
'acidification (fate not incl.)',
'eutrophication (fate not incl.)',
'eutrophication (incl. fate, average Europe total, A&B)',
'radiation',
'odour',
'EPS',
'Carcinogenic effects on humans (H.A)',
```

(continues on next page)

(continued from previous page)

```
 'Respiratory effects on humans caused by organic substances (H.A)',
 'Respiratory effects on humans caused by inorganic substances (H.A)',
 'Damages to human health caused by climate change (H.A)',
 'Human health effects caused by ionising radiation (H.A)',
 'Human health effects caused by ozone layer depletion (H.A)',
 'Damage to Ecosystem Quality caused by ecotoxic emissions (H.A)',
 'Damage to Ecosystem Quality caused by the combined effect of acidification␣
↪and eutrophication (H.A)',
 'Damage to Ecosystem Quality caused by land occupation (H.A)',
 'Damage to Ecosystem Quality caused by land conversion (H.A)',
 'Damage to Resources caused by extraction of minerals (H.A)',
 'Damage to Resources caused by extraction of fossil fuels (H.A)',
 'Carcinogenic effects on humans (E.E)',
 'Respiratory effects on humans caused by organic substances (E.E)',
 'Respiratory effects on humans caused by inorganic substances (E.E)',
 'Damages to human health caused by climate change (E.E)',
 'Human health effects caused by ionising radiation (E.E)',
 'Human health effects caused by ozone layer depletion (E.E)',
 'Damage to Ecosystem Quality caused by ecotoxic emissions (E.E))',
 'Damage to Ecosystem Quality caused by the combined effect of acidification␣
↪and eutrophication (E.E)',
 'Damage to Ecosystem Quality caused by land occupation (E.E)',
 'Damage to Ecosystem Quality caused by land conversion (E.E)',
 'Damage to Resources caused by extraction of minerals (E.E)',
 'Damage to Resources caused by extraction of fossil fuels (E.E)',
 'Carcinogenic effects on humans (I.I)',
 'Respiratory effects on humans caused by organic substances (I.I)',
 'Respiratory effects on humans caused by inorganic substances (I.I)',
 'Damages to human health caused by climate change (I.I)',
 'Human health effects caused by ionising radiation (I.I)',
 'Human health effects caused by ozone layer depletion (I.I)',
 'Damage to Ecosystem Quality caused by ecotoxic emissions (I.I)',
 'Damage to Ecosystem Quality caused by the combined effect of acidification␣
↪and eutrophication (I.I)',
 'Damage to Ecosystem Quality caused by land occupation (I.I)',
 'Damage to Ecosystem Quality caused by land conversion (I.I)',
 'Damage to Resources caused by extraction of minerals (I.I)',
 'Damage to Resources caused by extraction of fossil fuels (I.I)',
 'photochemical oxidation (high NOx)(incl. NOx average, NMVOC average)',
 'photochemical oxidation (high NOx)(incl. NMVOC average)',
 'human toxicity HTP inf. (incl. PAH average, Xylene average, NMVOC average)',
 'Freshwater aquatic ecotoxicity FAETP inf. (incl. PAH average, Xylene␣
↪average, NMVOC average)',
 'Marine aquatic ecotoxicity MAETP inf. (incl. PAH average, Xylene average,␣
↪NMVOC average)',
 'Terrestrial ecotoxicity TETP inf. (incl. PAH average, Xylene average, NMVOC␣
↪average)',
 'global warming GWP100 (incl. NMVOC average)',
 'ozone layer depletion ODP steady state (incl. NMVOC average)',
```

(continues on next page)

(continued from previous page)

```
 'Total Emission relevant energy use',
 'Total Energy inputs from nature',
 'Total Energy supply',
 'Total Energy Use',
 'Total Heat rejected to fresh water',
 'Domestic Extraction',
 'Unused Domestic Extraction',
 'Water Consumption Green - Agriculture',
 'Water Consumption Blue - Agriculture',
 'Water Consumption Blue - Livestock',
 'Water Consumption Blue - Manufacturing',
 'Water Consumption Blue - Electricity',
 'Water Consumption Blue - Domestic',
 'Water Consumption Blue - Total',
 'Water Withdrawal Blue - Manufacturing',
 'Water Withdrawal Blue - Electricity',
 'Water Withdrawal Blue - Domestic',
 'Water Withdrawal Blue - Total',
 'Land use']
```

And to get for example the footprint of a specific impact do:

[14]:
```
print(exio2.impact.unit.loc["global warming (GWP100)"])
exio2.impact.D_cba_reg.loc["global warming (GWP100)"]
```

```
unit    kg CO2 eq.
Name: global warming (GWP100), dtype: object
```

[14]:
```
region
AT    1.450787e+11
BE    1.991422e+11
BG    6.266676e+10
CY    1.556996e+10
CZ    1.471491e+11
DE    1.394892e+12
DK    1.079304e+11
EE    2.381673e+10
ES    6.079175e+11
FI    1.153875e+11
FR    8.019998e+11
GR    2.247927e+11
HU    9.096635e+10
IE    9.591233e+10
IT    8.419421e+11
LT    3.366823e+10
LU    1.467799e+10
LV    2.255212e+10
MT    5.014763e+09
NL    2.992112e+11
PL    4.136385e+11
PT    1.120749e+11
```

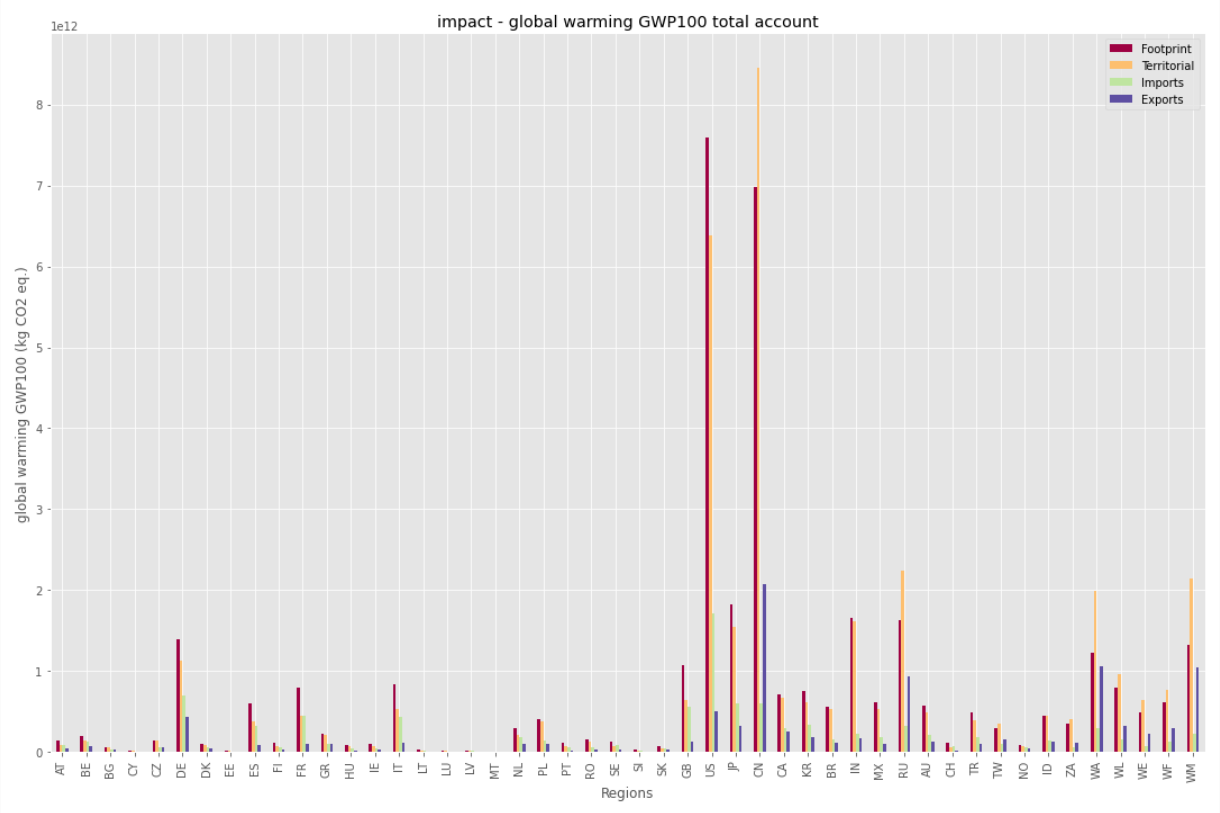
(continues on next page)

(continued from previous page)

```
RO    1.543358e+11
SE    1.282029e+11
SI    3.239223e+10
SK    6.911104e+10
GB    1.073548e+12
US    7.591895e+12
JP    1.825128e+12
CN    6.986984e+12
CA    7.142173e+11
KR    7.566406e+11
BR    5.595118e+11
IN    1.658771e+12
MX    6.219372e+11
RU    1.635710e+12
AU    5.715893e+11
CH    1.201448e+11
TR    4.939783e+11
TW    2.924074e+11
NO    8.791708e+10
ID    4.552600e+11
ZA    3.547961e+11
WA    1.224565e+12
WL    7.970228e+11
WE    4.931660e+11
WF    6.100073e+11
WM    1.329488e+12
Name: global warming (GWP100), dtype: float64
```

6.5.6 Visualizing the data

[15]:
```
with plt.style.context("ggplot"):
    exio2.impact.plot_account(["global warming (GWP100)"], figsize=(15, 10))
    plt.show()
```



See the other notebooks for further information on *aggregation* and *file io*.

6.6 Parsing the Eora26 EE MRIO database

6.6.1 Getting Eora26

You need to register at http://www.worldmrio.com to download Eora 26 database. You can download the files from http://www.worldmrio.com/simplified. Or you can download these files with the pymrio automatic downloader as described at *Eora26 download*.

6.6.2 Parse

To parse a single year do:

[1]:
```
import pymrio
```

[2]:
```
eora_storage = "/tmp/mrios/eora26"
```

[ ]:
```
eora_meta = pymrio.download_eora26(
    storage_folder=eora_storage,
    email="<Your Eora account email>",
    password="<Your Eora account password>",
)
```

[3]:
```
eora = pymrio.parse_eora26(year=2005, path=eora_storage)
```

```
/home/konstans/bin/anaconda3/envs/pymrio_dev/lib/python3.7/site-packages/
↪pandas/core/generic.py:3887: PerformanceWarning: dropping on a non-
↪lexsorted multi-index without a level parameter may impact performance.
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

6.6.3 Explore

Eora includes (almost) all countries:

[4]:
```
eora.get_regions()
```

[4]:
```
Index(['AFG', 'ALB', 'DZA', 'AND', 'AGO', 'ATG', 'ARG', 'ARM', 'ABW', 'AUS',
       ...
       'TZA', 'USA', 'URY', 'UZB', 'VUT', 'VEN', 'VNM', 'YEM', 'ZMB', 'ZWE'],
      dtype='object', name='region', length=189)
```

This can easily be aggregated to, for example, the OECD/NON_OECD countries with the help of the country converter coco.

[5]:
```
import country_converter as coco
```

[6]:
```
eora.aggregate(
    region_agg=coco.agg_conc(
        original_countries="Eora", aggregates=["OECD"], missing_countries=
↪"NON_OECD"
    )
)
```

[6]:
```
<pymrio.core.mriosystem.IOSystem at 0x7fb7f2a14e90>
```

[7]:
```
eora.get_regions()
```

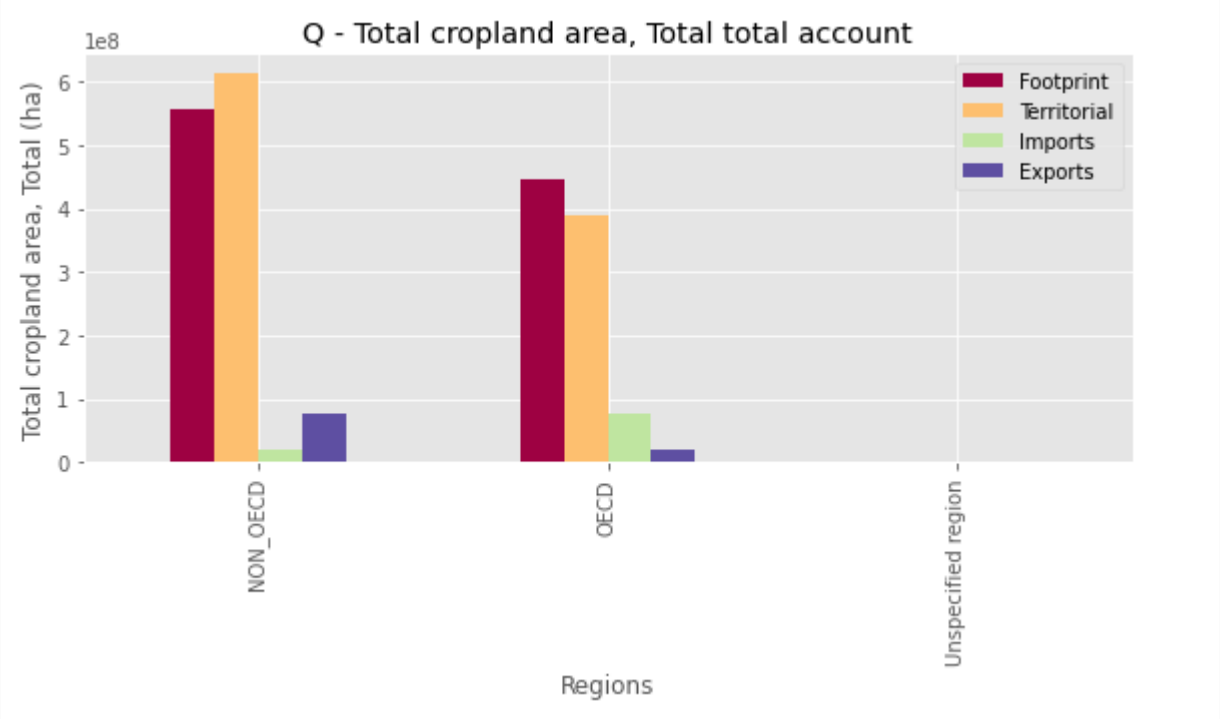
[7]:
```
Index(['NON_OECD', 'OECD', 'Unspecified region'], dtype='object', name='region
↪')
```

[8]:
```
eora.calc_all()
```

[8]:
```
<pymrio.core.mriosystem.IOSystem at 0x7fb7f2a14e90>
```

[9]:
```
import matplotlib.pyplot as plt

with plt.style.context("ggplot"):
    eora.Q.plot_account(("Total cropland area", "Total"), figsize=(8, 5))
    plt.show()
```



See the other notebooks for further information on *aggregation* and *file io*.

6.7 Working with the OECD - ICIO database

The OECD Inter-Country Input-Output tables (ICIO) are available on the OECD webpage.

The parsing function >parse_oecd< works for both, the 2016 and 2018 release.

The tables can either be downloaded manually (using the csv format), or the pymrio *OECD automatic downloader can be used*.

For example, to get the 2011 table of the 2018 release do:

[1]:
```
import pymrio
```

[2]:
```
from pathlib import Path
```

[3]:
```
oecd_storage = Path("/tmp/mrios/OECD")
```

[4]:
```
meta_2018_download = pymrio.download_oecd(storage_folder=oecd_storage,␣
↪years=[2011])
```

OECD provides the data compressed in zip files. The pymrio oecd parser works with both, the compressed and unpacked version.

6.7.1 Parsing

To parse a single year of the database, either specify a path and year:

[5]:
```
oecd_path_year = pymrio.parse_oecd(path=oecd_storage, year=2011)
```

Or directly specify a file to parse:

[6]:
```
oecd_file = pymrio.parse_oecd(path=oecd_storage / "ICIO2018_2011.zip")
```

[7]:
```
oecd_path_year == oecd_file
```

[7]:
```
True
```

Note: The original OECD ICIO tables provide some disaggregation of the Mexican and Chinese tables for the interindustry flows. The pymrio parser automatically aggregates these into Chinese And Mexican totals. Thus, the MX1, MX2, .. and CN1, CN2, … entries are aggregated into MEX and CHN.

Currently, the parser only includes the value added and taxes data given in original file as satellite accounts. These are accessable in the extension "factor_inputs":

[8]:
```
oecd_file.factor_inputs.F.head()
```

[8]:
```
region           ARG                                                  \
sector         01T03     05T06     07T08        09     10T12     13T15
inputtype
AUS_TAXSUB  0.121714  0.017712  0.017512  0.017816  0.001661  0.095973
AUT_TAXSUB  0.073775  0.033071  0.013816  0.005258  0.078889  0.038198
BEL_TAXSUB  0.185901  0.062070  0.024458  0.011260  0.148690  0.094059
CAN_TAXSUB  1.190519  0.289064  0.230554  0.056490  0.455820  0.150127
CHL_TAXSUB  0.800514  0.253973  0.176948  0.044563  1.024921  1.232006

region                                              ...       ZAF            \
sector            16     17T18        19     20T21  ...        61     62T63
inputtype                                           ...
AUS_TAXSUB  0.007718  0.034678  0.128502  0.162166  ...  0.402171  0.674091
AUT_TAXSUB  0.010675  0.025981  0.015319  0.083526  ...  0.079267  0.042284
BEL_TAXSUB  0.012182  0.057137  0.061476  0.196174  ...  0.117514  0.092787
CAN_TAXSUB  0.035587  0.076343  0.871824  0.537732  ...  0.111660  0.064595
CHL_TAXSUB  0.102607  0.321750  0.381983  0.583021  ...  0.006196  0.006398

region                                                                \
sector         64T66        68     69T82        84        85     86T88
inputtype
AUS_TAXSUB  0.100344  0.626509  1.373728  1.610570  0.246573  0.634604
AUT_TAXSUB  0.022676  0.061500  0.165165  0.420449  0.042098  0.449250
BEL_TAXSUB  0.078851  0.145796  0.394643  0.535211  0.088572  0.357782
CAN_TAXSUB  0.022136  0.065346  0.174113  0.272304  0.037256  0.102510
CHL_TAXSUB  0.003776  0.019049  0.029477  0.029843  0.013084  0.022314

region
sector         90T96 97T98
inputtype
```

(continues on next page)

(continued from previous page)

```
AUS_TAXSUB  0.333580   0.0
AUT_TAXSUB  0.072340   0.0
BEL_TAXSUB  0.092843   0.0
CAN_TAXSUB  0.086703   0.0
CHL_TAXSUB  0.009911   0.0

[5 rows x 2340 columns]
```

Handling of the data happens similar to the other databases, see for example *"Exploring EXIOBASE"*.

6.8 Loading, saving and exporting data

Pymrio includes several functions for data reading and storing. This section presents the methods to use for saving and loading data already in a pymrio compatible format. For parsing raw MRIO data see the different tutorials for *working with available MRIO databases*.

Here, we use the included small test MRIO system to highlight the different function. The same functions are available for any MRIO loaded into pymrio. Expect, however, significantly decreased performance due to the size of real MRIO system.

[1]:
```
import pymrio
import os

io = pymrio.load_test().calc_all()
```

6.8.1 Basic save and read

To save the full system, use:

[2]:
```
save_folder_full = "/tmp/testmrio/full"
io.save_all(path=save_folder_full)
```

[2]:
```
<pymrio.core.mriosystem.IOSystem at 0x7fe06ba0e880>
```

To read again from that folder do:

[3]:
```
io_read = pymrio.load_all(path=save_folder_full)
```

The fileio activities are stored in the included meta data history field:

[4]:
```
io_read.meta
```

[4]:
```
Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /tmp/testmrio/full/metadata.json
History:
20210224 11:32:06 - FILEIO -  Added satellite account from /tmp/testmrio/full/
```

(continues on next page)

(continued from previous page)

```
↪factor_inputs
20210224 11:32:06 - FILEIO -  Added satellite account from /tmp/testmrio/full/
↪emissions
20210224 11:32:06 - FILEIO -  Loaded IO system from /tmp/testmrio/full
20210224 11:32:06 - FILEIO -  Saved testmrio to /tmp/testmrio/full
20210224 11:32:06 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
20210224 11:32:06 - MODIFICATION -  Calculating accounts for extension factor_
↪inputs
20210224 11:32:06 - MODIFICATION -  Leontief matrix L calculated
20210224 11:32:06 - MODIFICATION -  Coefficient matrix A calculated
20210224 11:32:06 - MODIFICATION -  Industry output x calculated
20210224 11:32:06 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/
↪pymrio/mrio_models/test_mrio
 ... (more lines in history)
```

6.8.2 Storage format

Internally, pymrio stores data in csv format, with the 'economic core' data in the root and each satellite account in a subfolder. Metadata as file as a file describing the data format ('file_parameters.json') are included in each folder.

[5]:
```
import os

os.listdir(save_folder_full)
```

[5]:
```
['emissions',
 'factor_inputs',
 'metadata.json',
 'file_parameters.json',
 'population.txt',
 'unit.txt',
 'L.txt',
 'A.txt',
 'x.txt',
 'Y.txt',
 'Z.txt']
```

The file format for storing the MRIO data can be switched to a binary pickle format with:

[6]:
```
save_folder_bin = "/tmp/testmrio/binary"
io.save_all(path=save_folder_bin, table_format="pkl")
os.listdir(save_folder_bin)
```

[6]:
```
['emissions',
 'factor_inputs',
 'metadata.json',
 'file_parameters.json',
 'population.pkl',
 'unit.pkl',
```

(continues on next page)

(continued from previous page)

```
 'L.pkl',
 'A.pkl',
 'x.pkl',
 'Y.pkl',
 'Z.pkl']
```

This can be used to reduce the storage space required on the disk for large MRIO databases.

6.8.3 Archiving MRIOs databases

To archive a MRIO system after saving use pymrio.archive:

[7]:
```
mrio_arc = "/tmp/testmrio/archive.zip"

# Remove a potentially existing archive from before
try:
    os.remove(mrio_arc)
except FileNotFoundError:
    pass

pymrio.archive(source=save_folder_full, archive=mrio_arc)
```

Data can be read directly from such an archive by:

[8]:
```
tt = pymrio.load_all(mrio_arc)
```

Currently data can not be saved directly into a zip archive. It is, however, possible to remove the source files after archiving:

[9]:
```
tmp_save = "/tmp/testmrio/tmp"

# Remove a potentially existing archive from before
try:
    os.remove(mrio_arc)
except FileNotFoundError:
    pass

io.save_all(tmp_save)

print("Directories before archiving: {}".format(os.listdir("/tmp/testmrio")))
pymrio.archive(source=tmp_save, archive=mrio_arc, remove_source=True)
print("Directories after archiving: {}".format(os.listdir("/tmp/testmrio")))
```

```
Directories before archiving: ['tmp', 'emission_footprints.xlsx', 'emissions',
↪ 'binary', 'full']
Directories after archiving: ['archive.zip', 'emission_footprints.xlsx',
↪'emissions', 'binary', 'full']
```

Several MRIO databases can be stored in the same archive:

[10]:
```
# Remove a potentially existing archive from before
try:
    os.remove(mrio_arc)
except FileNotFoundError:
    pass

tmp_save = "/tmp/testmrio/tmp"

io.save_all(tmp_save)
pymrio.archive(
    source=tmp_save, archive=mrio_arc, path_in_arc="version1/", remove_
↪source=True
)
io2 = io.copy()
del io2.emissions
io2.save_all(tmp_save)
pymrio.archive(
    source=tmp_save, archive=mrio_arc, path_in_arc="version2/", remove_
↪source=True
)
```

When loading from an archive which includes multiple MRIO databases, specify one with the parameter 'path_in_arc':

[11]:
```
io1_load = pymrio.load_all(mrio_arc, path_in_arc="version1/")
io2_load = pymrio.load_all(mrio_arc, path_in_arc="version2/")

print(
    "Extensions of the loaded io1 {ver1} and of io2: {ver2}".format(
        ver1=sorted(io1_load.get_extensions()), ver2=sorted(io2_load.get_
↪extensions())
    )
)
```

```
Extensions of the loaded io1 ['emissions', 'factor_inputs'] and of io2: [
↪'factor_inputs']
```

The pymrio.load function can be used directly to only a specific satellite account of a MRIO database from a zip archive:

[12]:
```
emissions = pymrio.load(mrio_arc, path_in_arc="version1/emissions")
print(emissions)
```

```
Extension Emissions with parameters: name, F, F_Y, S, S_Y, M, D_cba, D_pba, D_
↪imp, D_exp, unit, D_cba_reg, D_pba_reg, D_imp_reg, D_exp_reg, D_cba_cap, D_
↪pba_cap, D_imp_cap, D_exp_cap
```

The archive function is a wrapper around python.zipfile module. There are, however, some differences to the defaults choosen in the original:

- In contrast to zipfile.write, pymrio.archive raises an error if the data (path + filename) are identical in the zip archive. Background: the zip standard allows that files with the same name and path are stored side by side in a zip file. This becomes an issue when unpacking this files as they overwrite

each other upon extraction.

- The standard for the parameter 'compression' is set to ZIP_DEFLATED This is different from the zipfile default (ZIP_STORED) which would not give any compression. See the zipfile docs for further information. Depending on the value given for the parameter 'compression' additional modules might be necessary (e.g. zlib for ZIP_DEFLATED). Futher information on this can also be found in the zipfile python docs.

6.8.4 Storing or exporting a specific table or extension

Each extension of the MRIO system can be stored separetly with:

[13]:
```
save_folder_em = "/tmp/testmrio/emissions"
```

[14]:
```
io.emissions.save(path=save_folder_em)
```

[14]:
```
<pymrio.core.mriosystem.Extension at 0x7fe06ba19fa0>
```

This can then be loaded again as separate satellite account:

[15]:
```
emissions = pymrio.load(save_folder_em)
```

[16]:
```
emissions
```

[16]:
```
<pymrio.core.mriosystem.Extension at 0x7fe0698db130>
```

[17]:
```
emissions.D_cba
```

[17]:
```
region                              reg1                                \
sector                              food         mining manufactoring
stressor       compartment
emission_type1 air          2.056183e+06  179423.535893  9.749300e+07
emission_type2 water        2.423103e+05   25278.192086  1.671240e+07

region                                                                  \
sector                       electricity  construction         trade
stressor       compartment
emission_type1 air          1.188759e+07  3.342906e+06  3.885884e+06
emission_type2 water        1.371303e+05  3.468292e+05  7.766205e+05

region                                                          reg2  \
sector                         transport         other          food
stressor       compartment
emission_type1 air          1.075027e+07  1.582152e+07  1.793338e+06
emission_type2 water        4.999628e+05  8.480505e+06  2.136528e+05

region                                  ...          reg5                \
sector                           mining ...     transport         other
stressor       compartment              ...
emission_type1 air          19145.604911 ...  4.209505e+07  1.138661e+07
emission_type2 water         3733.601474 ...  4.243738e+06  7.307208e+06
```

(continues on next page)

(continued from previous page)

```
region                              reg6                                    \
sector                              food        mining manufactoring
stressor       compartment
emission_type1 air          1.517235e+07  1.345318e+06  7.145075e+07
emission_type2 water        4.420574e+06  5.372216e+05  1.068144e+07

region                                                                      \
sector                       electricity  construction         trade
stressor       compartment
emission_type1 air          3.683167e+07  1.836696e+06  4.241568e+07
emission_type2 water        5.728136e+05  9.069515e+05  5.449044e+07

region
sector                          transport         other
stressor       compartment
emission_type1 air          4.805409e+07  3.602298e+07
emission_type2 water        8.836484e+06  4.634899e+07

[2 rows x 48 columns]
```

As all data in pymrio is stored as pandas DataFrame, the full pandas stack for exporting tables is available. For example, to export a table as excel sheet use:

```
[18]: io.emissions.D_cba.to_excel("/tmp/testmrio/emission_footprints.xlsx")
```

For further information see the pandas documentation on import/export.

6.9 Adjusting, Renaming and Restructuring

```
[2]: import pymrio
```

```
[3]: mrio = pymrio.load_test()
```

We can use several functions to get a quick overview over the MRIO system:

```
[4]: mrio.get_sectors()
```

```
[4]: Index(['food', 'mining', 'manufactoring', 'electricity', 'construction',
       'trade', 'transport', 'other'],
      dtype='object', name='sector')
```

```
[5]: mrio.get_regions()
```

```
[5]: Index(['reg1', 'reg2', 'reg3', 'reg4', 'reg5', 'reg6'], dtype='object', name=
↪'region')
```

```
[6]: mrio.get_Y_categories()
```

```
[6]: Index(['Final consumption expenditure by households',
             'Final consumption expenditure by non-profit organisations serving␣
     ↪households (NPISH)',
             'Final consumption expenditure by government',
             'Gross fixed capital formation', 'Changes in inventories',
             'Changes in valuables', 'Export'],
           dtype='object', name='category')
```

A list of available satellite accounts can be obtained by

```
[7]: mrio.get_extensions()
[7]: <generator object IOSystem.get_extensions at 0x7f36f120c4a0>
```

this returns a generator over all extensions, to just get the names which can be used to loop over the extensions. To just get the names:

```
[8]: list(mrio.get_extensions())
[8]: ['factor_inputs', 'emissions']
```

All the names returned are literally just names and can easily be renamed. To rename sectors or regions just pass a dictionary with the new names:

```
[9]: mrio.rename_regions({"reg1": "REGION A", "reg2": "REGION B"})
[9]: <pymrio.core.mriosystem.IOSystem at 0x7f36f11de940>
```

```
[10]: mrio.get_regions()
[10]: Index(['REGION A', 'REGION B', 'reg3', 'reg4', 'reg5', 'reg6'], dtype='object
      ↪', name='region')
```

Renaming sectors or Y categories works the same way

```
[11]: mrio.rename_sectors({"mining": "dwarf business"})
      mrio.rename_Y_categories({"Final consumption expenditure by households": "fin_
      ↪house"})
[11]: <pymrio.core.mriosystem.IOSystem at 0x7f36f11de940>
```

```
[12]: mrio.get_sectors()
      mrio.get_Y_categories()
[12]: Index(['fin_house',
             'Final consumption expenditure by non-profit organisations serving␣
     ↪households (NPISH)',
             'Final consumption expenditure by government',
             'Gross fixed capital formation', 'Changes in inventories',
             'Changes in valuables', 'Export'],
           dtype='object', name='category')
```

Some MRIOs come with a selections of names which can be used for renaming. These can be obtained via 'get_classification', passing the name of the IO (pass None to get a list of available classifications).

[13]:
```
mrio_class = pymrio.get_classification(mrio_name="test")
```

The classification data contains different names and aggregation levels for the sectors and final demand categories. The easiest way to explore the classification data is by using the autocomplete functionality. Depending on the editor your are using this might work with typing mrio_class.sectors. pressing Tab or Ctrl-Space. The same works for mrio_class.finaldemand.

Alternatively it is possible to just print the underlying dataframes with (similar for finaldemand)

[14]:
```
mrio_class.sectors
```

[14]:

	TestMrioNumber	TestMrioName	TestMrioCode	Type
0	1	food	sec1	eat
1	2	mining	sec2	dig
2	3	manufactoring	sec3	build
3	4	electricity	sec4	util
4	5	construction	sec5	build
5	6	trade	sec6	marg
6	7	transport	sec7	marg
7	8	other	sec8	misc

This can be used to generate dictionaries for renaming the sectors of the MRIO, eg with:

[19]:
```
conv_dict = mrio_class.get_sector_dict(
    mrio_class.sectors.TestMrioName, mrio_class.sectors.TestMrioCode
)
conv_dict
```

[19]:
```
{'food': 'sec1',
 'mining': 'sec2',
 'manufactoring': 'sec3',
 'electricity': 'sec4',
 'construction': 'sec5',
 'trade': 'sec6',
 'transport': 'sec7',
 'other': 'sec8'}
```

This can then be used for renaming the sectors:

[16]:
```
mrio = pymrio.load_test()
```

[21]:
```
mrio.rename_sectors(conv_dict)
mrio.get_sectors()
```

[21]:
```
Index(['sec1', 'sec2', 'sec3', 'sec4', 'sec5', 'sec6', 'sec7', 'sec8'], dtype=
↪'object', name='sector')
```

In the mrio_class.sectors you will also find an entry 'Type' which represents a many to one mapping. This can be used for aggregating the MRIO and is explained *in the aggregation tutorial*.

[ ]:
```

```

6.10 Using the aggregation functionality of pymrio

Pymrio offers various possibilities to achieve an aggregation of a existing MRIO system. The following section will present all of them in turn, using the test MRIO system included in pymrio. The same concept can be applied to real life MRIOs.

Some of the examples rely in the country converter coco. The minimum version required is coco >= 0.6.3 - install the latest version with

```
pip install country_converter --upgrade
```

Coco can also be installed from the Anaconda Cloud - see the coco readme for further infos.

6.10.1 Loading the test mrio

First, we load and explore the test MRIO included in pymrio:

[1]:
```
import numpy as np
import pymrio
```

[2]:
```
io = pymrio.load_test()
io.calc_all()
```

[2]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e90c8beb0>
```

[3]:
```
print(
    "Sectors: {sec},\nRegions: {reg}".format(
        sec=io.get_sectors().tolist(), reg=io.get_regions().tolist()
    )
)
```

```
Sectors: ['food', 'mining', 'manufactoring', 'electricity', 'construction',
↪'trade', 'transport', 'other'],
Regions: ['reg1', 'reg2', 'reg3', 'reg4', 'reg5', 'reg6']
```

6.10.2 Aggregation using a numerical concordance matrix

This is the standard way to aggregate MRIOs when you work in Matlab. To do so, we need to set up a concordance matrix in which the columns correspond to the orignal classification and the rows to the aggregated one.

[4]:
```
sec_agg_matrix = np.array(
    [[1, 0, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 1,
↪ 1]]
)

reg_agg_matrix = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
```

[5]:
```
io.aggregate(region_agg=reg_agg_matrix, sector_agg=sec_agg_matrix)
```

[5]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e90c8beb0>
```

[6]:
```
print(
    "Sectors: {sec},\nRegions: {reg}".format(
        sec=io.get_sectors().tolist(), reg=io.get_regions().tolist()
    )
)
```

```
Sectors: ['sec0', 'sec1', 'sec2'],
Regions: ['reg0', 'reg1']
```

[7]:
```
io.calc_all()
```

[7]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e90c8beb0>
```

[8]:
```
io.emissions.D_cba
```

[8]:
```
region                              reg0                              \
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          9.041149e+06  3.018791e+08  1.523236e+08
emission_type2 water        2.123543e+06  4.884509e+07  9.889757e+07

region                              reg1
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          2.469465e+07  3.468742e+08  2.454117e+08
emission_type2 water        6.000239e+06  4.594530e+07  1.892731e+08
```

To use custom names for the aggregated sectors or regions, pass a list of names in order of rows in the concordance matrix:

[9]:
```
io = (
    pymrio.load_test()
    .calc_all()
    .aggregate(
        region_agg=reg_agg_matrix,
        region_names=["World Region A", "World Region B"],
        inplace=False,
    )
)
```

[10]:
```
io.get_regions()
```

[10]:
```
Index(['World Region A', 'World Region B'], dtype='object', name='region')
```

6.10.3 Aggregation using a numerical vector

Pymrio also accepts the aggregatio information as numerical or string vector. For these, each entry in the vector assignes the sector/region to a aggregation group. Thus the two aggregation matrices from above (*sec_agg_matrix* and *reg_agg_matrix*) can also be represented as numerical or string vectors/lists:

[11]:
```
sec_agg_vec = np.array([0, 1, 1, 1, 1, 2, 2, 2])
reg_agg_vec = ["R1", "R1", "R1", "R2", "R2", "R2"]
```

can also be represented as aggregation vector:

[12]:
```
io_vec_agg = (
    pymrio.load_test()
    .calc_all()
    .aggregate(region_agg=reg_agg_vec, sector_agg=sec_agg_vec, inplace=False)
)
```

[13]:
```
print(
    "Sectors: {sec},\nRegions: {reg}".format(
        sec=io_vec_agg.get_sectors().tolist(), reg=io_vec_agg.get_regions().
↪tolist()
    )
)
```

```
Sectors: ['sec0', 'sec1', 'sec2'],
Regions: ['R1', 'R2']
```

[14]:
```
io_vec_agg.emissions.D_cba_reg
```

[14]:
```
region                                R1            R2
stressor       compartment
emission_type1 air          6.690192e+08  1.686954e+09
emission_type2 water        5.337682e+08  5.902081e+08
```

6.10.4 Regional aggregation using the country converter coco

The previous examples are best suited if you want to reuse existing aggregation information. For new/ad hoc aggregation, the most user-friendly solution is to build the concordance with the country converter coco. The minimum version of coco required is 0.6.2. You can either use coco to build independent aggregations (first case below) or use the predefined classifications included in coco (second case - Example WIOD below).

[15]:
```
import country_converter as coco
```

Independent aggregation

[16]:
```
io = pymrio.load_test().calc_all()
```

[17]:
```
reg_agg_coco = coco.agg_conc(
    original_countries=io.get_regions(),
    aggregates={
        "reg1": "World Region A",
        "reg2": "World Region A",
        "reg3": "World Region A",
    },
    missing_countries="World Region B",
)
```

[18]:
```
io.aggregate(region_agg=reg_agg_coco)
```

[18]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e8eb6aa60>
```

[19]:
```
print(
    "Sectors: {sec},\nRegions: {reg}".format(
        sec=io.get_sectors().tolist(), reg=io.get_regions().tolist()
    )
)
```

```
Sectors: ['food', 'mining', 'manufactoring', 'electricity', 'construction',
↪'trade', 'transport', 'other'],
Regions: ['World Region A', 'World Region B']
```

This can be passed directly to pymrio:

[20]:
```
io.emissions.D_cba_reg
```

[20]:
```
region                       World Region A  World Region B
stressor       compartment
emission_type1 air              6.690192e+08    1.686954e+09
emission_type2 water            5.337682e+08    5.902081e+08
```

A pandas DataFrame corresponding to the output from *coco* can also be passed to *sector_agg* for aggregation. A sector aggregation package similar to the country converter is planned.

Using the build-in classifications - WIOD example

The country converter is most useful when you work with a MRIO which is included in coco. In that case you can just pass the desired country aggregation to coco and it returns the required aggregation matrix:

For the example here, we assume that a raw WIOD download is available at:

[21]:
```
wiod_raw = "/tmp/mrios/WIOD2013"
```

We will parse the year 2000 and calculate the results:

[22]:
```
wiod_orig = pymrio.parse_wiod(path=wiod_raw, year=2000).calc_all()
```

and then aggregate the database to first the EU countries and group the remaining countries based on OECD membership. In the example below, we single out Germany (DEU) to be not included in the aggregation:

[23]:
```
wiod_agg_DEU_EU_OECD = wiod_orig.aggregate(
    region_agg=coco.agg_conc(
        original_countries="WIOD",
        aggregates=[{"DEU": "DEU"}, "EU", "OECD"],
        missing_countries="Other",
        merge_multiple_string=None,
    ),
    inplace=False,
)
```

We can then rename the regions to make the membership clearer:

[24]:
```
wiod_agg_DEU_EU_OECD.rename_regions({"OECD": "OECDwoEU", "EU": "EUwoGermany"})
```

[24]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f0e8a1e9be0>
```

To see the result for the air emission footprints:

[25]:
```
wiod_agg_DEU_EU_OECD.AIR.D_cba_reg
```

[25]:

region	OECDwoEU	EUwoGermany	Other	DEU
stressor				
CO2	1.029626e+07	3.120346e+06	9.232742e+06	1.123772e+06
CH4	6.595964e+07	2.605212e+07	1.487615e+08	7.953304e+06
N2O	2.312456e+06	1.055048e+06	6.166586e+06	2.941486e+05
NOX	3.986117e+07	1.130268e+07	5.103133e+07	3.164278e+06
SOX	3.567011e+07	1.034605e+07	5.137882e+07	2.045926e+06
CO	2.032219e+08	4.789266e+07	4.424992e+08	1.296816e+07
NMVOC	3.680383e+07	1.280083e+07	8.186918e+07	2.870176e+06
NH3	6.013446e+06	3.307710e+06	1.674807e+07	8.656818e+05

For further examples on the capabilities of the country converter see the coco tutorial notebook

6.10.5 Aggregation by renaming

One alternative method for aggregating the MRIO system is to rename specific regions and/or sectors to duplicated names. Duplicated sectors and regions can then be automatically aggregated. This makes most sense when having some categories of some kind (e.g. consumption categories) or detailed classification which can easily be broadened (e.g. A01, A02, which could be renamed all to A). In the example below, we will aggregate sectors to consumption categories using some predefined categories included in pymrio. Check the *Adjusting, Renaming and Restructuring notebook for more details.*

[ ]:
```
mrio = pymrio.load_test()
```

[ ]:
```
class_info = pymrio.get_classification("test")
rename_dict = class_info.get_sector_dict(
    orig=class_info.sectors.TestMrioName, new=class_info.sectors.Type
)
```

If we take a look at the rename_dict, we see that it maps several sectors of the original MRIO to combined regions (technically a many to one mapping).

[ ]:
```
rename_dict
```

Using this dict to rename sectors leads to an index with overlapping labels.

[ ]:
```
mrio.rename_sectors(rename_dict)
mrio.Z
```

Which can then be aggregated with

[ ]:
```
mrio.aggregate_duplicates()
mrio.Z
```

This method also comes handy when aggregating parts of the MRIO regions. E.g.:

[ ]:
```
region_convert = {"reg1": "Antarctica", "reg2": "Antarctica"}
mrio.rename_regions(region_convert).aggregate_duplicates()
mrio.Z
```

Which lets us calculate the footprint of the consumption category 'eat' in 'Antarctica':

[ ]:
```
mrio.calc_all()
mrio.emissions.D_cba.loc[:, ("Antarctica", "eat")]
```

6.10.6 Aggregation to one total sector / region

Both, *region_agg* and *sector_agg*, also accept a string as argument. This leads to the aggregation to one total region or sector for the full IO system.

[26]:
```
pymrio.load_test().calc_all().aggregate(
    region_agg="global", sector_agg="total"
).emissions.D_cba
```

[26]:
```
region                            global
sector                             total
stressor       compartment
emission_type1 air          1.080224e+09
emission_type2 water        3.910848e+08
```

6.10.7 Pre- vs post-aggregation account calculations

It is generally recommended to calculate MRIO accounts with the highest detail possible and aggregated the results afterwards (post-aggregation - see for example Steen-Olsen et al 2014, Stadler et al 2014 or Koning et al 2015.

Pre-aggregation, that means the aggregation of MRIO sectors and regions before calculation of footprint accounts, might be necessary when dealing with MRIOs on computers with limited RAM resources. However, one should be aware that the results might change.

Pymrio can handle both cases and can be used to highlight the differences. To do so, we use the two concordance matrices defined at the beginning (*sec_agg_matrix* and *reg_agg_matrix*) and aggregate the test system before and after the calculation of the accounts:

[27]:
```
io_pre = (
    pymrio.load_test()
    .aggregate(region_agg=reg_agg_matrix, sector_agg=sec_agg_matrix)
    .calc_all()
)
io_post = (
    pymrio.load_test()
    .calc_all()
    .aggregate(region_agg=reg_agg_matrix, sector_agg=sec_agg_matrix)
)
```

[28]:
```
io_pre.emissions.D_cba
```

[28]:
```
region                              reg0                              \
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          7.722782e+06  3.494413e+08  1.388764e+08
emission_type2 water        1.862161e+06  5.240950e+07  1.583465e+08

region                              reg1
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          2.695396e+07  3.354598e+08  2.217703e+08
emission_type2 water        6.399685e+06  4.080509e+07  1.312619e+08
```

[29]:
```
io_post.emissions.D_cba
```

[29]:
```
region                              reg0                              \
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          9.041149e+06  3.018791e+08  1.523236e+08
emission_type2 water        2.123543e+06  4.884509e+07  9.889757e+07

region                              reg1
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          2.469465e+07  3.468742e+08  2.454117e+08
emission_type2 water        6.000239e+06  4.594530e+07  1.892731e+08
```

The same results as in io_pre are obtained for io_post, if we recalculate the footprint accounts based on the aggregated system:

[30]:
```
io_post.reset_all_full().calc_all().emissions.D_cba
```

[30]:
```
region                              reg0                                \
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          7.722782e+06  3.494413e+08  1.388764e+08
emission_type2 water        1.862161e+06  5.240950e+07  1.583465e+08

region                              reg1
sector                              sec0          sec1          sec2
stressor       compartment
emission_type1 air          2.695396e+07  3.354598e+08  2.217703e+08
emission_type2 water        6.399685e+06  4.080509e+07  1.312619e+08
```

6.11 Analysing the source of stressors (flow matrix)

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 section shows how to do this based on the small test mrio included in pymrio. The same procedure can be use for any other MRIO, but keep in mind that diagonalizing a stressor dramatically increases the memory need for the calculations.

6.11.1 Basic example

First we load the test mrio:

[1]:
```
import pymrio

io = pymrio.load_test()
```

The test mrio includes several extensions:

[2]:
```
list(io.get_extensions())
```

[2]:
```
['factor_inputs', 'emissions']
```

For the example here, we use 'emissions' - 'emission_type1':

[3]:
```
io.emissions.F
```

[3]:
```
region                          reg1                                          ␣
↪ \
sector                          food      mining manufactoring  electricity
stressor       compartment
emission_type1 air          1848064.80  986448.090   23613787.00  28139100.00
emission_type2 water         139250.47   22343.295     763569.18    273981.55

region                                                                    \
sector                   construction      trade   transport      other
```

(continues on next page)

(continued from previous page)

```
stressor       compartment
emission_type1 air           2584141.80  4132656.3  21766987.0  7842090.6
emission_type2 water          317396.51  1254477.8   1012999.1  2449178.0

region                            reg2              ...      reg5           ␣
↪ \
sector                            food      mining  ... transport       other
stressor       compartment                          ...
emission_type1 air           1697937.30  347378.150  ...  42299319  10773826.0
emission_type2 water          204835.44   29463.944  ...   4199841   7191006.3

region                            reg6                                        \
sector                            food     mining manufactoring electricity
stressor       compartment
emission_type1 air           15777996.0  6420955.5   113172450.0  56022534.0
emission_type2 water          4826108.1  1865625.1    12700193.0    753213.7

region
sector                     construction     trade   transport     other
stressor       compartment
emission_type1 air            4861838.5  18195621  47046542.0  21632868
emission_type2 water          2699288.3  13892313   8765784.3  16782553

[2 rows x 48 columns]
```

[4]:
```
et1_diag = io.emissions.diag_stressor(("emission_type1", "air"), name=
↪"emtype1_diag")
```

The parameter name is optional, if not given the name is set to the stressor name + '_diag'

The new emission matrix now looks like this:

[5]:
```
et1_diag.F.head(15)
```

[5]:
```
region                    reg1                                        \
sector                    food     mining manufactoring electricity
region sector
reg1   food           1848064.8       0.00           0.0         0.0
       mining                0.0  986448.09           0.0         0.0
       manufactoring         0.0       0.00    23613787.0         0.0
       electricity           0.0       0.00           0.0  28139100.0
       construction          0.0       0.00           0.0         0.0
       trade                 0.0       0.00           0.0         0.0
       transport             0.0       0.00           0.0         0.0
       other                 0.0       0.00           0.0         0.0
reg2   food                  0.0       0.00           0.0         0.0
       mining                0.0       0.00           0.0         0.0
       manufactoring         0.0       0.00           0.0         0.0
       electricity           0.0       0.00           0.0         0.0
       construction          0.0       0.00           0.0         0.0
```

(continues on next page)

(continued from previous page)

```
       trade                0.0       0.00           0.0         0.0
       transport            0.0       0.00           0.0         0.0

region                                                                \
sector              construction      trade   transport      other
region sector
reg1   food                  0.0        0.0         0.0        0.0
       mining                0.0        0.0         0.0        0.0
       manufactoring         0.0        0.0         0.0        0.0
       electricity           0.0        0.0         0.0        0.0
       construction     2584141.8        0.0         0.0        0.0
       trade                 0.0  4132656.3         0.0        0.0
       transport             0.0        0.0  21766987.0        0.0
       other                 0.0        0.0         0.0  7842090.6
reg2   food                  0.0        0.0         0.0        0.0
       mining                0.0        0.0         0.0        0.0
       manufactoring         0.0        0.0         0.0        0.0
       electricity           0.0        0.0         0.0        0.0
       construction          0.0        0.0         0.0        0.0
       trade                 0.0        0.0         0.0        0.0
       transport             0.0        0.0         0.0        0.0

region                     reg2             ...      reg5       reg6        \
sector                     food     mining  ... transport other food mining
region sector                               ...
reg1   food                 0.0       0.00  ...       0.0   0.0  0.0    0.0
       mining               0.0       0.00  ...       0.0   0.0  0.0    0.0
       manufactoring        0.0       0.00  ...       0.0   0.0  0.0    0.0
       electricity          0.0       0.00  ...       0.0   0.0  0.0    0.0
       construction         0.0       0.00  ...       0.0   0.0  0.0    0.0
       trade                0.0       0.00  ...       0.0   0.0  0.0    0.0
       transport            0.0       0.00  ...       0.0   0.0  0.0    0.0
       other                0.0       0.00  ...       0.0   0.0  0.0    0.0
reg2   food           1697937.3       0.00  ...       0.0   0.0  0.0    0.0
       mining               0.0  347378.15  ...       0.0   0.0  0.0    0.0
       manufactoring        0.0       0.00  ...       0.0   0.0  0.0    0.0
       electricity          0.0       0.00  ...       0.0   0.0  0.0    0.0
       construction         0.0       0.00  ...       0.0   0.0  0.0    0.0
       trade                0.0       0.00  ...       0.0   0.0  0.0    0.0
       transport            0.0       0.00  ...       0.0   0.0  0.0    0.0

region                                                                  \
sector              manufactoring electricity construction trade transport
region sector
reg1   food                   0.0         0.0          0.0   0.0       0.0
       mining                 0.0         0.0          0.0   0.0       0.0
       manufactoring          0.0         0.0          0.0   0.0       0.0
       electricity            0.0         0.0          0.0   0.0       0.0
       construction           0.0         0.0          0.0   0.0       0.0
```

(continues on next page)

(continued from previous page)

```
       trade                0.0          0.0          0.0   0.0          0.0
       transport            0.0          0.0          0.0   0.0          0.0
       other                0.0          0.0          0.0   0.0          0.0
reg2   food                 0.0          0.0          0.0   0.0          0.0
       mining               0.0          0.0          0.0   0.0          0.0
       manufactoring        0.0          0.0          0.0   0.0          0.0
       electricity          0.0          0.0          0.0   0.0          0.0
       construction         0.0          0.0          0.0   0.0          0.0
       trade                0.0          0.0          0.0   0.0          0.0
       transport            0.0          0.0          0.0   0.0          0.0

region
sector                other
region sector
reg1   food             0.0
       mining           0.0
       manufactoring    0.0
       electricity      0.0
       construction     0.0
       trade            0.0
       transport        0.0
       other            0.0
reg2   food             0.0
       mining           0.0
       manufactoring    0.0
       electricity      0.0
       construction     0.0
       trade            0.0
       transport        0.0

[15 rows x 48 columns]
```

And can be connected back to the system with:

[6]:
```
io.et1_diag = et1_diag
```

Finally we can calulate the all stressor accounts with:

[7]:
```
io.calc_all()
```

[7]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f9c25d25d30>
```

This results in a square footprint matrix. In this matrix, every column respresents the amount of stressor occuring in each region - sector driven by the consumption stated in the column header. Conversly, each row states where the stressor impacts occuring in the row are distributed due (from where they are driven).

[8]:
```
io.et1_diag.D_cba.head(20)
```

[8]:
```
region                        reg1                                          ␣
↪ \
sector                        food        mining manufactoring   electricity
```

(continues on next page)

(continued from previous page)

```
region sector
reg1   food           609347.998747     34.963223  1.987631e+05  7.678755e+02
       mining           2527.449441  61271.249639  1.232716e+05  5.406781e+04
       manufactoring    1199.041530     38.236679  4.686837e+06  8.108636e+02
       electricity    148505.091902  12764.784297  1.519466e+06  1.167804e+07
       construction       49.018479      6.053459  4.019302e+02  3.081457e+02
       trade             138.041355      3.139596  1.880042e+03  8.852940e+01
       transport         521.216924    122.504968  1.585636e+04  1.428149e+03
       other             537.062679     24.688020  7.752597e+03  1.170368e+03
reg2   food              234.870108      0.041282  1.213308e+03  3.241119e+00
       mining            215.562762   1690.186670  5.892279e+04  1.862733e+03
       manufactoring      75.621346      4.229463  7.185296e+06  6.763336e+01
       electricity         4.912183      4.392270  6.448813e+03  1.947867e+02
       construction        0.179274      0.201490  4.711030e+02  8.221412e-01
       trade               9.487802      0.442851  2.547850e+03  5.931388e+00
       transport          30.167917     11.691703  1.095783e+04  7.704484e+01
       other               4.710152      0.999656  3.487999e+03  1.504331e+01
reg3   food               79.487995      0.012420  2.707179e+03  3.212251e-01
       mining              1.660826      9.283144  2.805174e+03  4.683637e+00
       manufactoring     256.950787     12.496966  1.951101e+07  1.576456e+02
       electricity       346.618944     75.166170  3.907669e+06  8.377082e+03

region                                                                    ␣
↪\
sector                construction         trade     transport         other
region sector
reg1   food           2.873371e+03  5.603158e+04  1.448778e+03  5.225312e+04
       mining         1.632967e+05  1.459774e+04  4.916876e+03  4.975184e+04
       manufactoring  1.816229e+04  7.713610e+03  2.825229e+03  1.634290e+04
       electricity    5.265922e+05  1.307806e+06  4.851957e+05  4.308489e+06
       construction   2.568908e+06  7.291250e+02  5.853107e+02  9.718253e+03
       trade          1.420349e+03  2.390871e+06  4.371531e+02  2.383210e+03
       transport      6.467383e+03  2.741408e+04  1.018383e+07  4.388313e+04
       other          8.587431e+03  1.322798e+04  4.448616e+03  7.516913e+06
reg2   food           9.719899e+00  1.004924e+01  4.822982e-01  1.281313e+01
       mining         7.746514e+02  7.570607e+02  1.582065e+02  2.302825e+03
       manufactoring  7.171199e+02  4.429766e+02  1.914508e+02  1.076869e+03
       electricity    1.552407e+01  2.864877e+01  1.010699e+01  9.682693e+01
       construction   1.045043e+02  1.884894e+00  1.624084e+00  1.924495e+01
       trade          3.093165e+01  2.414790e+02  8.294245e+00  5.515915e+01
       transport      2.369344e+02  1.171461e+03  4.962901e+03  1.299130e+03
       other          6.607788e+01  8.958452e+01  3.266330e+01  1.104173e+03
reg3   food           1.118965e+00  8.545640e+00  3.848642e-01  2.342262e+01
       mining         3.721404e+00  1.903501e+01  1.984584e+00  1.130052e+02
       manufactoring  1.369020e+03  9.774252e+02  4.651207e+02  9.975075e+03
       electricity    1.955704e+03  3.618932e+03  1.930657e+03  5.954611e+05

region                        reg2                ...           reg5  \
sector                        food        mining  ...      transport
```

(continues on next page)

(continued from previous page)

```
region sector                                    ...
reg1   food           4.826852e+04      0.453097  ...      74.449012
       mining         4.311608e+02    288.903690  ...     261.326848
       manufactoring  3.205155e+02      2.886944  ...     424.008556
       electricity    1.908031e+04    240.730963  ...    9294.381551
       construction   4.408230e+00      0.037972  ...       1.679791
       trade          2.637857e+01      0.095853  ...      36.433641
       transport      9.410163e+01      2.099087  ...    2854.767892
       other          6.837072e+01      0.767113  ...      56.726997
reg2   food           1.694248e+06      0.041642  ...       3.026472
       mining         1.787512e+03  10287.905967  ...     200.678830
       manufactoring  9.941057e+02      5.361056  ...     352.701574
       electricity    1.143809e+03     23.763711  ...     225.020323
       construction   1.001309e+02      1.098256  ...     130.390027
       trade          3.073687e+02      1.313536  ...     159.340296
       transport      9.083313e+02     15.109529  ...  789093.398671
       other          2.621005e+02      3.670327  ...     376.984093
reg3   food           6.321926e+01      0.001120  ...       0.660378
       mining         1.043411e+00      1.482323  ...       1.579297
       manufactoring  2.274549e+02      2.093210  ...     856.521423
       electricity    9.697584e+02     33.309324  ...    1833.141385

region                                       reg6                              \
sector                       other          food       mining manufactoring
region sector
reg1   food             222.953289  25525.516392     6.557600  1.382548e+05
       mining           829.871069    268.406793  1990.840054  7.400739e+04
       manufactoring    517.344586    145.900439    72.195039  3.424703e+06
       electricity    28937.569016   7947.002552   476.106897  1.072302e+06
       construction       4.461035      2.211530     0.213951  2.898921e+02
       trade             50.392051     42.098945     1.141682  1.240785e+03
       transport        222.900909    568.393996    55.916912  1.130829e+04
       other            435.703359     36.640953     3.856648  5.346664e+03
reg2   food               0.475024     80.139206     0.025141  2.183449e+02
       mining            58.083367     66.185618  1351.809333  1.430140e+04
       manufactoring    125.165588     44.659838    19.157594  1.140699e+06
       electricity        2.614112      0.740525     3.162466  1.079151e+03
       construction       4.804088      0.230208     0.152699  7.708346e+01
       trade             69.421120     25.278297     0.783272  5.438996e+02
       transport        117.822465    101.020895     7.816896  2.491430e+03
       other            213.700696      7.178914     1.523151  6.818453e+02
reg3   food               1.095579     96.427089     0.060430  1.350698e+03
       mining             4.760043      1.028577     9.018513  1.370585e+03
       manufactoring   1696.838781    304.748914   109.539716  9.493385e+06
       electricity     6869.669139   1750.805701  1639.927136  1.908654e+06

region                                                                      \
sector                 electricity construction         trade    transport
region sector
```

(continues on next page)

(continued from previous page)

```
reg1   food              38.345126   408.937891  4.425472e+04    63.940215
       mining          6327.871035   299.916699  1.732632e+04   242.874444
       manufactoring     94.046696   129.127059  7.826193e+03   241.185186
       electricity    38647.049939   495.159380  9.712017e+05  3850.133826
       construction       1.682977    12.667370  5.303592e+02     0.988979
       trade              4.989788    14.192623  1.726377e+06    45.633464
       transport        285.596244   127.388007  2.198931e+04  8734.726973
       other             20.750945    15.152425  1.029493e+04    58.278076
reg2   food               0.472522     7.011110  1.605455e+02     0.455175
       mining          2650.008184    42.230072  8.909866e+03    79.981946
       manufactoring     19.220461    50.982784  3.009360e+03    74.837190
       electricity        8.601151     0.254639  2.214440e+03     1.186302
       construction       0.545433    47.750232  3.449526e+02     1.023520
       trade              2.795486     8.852273  1.174931e+06    20.099409
       transport         51.079756    28.609582  8.615321e+03  2234.585845
       other              7.576198     7.380601  3.072026e+03    37.695142
reg3   food               0.136524     0.253732  5.444486e+02     0.330214
       mining            19.208836     0.609243  7.625285e+02     1.234952
       manufactoring     71.802017   260.584083  6.010179e+04   587.491671
       electricity    44575.821242   628.061059  6.364065e+06  4888.472424

region
sector                        other
region sector
reg1   food           4.876200e+02
       mining         4.038056e+03
       manufactoring  9.015651e+02
       electricity    9.804940e+03
       construction   3.859558e+00
       trade          4.276151e+01
       transport      1.309008e+03
       other          1.356728e+03
reg2   food           3.092008e+00
       mining         3.551366e+03
       manufactoring  2.460570e+02
       electricity    1.102287e+01
       construction   5.208169e+00
       trade          2.543311e+01
       transport      4.125342e+02
       other          7.442119e+02
reg3   food           1.007920e+02
       mining         6.075141e+02
       manufactoring  3.817698e+04
       electricity    3.661473e+06

[20 rows x 48 columns]
```

The total footprints of a region - sector are given by summing the footprints along rows:

[9]:
```
io.et1_diag.D_cba.sum(axis=0).reg1
```

[9]:
```
sector
food              2.056183e+06
mining            1.794235e+05
manufactoring     9.749300e+07
electricity       1.188759e+07
construction      3.342906e+06
trade             3.885884e+06
transport         1.075027e+07
other             1.582152e+07
dtype: float64
```

[10]:
```
io.emissions.D_cba.reg1
```

[10]:
```
sector                              food         mining  manufactoring  \
stressor       compartment
emission_type1 air           2.056183e+06  179423.535893   9.749300e+07
emission_type2 water         2.423103e+05   25278.192086   1.671240e+07

sector                       electricity  construction         trade  \
stressor       compartment
emission_type1 air           1.188759e+07  3.342906e+06  3.885884e+06
emission_type2 water         1.371303e+05  3.468292e+05  7.766205e+05

sector                         transport         other
stressor       compartment
emission_type1 air           1.075027e+07  1.582152e+07
emission_type2 water         4.999628e+05  8.480505e+06
```

The total stressor in a sector corresponds to the sum of the columns:

[11]:
```
io.et1_diag.D_cba.sum(axis=1).reg1
```

[11]:
```
sector
food              1848064.80
mining             986448.09
manufactoring    23613787.00
electricity      28139100.00
construction      2584141.80
trade             4132656.30
transport        21766987.00
other             7842090.60
dtype: float64
```

[12]:
```
io.emissions.F.reg1
```

[12]:
```
sector                            food      mining  manufactoring  \
stressor       compartment
emission_type1 air           1848064.80  986448.090    23613787.00
emission_type2 water          139250.47   22343.295      763569.18
```

(continues on next page)

(continued from previous page)

```
sector                     electricity  construction      trade   transport ␣
↪\
stressor       compartment
emission_type1 air          28139100.00    2584141.80  4132656.3  21766987.0
emission_type2 water          273981.55     317396.51  1254477.8   1012999.1

sector                           other
stressor       compartment
emission_type1 air          7842090.6
emission_type2 water        2449178.0
```

6.11.2 Aggregation of source footprints

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 region of stressor, do:

[13]:
```
io.et1_diag.D_cba.groupby(level="region", axis=0).sum()
```

[13]:
```
region          reg1                                                      ␣
↪\
sector          food        mining manufactoring   electricity  construction
region
reg1    7.628249e+05  74265.619882  6.554229e+06  1.173668e+07  3.296308e+06
reg2    5.755115e+02   1712.185384  7.269345e+06  2.227236e+03  1.955463e+03
reg3    1.054578e+03    215.654591  2.382082e+07  9.500254e+03  7.044809e+03
reg4    1.147382e+03   2323.792084  1.219510e+07  3.931814e+03  6.717898e+03
reg5    1.283812e+06   6596.713907  1.588478e+07  1.642030e+04  1.234768e+04
reg6    6.769053e+03  94309.570045  3.176873e+07  1.188346e+05  1.853198e+04

region                                                    reg2              ␣
↪\
sector         trade     transport         other          food        mining
region
reg1    3.818391e+06  1.068369e+07  1.199973e+07  6.829377e+04    535.974719
reg2    2.743145e+03  5.365729e+03  5.967041e+03  1.699751e+06  10338.264024
reg3    1.285847e+04  2.265449e+04  2.844767e+06  1.740953e+03     54.246936
reg4    2.272241e+03  2.088695e+03  1.054927e+04  8.986664e+02   5812.809417
reg5    1.598377e+04  2.387931e+04  2.906402e+04  1.828518e+04   1121.255607
reg6    3.363451e+04  1.259087e+04  9.314423e+05  4.368062e+03   1283.054207

region  ...          reg5                        reg6                \
sector  ...     transport         other          food        mining
region  ...
reg1    ...  1.300377e+04  3.122120e+04  3.453617e+04  2.606829e+03
reg2    ...  7.905415e+05  5.920865e+02  3.254335e+02  1.384431e+03
reg3    ...  2.026441e+04  3.336623e+04  5.910854e+03  2.129253e+03
reg4    ...  1.818417e+03  1.930517e+03  2.402317e+03  1.257375e+06
```

(continues on next page)

(continued from previous page)

```
reg5    ...  4.126106e+07  1.129572e+07  4.599517e+04  7.759982e+03
reg6    ...  8.360619e+03  2.378125e+04  1.508319e+07  7.406276e+04

region                                                                 ␣
↪\
sector manufactoring   electricity  construction         trade     transport
region
reg1    4.727453e+06  4.542033e+04  1.502541e+03  2.799800e+06  1.323776e+04
reg2    1.160092e+06  2.740299e+03  1.930713e+02  1.201258e+06  2.449865e+03
reg3    1.160211e+07  4.640647e+04  2.506927e+03  2.295789e+07  4.824482e+04
reg4    8.856003e+06  7.878447e+03  2.121123e+03  6.469320e+06  5.546359e+03
reg5    1.519859e+07  1.879346e+04  8.076635e+03  7.701012e+06  4.094084e+04
reg6    2.990651e+07  3.671043e+07  1.822296e+06  1.286404e+06  4.794367e+07

region
sector         other
region
reg1    1.794454e+04
reg2    4.998925e+03
reg3    1.756144e+07
reg4    1.756339e+04
reg5    2.125918e+04
reg6    1.839977e+07

[6 rows x 48 columns]
```

In addition, the *aggregation function* of pymrio also work on the diagonalized footprints. Here as example together with the country converter coco:

[14]:
```
import country_converter as coco

io.aggregate(
    region_agg=coco.agg_conc(
        original_countries=io.get_regions(),
        aggregates={
            "reg1": "World Region A",
            "reg2": "World Region A",
            "reg3": "World Region A",
        },
        missing_countries="World Region B",
    )
)
```

[14]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f9c25d25d30>
```

[15]:
```
io.et1_diag.D_cba
```

[15]:
```
region                       World Region A                                \
sector                                 food        mining manufactoring
region         sector
```

(continues on next page)

(continued from previous page)

```
World Region A food            6.413682e+06  5.952471e+01  6.070321e+05
               mining          6.832129e+03  2.421509e+06  4.487266e+05
               manufactoring   1.575255e+04  4.337974e+03  5.857218e+07
               electricity     1.148908e+06  8.329886e+05  1.095357e+07
               construction    1.287094e+03  3.530581e+03  7.855368e+03
               trade           1.302177e+04  3.358650e+03  1.320568e+05
               transport       1.661673e+04  1.013917e+04  2.026243e+05
               other           4.973365e+04  5.119490e+04  4.157135e+05
World Region B food            1.331074e+06  5.840707e+01  1.158034e+06
               mining          1.120813e+04  1.223669e+05  6.244537e+06
               manufactoring   1.165415e+04  1.112145e+03  1.403366e+08
               electricity     6.624049e+03  2.433130e+04  5.205461e+06
               construction    2.441345e+03  3.338858e+02  6.498032e+04
               trade           7.186649e+02  1.172395e+02  8.340493e+04
               transport       1.118710e+04  1.051599e+03  3.382896e+05
               other           4.082899e+02  1.690002e+02  4.761138e+04

region                                                                  \
sector                          electricity  construction         trade
region         sector
World Region A food            9.326086e+02  3.306995e+03  5.987980e+04
               mining          1.936672e+05  1.984123e+05  2.126768e+04
               manufactoring   5.217160e+03  1.108166e+05  1.787260e+04
               electricity     5.960881e+07  1.529502e+06  1.771262e+06
               construction    4.364648e+03  1.082799e+07  1.811093e+03
               trade           5.581957e+03  7.080932e+04  4.827615e+06
               transport       1.186026e+04  9.463786e+04  6.605838e+04
               other           3.141636e+04  2.451423e+05  7.347058e+04
World Region B food            7.157090e+02  2.277605e+03  2.977910e+04
               mining          3.478494e+05  3.383509e+04  1.050007e+05
               manufactoring   4.253396e+03  1.644812e+04  4.518924e+04
               electricity     1.790907e+05  1.709844e+04  2.343087e+06
               construction    1.067936e+03  2.843476e+04  1.094280e+04
               trade           3.915868e+02  7.298241e+02  1.845264e+07
               transport       3.512873e+03  4.336422e+03  6.536434e+04
               other           5.086643e+02  7.504197e+02  2.293492e+04

region                                                  World Region B  \
sector                             transport         other           food
region         sector
World Region A food            3.514385e+03  5.762977e+04   3.437446e+04
               mining          1.676197e+04  7.595762e+04   4.373682e+02
               manufactoring   4.264953e+04  8.572782e+04   1.196817e+03
               electricity     3.062426e+06  9.812779e+06   1.717518e+04
               construction    1.193404e+04  4.796826e+04   8.117404e+00
               trade           3.359931e+04  6.189364e+04   1.772741e+03
               transport       6.900978e+07  2.981319e+05   3.570924e+03
               other           3.463245e+05  3.192544e+07   1.468825e+03
World Region B food            1.009061e+03  6.741954e+03   2.366136e+07
```

(continues on next page)

(continued from previous page)

```
               mining         3.107988e+04  1.118387e+05   3.852121e+04
               manufactoring  1.350120e+04  3.504348e+04   7.406372e+04
               electricity    1.540263e+04  8.184191e+05   7.886875e+05
               construction   2.562950e+03  1.372537e+04   1.229142e+04
               trade          1.527523e+03  1.037099e+04   1.802717e+04
               transport      3.522339e+06  2.451637e+04   3.616496e+04
               other          2.199852e+03  4.906621e+06   5.528324e+03

region                                                                  \
sector                              mining manufactoring   electricity
region         sector
World Region A food           7.789050e+00  3.648174e+05  6.373523e+01
               mining         3.805693e+03  2.419141e+05  1.079248e+04
               manufactoring  2.389005e+02  5.612155e+07  1.442696e+03
               electricity    2.384122e+03  1.167721e+07  3.179723e+05
               construction   1.278994e+00  9.064738e+03  2.977577e+01
               trade          3.095513e+01  1.747324e+05  5.736669e+02
               transport      2.342901e+02  2.223755e+05  2.064082e+03
               other          2.988605e+02  5.284564e+05  3.388346e+03
World Region B food           1.599984e+02  7.393735e+05  3.069128e+03
               mining         1.612039e+06  6.627586e+06  1.172268e+06
               manufactoring  3.204186e+03  1.254308e+08  3.173150e+04
               electricity    2.316855e+04  6.039318e+06  1.225545e+08
               construction   2.193007e+03  4.813644e+04  3.446390e+04
               trade          4.401980e+02  9.207769e+04  4.792922e+03
               transport      2.977747e+03  2.337329e+05  3.716876e+04
               other          8.635489e+02  4.987069e+04  8.685131e+03

region                                                                  \
sector                        construction         trade     transport
region         sector
World Region A food           1.339737e+03  4.505792e+04  1.557188e+02
               mining         1.073022e+04  2.708989e+04  8.090257e+02
               manufactoring  6.427781e+03  7.159629e+04  3.128223e+03
               electricity    1.151092e+04  7.346021e+06  2.360035e+04
               construction   1.183350e+03  1.057865e+04  1.704971e+02
               trade          2.618051e+03  1.843441e+07  4.414173e+03
               transport      4.978304e+03  3.583977e+05  8.792139e+05
               other          7.111777e+03  6.871237e+05  8.921976e+03
World Region B food           2.404678e+04  1.717301e+05  3.012820e+04
               mining         5.475534e+05  1.556577e+05  8.352660e+04
               manufactoring  2.217880e+05  8.865177e+04  1.225773e+05
               electricity    4.700901e+05  2.298746e+05  1.006105e+06
               construction   1.097151e+07  3.431794e+04  6.586141e+04
               trade          3.300157e+04  1.960537e+07  4.424292e+04
               transport      9.823204e+04  2.505787e+05  1.102146e+08
               other          2.601768e+04  3.556939e+04  3.218539e+04

region
```

(continues on next page)

(continued from previous page)

```
sector                              other
region         sector
World Region A food           1.163124e+03
               mining         1.908491e+04
               manufactoring  9.631846e+04
               electricity    8.544805e+06
               construction   5.278625e+04
               trade          9.440231e+04
               transport      3.666801e+05
               other          3.158392e+07
World Region B food           7.478749e+04
               mining         2.849191e+05
               manufactoring  1.739262e+05
               electricity    2.680870e+06
               construction   3.153222e+05
               trade          4.933951e+04
               transport      2.615115e+05
               other          4.074021e+07
```

6.12 Characterization of stressors

The characterization of stressors is a standard procedure to calculate the environmental and social impacts of economic activity. This is usually accomplished by multiplying (matrix-multiplication) the stressor-matrix with a characterization-matrix. Doing that in the matrix forms requires a 1:1 correspondence of the columns of the characterization matrix to the rows of the stressor-matrix.

Pymrio uses a different approach with matching the strings of the characterization table (given in long-format) to the available stressors. By doing that, the order of the entries in the characterization-table becomes unimportant. This implementation also allows to use characterization tables which includes characterization for stressors not present in the given satellite account. All characterizations relying on not available stressor will be automatically removed.

6.12.1 Example

For this example we use the test MRIO included in Pymrio. We also need the Pandas library for loading the characterization table and pathlib for some folder manipulation.

[1]:
```
from pathlib import Path
```

[2]:
```
import pandas as pd
```

[3]:
```
import pymrio
from pymrio.core.constants import PYMRIO_PATH  # noqa
```

To load the test MRIO we use:

[4]:
```
io = pymrio.load_test()
```

and the characterization table with some foo factors can be loaded by

[5]:
```
charact_table = pd.read_csv(
    (PYMRIO_PATH["test_mrio"] / Path("concordance") / "emissions_charact.tsv
↪"),
    sep="\t",
)
charact_table
```

[5]:

	stressor	compartment	impact	factor	impact_unit
0	emission_type1	air	air water impact	0.002	t
1	emission_type2	water	air water impact	0.001	t
2	emission_type1	air	total emissions	1.000	kg
3	emission_type2	water	total emissions	1.000	kg
4	emission_type3	land	total emissions	1.000	kg
5	emission_type1	air	total air emissions	0.001	t

This table contains the columns 'stressor' and 'compartment' which correspond to the index names of the test_mrio emission satellite accounts:

[6]:
```
io.emissions.F
```

[6]:
```
region                          reg1                                          ␣
↪ \
sector                          food      mining manufactoring  electricity
stressor       compartment
emission_type1 air          1848064.80  986448.090   23613787.00  28139100.00
emission_type2 water         139250.47   22343.295     763569.18    273981.55

region                                                                    \
sector                    construction      trade   transport      other
stressor       compartment
emission_type1 air           2584141.80  4132656.3  21766987.0  7842090.6
emission_type2 water          317396.51  1254477.8   1012999.1  2449178.0

region                          reg2              ...      reg5              ␣
↪ \
sector                          food      mining  ... transport       other
stressor       compartment                        ...
emission_type1 air          1697937.30  347378.150  ...  42299319  10773826.0
emission_type2 water         204835.44   29463.944  ...   4199841   7191006.3

region                          reg6                                        \
sector                          food     mining manufactoring electricity
stressor       compartment
emission_type1 air          15777996.0  6420955.5   113172450.0  56022534.0
emission_type2 water         4826108.1  1865625.1    12700193.0    753213.7

region
sector                    construction     trade   transport     other
stressor       compartment
emission_type1 air           4861838.5  18195621  47046542.0  21632868
```

(continues on next page)

(continued from previous page)

```
emission_type2 water          2699288.3  13892313   8765784.3  16782553

[2 rows x 48 columns]
```

Theses index-names / columns-names need to match in order to match characterization factors to the stressors.

The other columns names can be passed to the characterization method. By default the method assumes the following column names:

- impact: name of the characterization/impact
- factor: the numerical (float) multiplication value for a specific stressor to derive the impact/characterized account
- impact_unit: the unit of the calculated characterization/impact

Alternative names can be passed through the parameters *characterized_name_column*, *characterization_factors_column* and *characterized_unit_column*.

Note, that units of stressor are currently not checked - units as given in the satellite account to be characterized are assumed. These can be seen by:

[7]:
```
io.emissions.unit
```

[7]:
```
                          unit
stressor       compartment
emission_type1 air           kg
emission_type2 water         kg
```

Also note, that the charact_table contains a characterization called 'total emissions', for which the calculation requires a stressor not present in the satellite account. This will be automatically omitted.

To calculate the characterization we use

[8]:
```
impacts = io.emissions.characterize(charact_table, name="impacts")
```

```
WARNING:root:Impact >total emissions< removed - calculation requires␣
↪stressors not present in extension >Emissions<
```

The parameter *name* is optional, if omitted the name will be set to extension_name + _characterized

The method call above results in a pymrio.Extension which can be inspected with the usual methods, e.g.:

[9]:
```
impacts.F
```

[9]:
```
region                      reg1                                          \
sector                      food       mining manufactoring  electricity
impact
air water impact     3835.38007  1995.239475   47991.14318  56552.18155
total air emissions  1848.06480   986.448090   23613.78700  28139.10000

region                                                          \
sector              construction      trade   transport       other
impact
```

(continues on next page)

(continued from previous page)

```
air water impact      5485.68011  9519.7904  44546.9731  18133.3592
total air emissions   2584.14180  4132.6563  21766.9870   7842.0906

region                     reg2              ...       reg5              \
sector                     food      mining  ...  transport       other
impact                                       ...
air water impact     3600.71004  724.220244  ...  88798.479  28738.6583
total air emissions  1697.93730  347.378150  ...  42299.319  10773.8260

region                     reg6                                        \
sector                     food      mining manufactoring  electricity
impact
air water impact     36382.1001  14707.5361    239045.093  112798.2817
total air emissions  15777.9960   6420.9555    113172.450   56022.5340

region
sector              construction      trade    transport      other
impact
air water impact      12422.9653  50283.555  102858.8683  60048.289
total air emissions    4861.8385  18195.621   47046.5420  21632.868

[2 rows x 48 columns]
```

[10]:
```
impacts.F_Y
```

[10]:
```
region                                                   reg1  \
category            Final consumption expenditure by households
impact
air water impact                                    183877.047
total air emissions                                  62335.321

region                                                                 ␣
↪                            \
category            Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
impact
air water impact                                                  0.0
total air emissions                                               0.0

region                                                          \
category            Final consumption expenditure by government
impact
air water impact                                            0.0
total air emissions                                         0.0

region                                                                \
category            Gross fixed capital formation Changes in inventories
impact
air water impact                              0.0                   0.0
total air emissions                           0.0                   0.0
```

(continues on next page)

(continued from previous page)

```
region                                          \
category            Changes in valuables Export
impact
air water impact                    0.0    0.0
total air emissions                 0.0    0.0

region                                                reg2  \
category            Final consumption expenditure by households
impact
air water impact                                   117347.860
total air emissions                                 38566.929

region                                                                  ␣
↪                             \
category            Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
impact
air water impact                                                     0.0
total air emissions                                                  0.0

region                                                             ...  \
category            Final consumption expenditure by government  ...
impact                                                           ...
air water impact                                                 0.0  ...
total air emissions                                              0.0  ...

region                                reg5                                \
category            Changes in inventories Changes in valuables Export
impact
air water impact                       0.0                  0.0    0.0
total air emissions                    0.0                  0.0    0.0

region                                                reg6  \
category            Final consumption expenditure by households
impact
air water impact                                   1305918.65
total air emissions                                 571278.30

region                                                                  ␣
↪                             \
category            Final consumption expenditure by non-profit organisations␣
↪serving households (NPISH)
impact
air water impact                                                     0.0
total air emissions                                                  0.0

region                                                             \
category            Final consumption expenditure by government
```

(continues on next page)

(continued from previous page)

```
impact
air water impact                                                0.0
total air emissions                                             0.0

region                                                                \
category            Gross fixed capital formation Changes in inventories
impact
air water impact                              0.0                   0.0
total air emissions                           0.0                   0.0

region
category            Changes in valuables Export
impact
air water impact                    0.0    0.0
total air emissions                 0.0    0.0

[2 rows x 42 columns]
```

and the extension can be added to the MRIO

[11]:
```
io.impacts = impacts
```

and used for subsequent calculations:

[12]:
```
io.calc_all()
io.impacts.D_cba
```

[12]:
```
region                      reg1                                        \
sector                      food      mining  manufactoring   electricity
impact
air water impact     4354.677050  384.125264  211698.404833  23912.313854
total air emissions  2056.183383  179.423536   97493.003893  11887.591799

region                                                                  \
sector              construction        trade     transport         other
impact
air water impact     7032.640535  8548.387702  22000.497397  40123.553814
total air emissions  3342.905658  3885.883601  10750.267281  15821.524280

region                      reg2             ...          reg5              ␣
↪\
sector                      food     mining  ...     transport         other
impact                                       ...
air water impact     3800.328439  42.024811  ...  88433.836812  30080.437653
total air emissions  1793.337828  19.145605  ...  42095.049614  11386.614919

region                       reg6                                       \
sector                       food      mining  manufactoring   electricity
impact
air water impact     34765.284482  3227.857425  153582.938963  74236.159458
```

(continues on next page)

(continued from previous page)

```
total air emissions  15172.354999  1345.317901   71450.748974  36831.672907

region
sector              construction          trade      transport          other
impact
air water impact     4580.343177  139321.814038  104944.664681  118394.939263
total air emissions  1836.695857   42415.684852   48054.090102   36022.975074

[2 rows x 48 columns]
```

Characterizing calculated results

The characterize method can also be used to characterize already calculated results. This works in the same way:

[13]:
```
io_aly = pymrio.load_test().calc_all()
```

[14]:
```
io_aly.emissions.D_cba
```

[14]:
```
region                             reg1                                \
sector                             food         mining manufactoring
stressor       compartment
emission_type1 air          2.056183e+06  179423.535893  9.749300e+07
emission_type2 water        2.423103e+05   25278.192086  1.671240e+07

region                                                                  \
sector                      electricity  construction         trade
stressor       compartment
emission_type1 air          1.188759e+07  3.342906e+06  3.885884e+06
emission_type2 water        1.371303e+05  3.468292e+05  7.766205e+05

region                                                          reg2  \
sector                         transport         other          food
stressor       compartment
emission_type1 air          1.075027e+07  1.582152e+07  1.793338e+06
emission_type2 water        4.999628e+05  8.480505e+06  2.136528e+05

region                                    ...          reg5               \
sector                            mining  ...     transport         other
stressor       compartment                ...
emission_type1 air          19145.604911  ...  4.209505e+07  1.138661e+07
emission_type2 water         3733.601474  ...  4.243738e+06  7.307208e+06

region                             reg6                                \
sector                             food        mining manufactoring
stressor       compartment
emission_type1 air          1.517235e+07  1.345318e+06  7.145075e+07
emission_type2 water        4.420574e+06  5.372216e+05  1.068144e+07
```

(continues on next page)

(continued from previous page)

```
region                                                                \
sector                        electricity  construction         trade
stressor       compartment
emission_type1 air           3.683167e+07  1.836696e+06  4.241568e+07
emission_type2 water         5.728136e+05  9.069515e+05  5.449044e+07

region
sector                          transport         other
stressor       compartment
emission_type1 air           4.805409e+07  3.602298e+07
emission_type2 water         8.836484e+06  4.634899e+07

[2 rows x 48 columns]
```

[15]:
```
io_aly.impacts = io_aly.emissions.characterize(charact_table, name="impacts_
↪new")
```

```
WARNING:root:Impact >total emissions< removed - calculation requires␣
↪stressors not present in extension >Emissions<
```

Note, that all results which can be characterized directly (all flow accounts like D_cba, D_pba, ...) are automatically included:

[16]:
```
io_aly.impacts.D_cba
```

[16]:
```
region                      reg1                                        \
sector                      food      mining  manufactoring   electricity
impact
air water impact     4354.677050  384.125264  211698.404833  23912.313854
total air emissions  2056.183383  179.423536   97493.003893  11887.591799

region                                                                  \
sector              construction        trade     transport         other
impact
air water impact     7032.640535  8548.387702  22000.497397  40123.553814
total air emissions  3342.905658  3885.883601  10750.267281  15821.524280

region                      reg2             ...          reg5              ␣
↪\
sector                      food     mining  ...     transport         other
impact                                       ...
air water impact     3800.328439  42.024811  ...  88433.836812  30080.437653
total air emissions  1793.337828  19.145605  ...  42095.049614  11386.614919

region                       reg6                                         \
sector                       food       mining  manufactoring   electricity
impact
air water impact     34765.284482  3227.857425  153582.938963  74236.159458
total air emissions  15172.354999  1345.317901   71450.748974  36831.672907
```

(continues on next page)

(continued from previous page)

```
region
sector              construction          trade      transport          other
impact
air water impact     4580.343177  139321.814038  104944.664681  118394.939263
total air emissions  1836.695857   42415.684852   48054.090102   36022.975074

[2 rows x 48 columns]
```

Whereas coefficient accounts (M, S) are removed:

[17]:
```
io_aly.impacts.M
```

To calculated these use

[18]:
```
io_aly.calc_all()
io_aly.impacts.M
```

[18]:
```
region                   reg1                                        \
sector                   food    mining manufactoring electricity
impact
air water impact     0.022428  0.052572      0.000259    0.225083
total air emissions  0.010865  0.025999      0.000127    0.111897

region                                                          reg2  \
sector              construction     trade transport     other      food
impact
air water impact        0.000295  0.000092  0.000952  0.000291  0.000092
total air emissions     0.000140  0.000042  0.000465  0.000132  0.000044

region                          ...      reg5                reg6           \
sector                  mining  ... transport     other      food    mining
impact                          ...
air water impact     0.016122   ...  0.007839  0.000481  0.000976  0.018407
total air emissions  0.007718   ...  0.003733  0.000182  0.000426  0.008250

region                                                                \
sector              manufactoring electricity construction     trade
impact
air water impact         0.000949    0.447245     0.007421  0.003222
total air emissions      0.000450    0.221901     0.002975  0.001207

region
sector              transport     other
impact
air water impact     0.001553  0.000739
total air emissions  0.000711  0.000277

[2 rows x 48 columns]
```

which will calculate the missing accounts.

For these calculations, the characterized accounts can also be used outside the MRIO system. Thus:

[19]:
```
independent_extension = io_aly.emissions.characterize(charact_table, name=
↪"impacts_new")
```

```
WARNING:root:Impact >total emissions< removed - calculation requires␣
↪stressors not present in extension >Emissions<
```

[20]:
```
type(independent_extension)
```

[20]:
```
pymrio.core.mriosystem.Extension
```

[21]:
```
independent_extension.M
```

[22]:
```
independent_extension_calc = independent_extension.calc_system(x=io_aly.x,␣
↪Y=io_aly.Y)
```

[23]:
```
independent_extension.M
```

[23]:
```
region                   reg1                                        \
sector                   food    mining manufactoring electricity
impact
air water impact     0.022428  0.052572      0.000259    0.225083
total air emissions  0.010865  0.025999      0.000127    0.111897

region                                                          reg2  \
sector              construction     trade transport     other      food
impact
air water impact        0.000295  0.000092  0.000952  0.000291  0.000092
total air emissions     0.000140  0.000042  0.000465  0.000132  0.000044

region                          ...      reg5                reg6           \
sector                  mining  ... transport     other      food    mining
impact                          ...
air water impact     0.016122   ...  0.007839  0.000481  0.000976  0.018407
total air emissions  0.007718   ...  0.003733  0.000182  0.000426  0.008250

region                                                                \
sector              manufactoring electricity construction     trade
impact
air water impact         0.000949    0.447245     0.007421  0.003222
total air emissions      0.000450    0.221901     0.002975  0.001207

region
sector              transport     other
impact
air water impact     0.001553  0.000739
total air emissions  0.000711  0.000277

[2 rows x 48 columns]
```

6.12.2 Inspecting the used characterization table

Pymrio automatically adjust the characterization table by removing accounts which can not be calculated using a given extension. The removed accounts are reported through a warning message (e.g. "WARNING:root:Impact >total emissions< removed - calculation requires stressors not present in extension >Emissions<" in the examples above).

It is also possible, to obtain the cleaned characterization-table for inspection and further use. To do so:

[24]:
```
impacts = io.emissions.characterize(
    charact_table, name="impacts", return_char_matrix=True
)
```

```
WARNING:root:Impact >total emissions< removed - calculation requires␣
↪stressors not present in extension >Emissions<
```

This changes the return type from a pymrio.Extension to a named tuple

[25]:
```
type(impacts)
```

[25]:
```
pymrio.core.mriosystem.characterization
```

with

[26]:
```
impacts.extension
```

[26]:
```
<pymrio.core.mriosystem.Extension at 0x7f37e35c02e0>
```

and

[27]:
```
impacts.factors
```

[27]:
```
         stressor compartment                impact  factor impact_unit
0  emission_type1         air     air water impact   0.002           t
1  emission_type2       water     air water impact   0.001           t
5  emission_type1         air  total air emissions   0.001           t
```

The latter is the characterization table used for the calculation.

For further information see the characterization docstring:

[28]:
```
print(io.emissions.characterize.__doc__)
```

```
Characterize stressors

        Characterizes the extension with the characterization factors given␣
↪in factors.
        Factors can contain more characterization factors which depend on␣
↪stressors not
        present in the Extension - these will be automatically removed.

        Note
        ----
        Accordance of units is not checked - you must ensure that the
        characterization factors correspond to the units of the extension to␣
```

(continues on next page)

(continued from previous page)

```
↪be
        characterized.

        Parameters
        ----------
        factors: pd.DataFrame
            A dataframe in long format with numerical index and columns named
            index.names of the extension to be characterized and
            'characterized_name_column', 'characterization_factors_column',
            'characterized_unit_column'

        characterized_name_column: str (optional)
            Name of the column with the names of the
            characterized account (default: "impact")

        characterization_factors_column: str (optional)
            Name of the column with the factors for the
            characterization (default: "factor")

        characterized_unit_column: str (optional)
            Name of the column with the units of the characterized accounts
            characterization (default: "impact_unit")

        name: string (optional)
            The new name for the extension,
            if None (default): name of the current extension with suffix
            '_characterized'

        return_char_matrix: boolean (optional)
            If False (default), returns just the characterized extension.
            If True, returns a namedtuple with extension and the actually used
            characterization matrix.

        _meta: MRIOMetaData, optional
            Metadata handler for logging, optional. Internal

        Returns
        --------
        pymrio.Extensions or
        namedtuple with (extension: pymrio.Extension, factors: pd.DataFrame)
        depending on return_char_matrix. Only the factors used for the␣
↪calculation
        are returned.
```

6.13 Advanced functionality - pandas groupby with pymrio satellite accounts

This notebook examplifies how to directly apply Pandas core functions (in this case groupby and aggregation) to the pymrio system.

6.13.1 WIOD material extension aggregation - stressor w/o compartment info

Here we use the WIOD MRIO system (see the notebook *"Automatic downloading of MRIO databases"* for how to automatically retrieve this database) and will aggregate the WIOD material stressor for used and unused materials. We assume, that the WIOD system is available at

[1]:
```
wiod_folder = "/tmp/mrios/WIOD2013"
```

To get started we import pymrio

[2]:
```
import pymrio
```

For the example here, we use the data from 2009:

[3]:
```
wiod09 = pymrio.parse_wiod(path=wiod_folder, year=2009)
```

WIOD includes multiple material accounts, specified for the "Used" and "Unused" category, as well as information on the total. We will use the latter to confirm our calculations:

[4]:
```
wiod09.mat.F
```

[4]:

region	AUS				\
sector	AtB	C	15t16	17t18	19
stressor					
Biomass_animals_Used	238.487190	0.000000e+00	0.0	0.0	0.0
Biomass_feed_Used	314501.775775	0.000000e+00	0.0	0.0	0.0
Biomass_food_Used	78736.348430	0.000000e+00	0.0	0.0	0.0
Biomass_forestry_Used	21443.712952	0.000000e+00	0.0	0.0	0.0
Biomass_other_Used	647.038563	0.000000e+00	0.0	0.0	0.0
Fossil_coal_Used	0.000000	4.084490e+05	0.0	0.0	0.0
Fossil_gas_Used	0.000000	3.671908e+04	0.0	0.0	0.0
Fossil_oil_Used	0.000000	2.191849e+04	0.0	0.0	0.0
Fossil_other_Used	0.000000	0.000000e+00	0.0	0.0	0.0
Minerals_construction_Used	0.000000	1.098489e+05	0.0	0.0	0.0
Minerals_industrial_Used	0.000000	2.444270e+04	0.0	0.0	0.0
Minerals_metals_Used	0.000000	7.019911e+05	0.0	0.0	0.0
Biomass_animals_Unused	38.094064	0.000000e+00	0.0	0.0	0.0
Biomass_feed_Unused	194.597667	0.000000e+00	0.0	0.0	0.0
Biomass_food_Unused	17925.841358	0.000000e+00	0.0	0.0	0.0
Biomass_forestry_Unused	3216.556943	0.000000e+00	0.0	0.0	0.0
Biomass_other_Unused	128.610253	0.000000e+00	0.0	0.0	0.0
Fossil_coal_Unused	0.000000	6.430405e+06	0.0	0.0	0.0
Fossil_gas_Unused	0.000000	4.759046e+03	0.0	0.0	0.0
Fossil_oil_Unused	0.000000	4.822068e+03	0.0	0.0	0.0
Fossil_other_Unused	0.000000	0.000000e+00	0.0	0.0	0.0

(continues on next page)

(continued from previous page)

```
Minerals_construction_Unused       0.000000  3.015773e+03   0.0   0.0  0.0
Minerals_industrial_Unused         0.000000  3.389710e+04   0.0   0.0  0.0
Minerals_metals_Unused             0.000000  6.919846e+05   0.0   0.0  0.0
Total                         437071.063196  8.472253e+06   0.0   0.0  0.0

region                                                  ...  RoW            \
sector                         20 21t22   23   24   25  ...   63   64    J
stressor                                                ...
Biomass_animals_Used          0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_feed_Used             0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_food_Used             0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_forestry_Used         0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_other_Used            0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_coal_Used              0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_gas_Used               0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_oil_Used               0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_other_Used             0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_construction_Used    0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_industrial_Used      0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_metals_Used          0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_animals_Unused        0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_feed_Unused           0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_food_Unused           0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_forestry_Unused       0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Biomass_other_Unused          0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_coal_Unused            0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_gas_Unused             0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_oil_Unused             0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Fossil_other_Unused           0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_construction_Unused  0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_industrial_Unused    0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Minerals_metals_Unused        0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
Total                         0.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0

region
sector                         70 71t74    L    M    N    O    P
stressor
Biomass_animals_Used          0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_feed_Used             0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_food_Used             0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_forestry_Used         0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_other_Used            0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_coal_Used              0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_gas_Used               0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_oil_Used               0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_other_Used             0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_construction_Used    0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_industrial_Used      0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_metals_Used          0.0   0.0  0.0  0.0  0.0  0.0  0.0
```

(continues on next page)

(continued from previous page)

```
Biomass_animals_Unused         0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_feed_Unused            0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_food_Unused            0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_forestry_Unused        0.0   0.0  0.0  0.0  0.0  0.0  0.0
Biomass_other_Unused           0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_coal_Unused             0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_gas_Unused              0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_oil_Unused              0.0   0.0  0.0  0.0  0.0  0.0  0.0
Fossil_other_Unused            0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_construction_Unused   0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_industrial_Unused     0.0   0.0  0.0  0.0  0.0  0.0  0.0
Minerals_metals_Unused         0.0   0.0  0.0  0.0  0.0  0.0  0.0
Total                          0.0   0.0  0.0  0.0  0.0  0.0  0.0

[25 rows x 1435 columns]
```

To aggregate these with the Pandas groupby function, we need to specify the groups which should be grouped by Pandas. Pymrio contains a helper function which builds such a matching dictionary. The matching can also include regular expressions to simplify the build:

[5]:
```
groups = wiod09.mat.get_index(
    as_dict=True,
    grouping_pattern={".*_Used": "Material Used", ".*_Unused": "Material␣
↪Unused"},
)
groups
```

[5]:
```
{'Biomass_animals_Used': 'Material Used',
 'Biomass_feed_Used': 'Material Used',
 'Biomass_food_Used': 'Material Used',
 'Biomass_forestry_Used': 'Material Used',
 'Biomass_other_Used': 'Material Used',
 'Fossil_coal_Used': 'Material Used',
 'Fossil_gas_Used': 'Material Used',
 'Fossil_oil_Used': 'Material Used',
 'Fossil_other_Used': 'Material Used',
 'Minerals_construction_Used': 'Material Used',
 'Minerals_industrial_Used': 'Material Used',
 'Minerals_metals_Used': 'Material Used',
 'Biomass_animals_Unused': 'Material Unused',
 'Biomass_feed_Unused': 'Material Unused',
 'Biomass_food_Unused': 'Material Unused',
 'Biomass_forestry_Unused': 'Material Unused',
 'Biomass_other_Unused': 'Material Unused',
 'Fossil_coal_Unused': 'Material Unused',
 'Fossil_gas_Unused': 'Material Unused',
 'Fossil_oil_Unused': 'Material Unused',
 'Fossil_other_Unused': 'Material Unused',
 'Minerals_construction_Unused': 'Material Unused',
 'Minerals_industrial_Unused': 'Material Unused',
```

(continues on next page)

(continued from previous page)

```
 'Minerals_metals_Unused': 'Material Unused',
 'Total': 'Total'}
```

Note, that the grouping contains the rows which do not match any of the specified groups. This allows to easily aggregates only parts of a specific stressor set. To actually omit these groups include them in the matching pattern and provide None as value.

To have the aggregated data alongside the original data, we first copy the detailed satellite account:

[6]:
```
wiod09.mat_agg = wiod09.mat.copy(new_name="Aggregated matrial accounts")
```

Then, we use the pymrio get_DataFrame iterator together with the pandas groupby and sum functions to aggregate the stressors. For the dataframe containing the unit information, we pass a custom function which concatenate non-unique unit strings.

[7]:
```
for df_name, df in zip(
    wiod09.mat_agg.get_DataFrame(data=False, with_unit=True, with_
↪population=False),
    wiod09.mat_agg.get_DataFrame(data=True, with_unit=True, with_
↪population=False),
):
    if df_name == "unit":
        wiod09.mat_agg.__dict__[df_name] = df.groupby(groups).apply(
            lambda x: " & ".join(x.unit.unique())
        )
    else:
        wiod09.mat_agg.__dict__[df_name] = df.groupby(groups).sum()
```

[8]:
```
wiod09.mat_agg.F
```

[8]:
```
region                     AUS                                              ␣
↪ \
sector                     AtB             C 15t16 17t18   19   20 21t22   23
Material Unused   21503.700285  7.168884e+06   0.0   0.0  0.0  0.0   0.0  0.0
Material Used    415567.362910  1.303369e+06   0.0   0.0  0.0  0.0   0.0  0.0
Total            437071.063196  8.472253e+06   0.0   0.0  0.0  0.0   0.0  0.0

region                     ...  RoW                                          ␣
↪\
sector              24   25  ...   63   64    J   70 71t74    L    M    N    O
Material Unused  0.0  0.0  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0
Material Used    0.0  0.0  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0
Total            0.0  0.0  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0  0.0  0.0

region
sector              P
Material Unused  0.0
Material Used    0.0
Total            0.0

[3 rows x 1435 columns]
```

[9]:
```
wiod09.mat_agg.unit
```

[9]:
```
Material Unused    1000 tonnes
Material Used      1000 tonnes
Total              1000 tonnes
dtype: object
```

6.13.2 Use with stressors including compartment information:

The same regular expression grouping can be used to aggregate stressor data which is given per compartment. To do so, the matching dict needs to consist of tuples corresponding to a valid index value in the DataFrames. Each position in the tuple is interpreted as a regular expression. Using the get_index method gives a good indication how a valid grouping dict should look like:

[10]:
```
tt = pymrio.load_test()
tt.emissions.get_index(as_dict=True)
```

[10]:
```
{('emission_type1', 'air'): ('emission_type1', 'air'),
 ('emission_type2', 'water'): ('emission_type2', 'water')}
```

With that information, we can now build our own grouping dict, e.g.:

[11]:
```
agg_groups = {("emis.*", ".*"): "all emissions"}
```

[12]:
```
group_dict = tt.emissions.get_index(as_dict=True, grouping_pattern=agg_groups)
group_dict
```

[12]:
```
{('emission_type1', 'air'): 'all emissions',
 ('emission_type2', 'water'): 'all emissions'}
```

Which can then be used to aggregate the satellite account:

[13]:
```
for df_name, df in zip(
    tt.emissions.get_DataFrame(data=False, with_unit=True, with_
↪population=False),
    tt.emissions.get_DataFrame(data=True, with_unit=True, with_
↪population=False),
):
    if df_name == "unit":
        tt.emissions.__dict__[df_name] = df.groupby(group_dict).apply(
            lambda x: " & ".join(x.unit.unique())
        )
    else:
        tt.emissions.__dict__[df_name] = df.groupby(group_dict).sum()
```

In this case we loose the information on the compartment. To reset the index do:

[14]:
```
import pandas as pd

tt.emissions.set_index(pd.Index(tt.emissions.get_index(), name="stressor"))
```

[15]:
```
tt.emissions.F
```

[15]:
```
region               reg1                                              \
sector               food       mining manufactoring  electricity
stressor
all emissions  1987315.27  1008791.385   24377356.18  28413081.55

region                                                              reg2  \
sector        construction      trade   transport       other        food
stressor
all emissions   2901538.31  5387134.1  22779986.1  10291268.6  1902772.74

region                      ...      reg5                    reg6            \
sector             mining   ... transport       other        food     mining
stressor                    ...
all emissions  376842.094   ...  46499160  17964832.3  20604104.1  8286580.6

region                                                                    \
sector        manufactoring electricity construction     trade   transport
stressor
all emissions   125872643.0  56775747.7    7561126.8  32087934  55812326.3

region
sector            other
stressor
all emissions  38415421

[1 rows x 48 columns]
```

CHAPTER

SEVEN

METADATA AND CHANGE RECORDING

DEPRECATED: this will be removed and changed to loguru soon

Each pymrio core system object contains a field 'meta' which stores meta data as well as changes to the MRIO system. This data is stored as json file in the root of a saved MRIO data and accessible through the attribute '.meta':

[1]:
```
import pymrio

io = pymrio.load_test()
```

[2]:
```
io.meta
```

[2]:
```
Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /home/konstans/proj/pymrio/pymrio/mrio_models/test_mrio/metadata.json
History:
20210224 10:41:58 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/
↪pymrio/mrio_models/test_mrio
20171024 12:11:47 - FILEIO -  Created metadata file ../test_mrio/metadata.json
```

[3]:
```
io.meta("Loaded the pymrio test sytem")
```

```
Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /home/konstans/proj/pymrio/pymrio/mrio_models/test_mrio/metadata.json
History:
20210224 10:41:58 - NOTE -  Loaded the pymrio test sytem
20210224 10:41:58 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/
↪pymrio/mrio_models/test_mrio
20171024 12:11:47 - FILEIO -  Created metadata file ../test_mrio/metadata.json
```

We can now do several steps to modify the system, for example:

[4]:
```
io.calc_all()
io.aggregate(region_agg="global")
```

[4]:
```
<pymrio.core.mriosystem.IOSystem at 0x7f4ed6d73d30>
```

[5]:
```
io.meta
```

[5]:
```
Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /home/konstans/proj/pymrio/pymrio/mrio_models/test_mrio/metadata.json
History:
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension factor_
↪inputs
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate population vector
20210224 10:41:58 - MODIFICATION -  Aggregate industry output x
20210224 10:41:58 - MODIFICATION -  Aggregate transaction matrix Z
20210224 10:41:58 - MODIFICATION -  Aggregate final demand y
20210224 10:41:58 - MODIFICATION -  Reset to absolute flows
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
 ... (more lines in history)
```

Notes can added at any time:

[6]:
```
io.meta.note("First round of calculations finished")
```

[7]:
```
io.meta
```

[7]:
```
Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /home/konstans/proj/pymrio/pymrio/mrio_models/test_mrio/metadata.json
History:
20210224 10:41:58 - NOTE -  First round of calculations finished
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension factor_
↪inputs
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate population vector
20210224 10:41:58 - MODIFICATION -  Aggregate industry output x
20210224 10:41:58 - MODIFICATION -  Aggregate transaction matrix Z
20210224 10:41:58 - MODIFICATION -  Aggregate final demand y
20210224 10:41:58 - MODIFICATION -  Reset to absolute flows
 ... (more lines in history)
```

In addition, all file io operations are recorde in the meta data:

```
[8]: io.save_all("/tmp/foo")
```

```
[8]: <pymrio.core.mriosystem.IOSystem at 0x7f4ed6d73d30>
```

```
[9]: io_new = pymrio.load_all("/tmp/foo")
```

```
[10]: io_new.meta
```

```
[10]: Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v1
File: /tmp/foo/metadata.json
History:
20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/factor_
↪inputs
20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/emissions
20210224 10:41:58 - FILEIO -  Loaded IO system from /tmp/foo
20210224 10:41:58 - FILEIO -  Saved testmrio to /tmp/foo
20210224 10:41:58 - NOTE -  First round of calculations finished
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension factor_
↪inputs
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate population vector
 ... (more lines in history)
```

The top level meta data can be changed as well. These changes will also be recorded in the history:

```
[11]: io_new.meta.change_meta("Version", "v2")
```

```
[12]: io_new.meta
```

```
[12]: Description: test mrio for pymrio
MRIO Name: testmrio
System: pxp
Version: v2
File: /tmp/foo/metadata.json
History:
20210224 10:41:58 - METADATA_CHANGE -  Changed parameter "version" from "v1"␣
↪to "v2"
20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/factor_
↪inputs
20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/emissions
20210224 10:41:58 - FILEIO -  Loaded IO system from /tmp/foo
20210224 10:41:58 - FILEIO -  Saved testmrio to /tmp/foo
20210224 10:41:58 - NOTE -  First round of calculations finished
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions
```

(continues on next page)

(continued from previous page)

```
20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension factor_
↪inputs
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
20210224 10:41:58 - MODIFICATION -  Aggregate extensions...
 ... (more lines in history)
```

To get the full history list, use:

[13]:
```
io_new.meta.history
```

[13]:
```
['20210224 10:41:58 - METADATA_CHANGE -  Changed parameter "version" from "v1
↪" to "v2"',
 '20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/factor_
↪inputs',
 '20210224 10:41:58 - FILEIO -  Added satellite account from /tmp/foo/
↪emissions',
 '20210224 10:41:58 - FILEIO -  Loaded IO system from /tmp/foo',
 '20210224 10:41:58 - FILEIO -  Saved testmrio to /tmp/foo',
 '20210224 10:41:58 - NOTE -  First round of calculations finished',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪factor_inputs',
 '20210224 10:41:58 - MODIFICATION -  Aggregate extensions...',
 '20210224 10:41:58 - MODIFICATION -  Aggregate extensions...',
 '20210224 10:41:58 - MODIFICATION -  Aggregate population vector',
 '20210224 10:41:58 - MODIFICATION -  Aggregate industry output x',
 '20210224 10:41:58 - MODIFICATION -  Aggregate transaction matrix Z',
 '20210224 10:41:58 - MODIFICATION -  Aggregate final demand y',
 '20210224 10:41:58 - MODIFICATION -  Reset to absolute flows',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪factor_inputs',
 '20210224 10:41:58 - MODIFICATION -  Leontief matrix L calculated',
 '20210224 10:41:58 - MODIFICATION -  Coefficient matrix A calculated',
 '20210224 10:41:58 - MODIFICATION -  Industry output x calculated',
 '20210224 10:41:58 - NOTE -  Loaded the pymrio test sytem',
 '20210224 10:41:58 - FILEIO -  Load test_mrio from /home/konstans/proj/
↪pymrio/pymrio/mrio_models/test_mrio',
 '20171024 12:11:47 - FILEIO -  Created metadata file ../test_mrio/metadata.
↪json']
```

This can be restricted to one of the history types by:

[14]:
```
io_new.meta.modification_history
```

[14]:
```
['20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪factor_inputs',
```

(continues on next page)

(continued from previous page)

```
 '20210224 10:41:58 - MODIFICATION -  Aggregate extensions...',
 '20210224 10:41:58 - MODIFICATION -  Aggregate extensions...',
 '20210224 10:41:58 - MODIFICATION -  Aggregate population vector',
 '20210224 10:41:58 - MODIFICATION -  Aggregate industry output x',
 '20210224 10:41:58 - MODIFICATION -  Aggregate transaction matrix Z',
 '20210224 10:41:58 - MODIFICATION -  Aggregate final demand y',
 '20210224 10:41:58 - MODIFICATION -  Reset to absolute flows',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪emissions',
 '20210224 10:41:58 - MODIFICATION -  Calculating accounts for extension␣
↪factor_inputs',
 '20210224 10:41:58 - MODIFICATION -  Leontief matrix L calculated',
 '20210224 10:41:58 - MODIFICATION -  Coefficient matrix A calculated',
 '20210224 10:41:58 - MODIFICATION -  Industry output x calculated']
```

or

```
[15]: io_new.meta.note_history
```

```
[15]: ['20210224 10:41:58 - NOTE -  First round of calculations finished',
 '20210224 10:41:58 - NOTE -  Loaded the pymrio test sytem']
```

CHAPTER

EIGHT

CONTRIBUTING

First off, thanks for taking the time to contribute!

There are many ways you can help to improve pymrio.

- Update and improve the documentation and tutorials.
- File bug reports and describe ideas for enhancement.
- Add new functionality to the code.

Pymrio follows an "issue/ticket driven development". This means, before you start working file an issue describing the planned changes or comment on an existing one to indicate that you work on it. This allows us to discuss changes before you actually start and gives us the chance to identify synergies across ongoing work and avoid potential double work. When you have finished, use a pull request to inform me about the improvements and make sure all tests pass (see below). In the pull request you can use various phrases which automatically close the issue you have been working on.

8.1 Working on the documentation

Any contribution to the description of pymrio is of huge value, in particular I very much appreciate tutorials which show how you can use pymrio in actual research.

The pymrio documentation combines reStructuredText and Jupyter notebooks. The Sphinx Documentation has an excellent introduction to reStructuredText. Review the Sphinx docs to perform more complex changes to the documentation as well.

8.2 Changing the code base

All code contribution must be provided as pull requests connected to a filed issue. Please set-up pull requests against the master branch of the repository. Use numpy style docstrings and lint using black and isort, and follow the pep8 style guide. Passing the black and isort liter is a requirement to pass the tests before merging a pull request.

The following commands can be used to automatically apply the black and isort formatting.

```
pip install black isort
isort --project pymrio --profile black .
black .
```

Check the "script" part in .travis.yml to check the required tests. If you are using Conda you can build a development environment from environment_dev.yml which includes all packages necessary for development, testing and running.

Since pymrio is already used in research projects, please aim for keeping compatibility with previous versions.

8.2.1 Running and extending the tests

Before filing a pull request, make sure your changes pass all tests. Pymrio uses the py.test package for testing. To run the tests either activate the environment_dev.yml file (if you are using Anaconda) or install the test requirments defined in requirments_test.txt.

Then run

```
coverage erase
isort --profile black --check-only .
coverage run -m pytest --black -vv .
coverage report
```

in the root of your local copy of pymrio. The file format_and_test.sh can be used in Linux environments to format the code according to the black / isort format and run all tests.

In addition to the unit tests, the Jupyter notebook tutorials are also used for integration tests of the full software. Some of them (the EXIOBASE and Eora example) require a pre-download of the respective MRIOs to a specific folder. Also, most of the tutorials have POSIX path specifications. However, in case you update some integral part of Pymrio, please either also check if the notebooks run or specify that you did not test them in the pull request.

For testing the notebooks install the nbval extension for py.test . Pymrio includes a sanitizing file to handle changing timestamps and object ids of the notebooks. To test all notebooks run the following command in the pymrio root directory:

```
pytest --nbval --sanitize-with .notebook_test_sanitize.cfg
```

8.2.2 Debugging and logging

Pymrio includes a logging class which is used for documenting changes in the IO system through Pymrio. This is defined in tools/iometadata.py.

To use is import it by

```
from pymrio.tools.iometadata import MRIOMetaData
```

and than document changes by using the methods provided by the class.

All logs necessary only for development or later debugging should be logged by

```
import logging
logging.debug("Message")
```

In the python terminal you can show these debug messages by:

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
```

8.3 Versioning

The versioning system follows http://semver.org/

8.4 Documentation and docstrings

Docstring should follow the numby docstring convention. See

- http://sphinx-doc.org/latest/ext/example_numpy.html
- https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt#docstring-standard

8.5 Open points

Pymrio is under active development. Open points include:

- parser for other available MRIOs
  - OPEN:EU (http://www.oneplaneteconomynetwork.org/)
- improve test cases
- wrapper for time series analysis
  - calculate timeseries
  - extract timeseries data
- reorder sectors/regions
- automatic sector aggregation (perhaps as a separate package similar to the country converter)
- country parameter file (GDP, GDP PPP, Population, area) for normalization of results (similar to the pop vector currently implemented for EXIOBASE 2)
- graphical output
  - flow maps of impacts embodied in trade flows
  - choropleth map for footprints
- structural decomposition analysis

CHAPTER

NINE

CHANGELOG

9.1 v0.6dev

9.1.1 Breaking changes

- get_extensions has a new signature. Two new paramters, names and istance_names.
  - **'names' allows to filter the extensions by name (set names of the attribute .name** or the instance names, also allows to pass the extension itself). Can be used to harmonize the names of an extension list.
  - 'instance_names' can be set to False to get the "set names" of the extensions.

  When using keyword arguments before, the new signature should just work.
- remove_extension: Previous all extensions got removed when no name was given. This has changed. Now all extensions are kept when no name is given (TypeError is raised). To remove all extensions, use mrio.remove_extension(mrio.get_extensions())

9.1.2 Depracted

- extension.get_row_data() The method get_row_data() is deprecated and will be removed in v0.6.0. Use extension.extract() instead.

9.1.3 Misceallaneous

- Updated and restructuring of the documentation

9.2 v0.5.2 - 20230815

9.2.1 New features

- OECD bundle download (by @hazimhussein) - see https://pymrio.readthedocs.io/en/latest/notebooks/autodownload.html#OECD-download
- Fix EORA26 parsing (by @hazimhussein)

9.2.2 Development

- Switched to Micromamba in the CI
- Fixed readthedocs settings

9.3 v0.5.1 - 20230615

- small bugfix with version numbering

9.4 v0.5.0 - 20230615

9.4.1 Development

- Move the repository to the public IndEcol organization on GitHub: https://github.com/IndEcol/pymrio

9.4.2 Breaking changes

- dropped support for Python 3.7 and added 3.10 and 3.11
- License changed to LESSER GNU GENERAL PUBLIC LICENSE v3 (LGPLv3)
- added pyarrow as requirment

9.4.3 New features

- Autodownloader for GLORIA MRIO (by @hazimhussein)
- Support of parquet format for load and save function

9.4.4 Bugfixes

- Fix Eora downloader (by @hazimhussein)

9.5 v0.4.8 - 20221116

- Added inbuild classification for - Test MRIO - EXIOBASE 2 - EXIOBASE 3
- Method for renaming sectors/regions based on the built in classification
- Method for aggregating duplicated indexes

9.5.1 Bugfixes

- F_Y was removed in reset_full - fixed
- updated deprecated pandas methods - fix #93

9.6 v0.4.7 - 20220428

- Fixed OECD downloader and parser (by @jaimeoliver1)

9.7 v0.4.6 - 20211118

- Fixed indexing bug in calc_accounts for non-full Y
- Added Stadler 2021 reference
- change github actions testing to development -> production for multiple os

9.7.1 Breaking changes

- dropped support for Python 3.6

9.8 v0.4.5 (March 03, 2021)

9.8.1 Bugfixes

- Index sorting consistent for all characterized impacts

9.9 v0.4.4 (February 26, 2021)

9.9.1 Bugfixes

- Characterization for cases when some stressors are missing from the characterization matrix
- Spelling mistakes
- Fixed installation description in readme and documentation

9.10 v0.4.3 (February 24, 2021)

9.10.1 New features

- Added automatic downloader for EXIOBASE 3 files
- Method for characterizing stressors (pymrio.Extension.characterize)

9.10.2 Bugfixes

- Fixed: xlrd and numpy requirments for later pandas versions

9.10.3 Development

- Switched from travis to github actions for testing and converage reports

9.11 v0.4.2 (November 19, 2020)

9.11.1 Bugfixes

- Fixed: OECD parsing bug caused by pandas update
- Fixed: Missing inclusion of auxiliary data for exiobase 2
- Fixed: Making python version explicit and update package requirements
- Fixed: hard-coded OS specific path

9.11.2 Development

- switched to black code style
- updated travis.yml for testing different python versions
- added github workflows for automated releases
- switched to git trunk based development

9.12 v0.4.1 (October 08, 2019)

9.12.1 Bugfixes

- Fixed: Parsing EXIOBASE 3 from zip on Windows system
- Fixed: Doc spelling

9.12.2 New features

- The tutorial notebooks of the documentation are now also used for integration tests. See CONTIBUTING.rst for more infos.

9.13 v0.4.0 (August 12, 2019)

9.13.1 New features

- New parser and automatic downloader for the OECD-ICIO tables (2016 and 2018 release)
- Improved test coverage to over 90 %
- Equality comparison for MRIO System and Extension

9.13.2 Bugfixes

- Fixed some typos

9.13.3 Backward incompatible changes

- Minimum python version changed to 3.7
- The FY and SY matrixes has been renamed to F_Y and S_Y. Previously stored data, however, can still be read (FY/SY files are automatically parsed as F_Y and S_Y)

9.14 v0.3.8 (November 06, 2018)

Hotfix for two EXIOBASE 3 issues

- FY in the raw files is named F_hh. F_hh now get automatically renamed to FY.
- In the ixi tables of EXIOBASE 3 some tables had ISO3 country names. The parser now renames these names to the standard ISO2.

9.15 v0.3.7 (October 10, 2018)

9.15.1 New features

- pymrio.parse_exiobase3, accepting the compressed archive files and extraced data (solves #26)
- pymrio.archive for archiving MRIO databases into zipfiles (solves #26)
- pymrio.load and pymrio.load_all can read data directly from a zipfile (solves #26)

9.15.2 Bugfixes

- Calculate FY and SY when final demand impacts are available (fixes issue #28)
- Ensures that mrio.x is a pandas DataFrame (fixes issue #24)
- Some warning if a reset method would remove data beyond recovery by calc_all (see issue #23 discussion)

9.15.3 Removed functionality

- Removed the Eora26 autodownloader b/c worldmrio.com needs a registration now (short time fix for #34)

9.15.4 Misc

- pymrio now depends on python > 3.6
- Stressed the issue driven development in CONTRIBUTING.rst

9.16 v0.3.6 (March 12, 2018)

Function get_index now has a switch to return dict for direct input into pandas groupby function.

Included function to set index across dataframes.

Docs includes examples how to use pymrio with pandas groupby.

Improved test coverage.

9.17 v0.3.5 (Jan 17, 2018)

Added xlrd to requirements

9.18 v0.3.4 (Jan 12, 2018)

9.18.1 API breaking changes

- Footprints and territorial accounts were renamed to “consumption based accounts” and “production based accounts”: D_fp was renamed to D_cba and D_terr to D_pba

9.19 v0.3.3 (Jan 11, 2018)

Note: This includes all changes from 0.3 to 0.3.3

- downloaders for EORA26 and WIOD
- codebase fully pep8 compliant
- restructured and extended the documentation
- License changed to GNU GENERAL PUBLIC LICENSE v3

9.19.1 Dependencies

- pandas minimal version changed to 0.22
- Optional (for aggregation): country converter coco >= 0.6.3

9.19.2 API breaking changes

- The format for saving MRIOs changed from csv + ini to csv + json. Use the method '_load_all_ini_based_io' to read a previously saved MRIO and than save it again to convert to the new save format.
- method set_sectors(), set_regions() and set_Y_categories() renamed to rename_sectors() etc.
- connected the aggregation function to the country_converter coco
- removed previously deprecated method 'per_source'. Use 'diag_stressor' instead.

9.20 v0.2.2 (May 27, 2016)

9.20.1 Dependencies

- pytest. For the unit tests.

9.20.2 Misc

- Fixed filename error for the test system.
- Various small bug fixes.
- Preliminary EXIOBASE 3 parser.
- Preliminary World Input-Output Database (WIOD) parser.

9.21 v0.2.1 (Nov 17, 2014)

9.21.1 Dependencies

- pandas version > 0.15. This required some change in the xls reading within the parser.
- pytest. For the unit tests.

9.21.2 Misc

- Unit testing for all mathematical functions and a first system wide check.
- Fixed some mistakes in the tutorials and readme

9.22 v0.2.0 (Sept 11, 2014)

9.22.1 API changes

- IOSystem.reset() replaced by IOSystem.reset_all_to_flows()
- IOSystem.reset_to_flows() and IOSystem.reset_to_coefficients() added
- Version number attribute added
- Parser for EXIOBASE like extensions (pymrio.parse_exio_ext) added.
- plot_accounts now works also for for specific products (with parameter "sector")

9.22.2 Misc

- Several bugfixes
- Mainmodule split into several packages and submodules
- Added 3rd tutorial
- Added CHANGELOG

9.23 v0.1.0 (June 20, 2014)

Initial version

CHAPTER

TEN

API REFERENCE

API references for all modules

10.1 Data input and output

10.1.1 Test system

load_test()	Returns a small test MRIO

pymrio.load_test

pymrio.**load_test**()

Returns a small test MRIO

The test system contains:

- six regions,
- seven sectors,
- seven final demand categories
- two extensions (emissions and factor_inputs)

The test system only contains Z, Y, F, F_Y. The rest can be calculated with calc_all()

Notes

For development: This function can be used as an example of how to parse an IOSystem

Return type
IOSystem

10.1.2 Download MRIO databases

Download publicly EE MRIO databases from the web. This is currently implemented for the WIOD and OECD_ICIO database (EXIOBASE and EORA26 require registration before downloading).

download_exiobase3(storage_folder[, ...])	years,	Downloads EXIOBASE 3 files from Zenodo
download_wiod2013(storage_folder[, ...])	years,	Downloads the 2013 wiod release
download_oecd(storage_folder[, version, ...])		Downloads the OECD ICIO tables

pymrio.download_exiobase3

pymrio.**download_exiobase3**(*storage_folder*, *years=None*, *system=None*, *overwrite_existing=False*, *doi='10.5281/zenodo.3583070'*)

Downloads EXIOBASE 3 files from Zenodo

Since version 3.7 EXIOBASE gets published on the Zenodo scientific data repository. This function download the lastest available version from Zenodo, for previous version the corresponding DOI (parameter 'doi') needs to specified.

Version 3.7: 10.5281/zenodo.3583071 Version 3.8: 10.5281/zenodo.4277368

Parameters

storage_folder (*str, valid path*) – Location to store the download, folder will be created if not existing. If the file is already present in the folder, the download of the specific file will be skipped.

years: list of int or str, optional

If years is given only downloads the specific years (be default all years will be downloaded). Years must be given in 4 digits.

system: string or list of strings, optional

'pxp': download product by product classification 'ixi': download industry by industry classification ['ixi', 'pxp'] or None (default): download both classifications

overwrite_existing: boolean, optional

If False, skip download of file already existing in the storage folder (default). Set to True to replace files.

doi: string, optional.

The EXIOBASE DOI to be downloaded. By default that resolves to the DOI citing the latest available version. For the previous DOI see the block 'Versions' on the right hand side of https://zenodo.org/record/4277368.

Return type

Meta data of the downloaded MRIOs

pymrio.download_wiod2013

pymrio.**download_wiod2013**(*storage_folder*, *years=None*, *overwrite_existing=False*, *satellite_urls=['http://www.wiod.org/protected3/data13/SEA/WIOD_SEA_July14.xlsx', 'http://www.wiod.org/protected3/data13/EU/EU_may12.zip', 'http://www.wiod.org/protected3/data13/EM/EM_may12.zip', 'http://www.wiod.org/protected3/data13/CO2/CO2_may12.zip', 'http://www.wiod.org/protected3/data13/AIR/AIR_may12.zip', 'http://www.wiod.org/protected3/data13/land/lan_may12.zip', 'http://www.wiod.org/protected3/data13/materials/mat_may12.zip', 'http://www.wiod.org/protected3/data13/water/wat_may12.zip']*)

Downloads the 2013 wiod release

Note: Currently, pymrio only works with the 2013 release of the wiod tables. The more recent 2016 release so far (October 2017) lacks the environmental and social extensions.

Parameters

storage_folder (*str, valid path*) – Location to store the download, folder will be created if not existing. If the file is already present in the folder, the download of the specific file will be skipped.

years: list of int or str, optional

If years is given only downloads the specific years. This only applies to the IO tables because extensions are stored by country and not per year. The years can be given in 2 or 4 digits.

overwrite_existing: boolean, optional

If False, skip download of file already existing in the storage folder (default). Set to True to replace files.

satellite_urls

[list of str (urls), optional] Which satellite accounts to download. Default: satellite urls defined in WIOD_CONFIG - list of all available urls Remove items from this list to only download a subset of extensions

Return type

Meta data of the downloaded MRIOs

pymrio.download_oecd

pymrio.**download_oecd**(*storage_folder*, *version='v2021'*, *years=None*, *overwrite_existing=False*)

Downloads the OECD ICIO tables

Parameters

- **storage_folder** (*str, valid path*) – Location to store the download, folder will be created if not existing. If the file is already present in the folder, the download of the specific file will be skipped.
- **version** (*string or int, optional*) – Two versions of the ICIO OECD tables are currently availabe: Version >v2016<: based on >SNA93< / >ISIC

Rev.3< Version >v2018<: based on >SNA08< / >ISIC Rev.4< (default) Pass any of the identifiers between >< to specifiy the version to be downloaded.

- **years** (*list of int (4 digit) or str, optional*) – If years is given only downloads the specific years or corresponding bundles in v2021. v2021 is only available in 5-years bundles 1995-1999, 2000-2004,..., 2015-2018, specific bundles can be specified as str "1995-1999" otherwise the corresponding bundles for the required years would be downloaded
- **overwrite_existing** (*boolean, optional*) – If False, skip download of file already existing in the storage folder (default). Set to True to replace files.

Return type
Meta data of the downloaded MRIOs

10.1.3 Raw data

parse_exiobase1(path)	Parse the exiobase1 raw data files.
parse_exiobase2(path[, charact, popvector])	Parse the exiobase 2.2.2 source files for the IOSystem
parse_exiobase3(path)	Parses the public EXIOBASE 3 system
generic_exiobase12_parser(exio_files[, system])	Generic EXIOBASE version 1 and 2 parser
parse_wiod(path[, year, names, popvector])	Parse the wiod source files for the IOSystem
parse_eora26(path[, year, price, country_names])	Parse the Eora26 database
parse_oecd(path[, year])	Parse the OECD ICIO tables

pymrio.parse_exiobase1

pymrio.**parse_exiobase1**(*path*)

Parse the exiobase1 raw data files.

This function works with

- pxp_ita_44_regions_coeff_txt
- ixi_fpa_44_regions_coeff_txt
- pxp_ita_44_regions_coeff_src_txt
- ixi_fpa_44_regions_coeff_src_txt

which can be found on www.exiobase.eu

The parser works with the compressed (zip) files as well as the unpacked files.

Parameters
path (*pathlib.Path or string*) – Path of the exiobase 1 data

Return type
pymrio.IOSystem with exio1 data

pymrio.parse_exiobase2

pymrio.**parse_exiobase2**(*path*, *charact=True*, *popvector='exio2'*)

Parse the exiobase 2.2.2 source files for the IOSystem

The function parse product by product and industry by industry source file in the coefficient form (A and S).

Filenames are hardcoded in the parser - for any other function the code has to be adopted. Check git comments to find older verions.

Parameters

- **path** (*string or pathlib.Path*) – Path to the EXIOBASE source files
- **charact** (*string or boolean, optional*) – Filename with path to the characterisation matrices for the extensions (xls). This is provided together with the EXIOBASE system and given as a xls file. The four sheets Q_factorinputs, Q_emission, Q_materials and Q_resources are read and used to generate one new extensions with the impacts. If set to True, the characterisation file found in path is used (can be in the zip or extracted). If a string, it is assumed that it points to valid characterisation file. If False or None, no characterisation file will be used.
- **popvector** (*string or pd.DataFrame, optional*) – The population vector for the countries. This can be given as pd.DataFrame(index = population, columns = countrynames) or, (default) will be taken from the pymrio module. If popvector = None no population data will be passed to the IOSystem.

Returns
A IOSystem with the parsed exiobase 2 data

Return type
IOSystem

Raises
ParserError – If the exiobase source files are not complete in the given path

pymrio.parse_exiobase3

pymrio.**parse_exiobase3**(*path*)

Parses the public EXIOBASE 3 system

This parser works with either the compressed zip archive as downloaded or the extracted system.

Note: The exiobase 3 parser does so far not include population and characterization data.

Parameters
path (*string or pathlib.Path*) – Path to the folder with the EXIOBASE files or the compressed archive.

Returns
A IOSystem with the parsed exiobase 3 data

Return type
IOSystem

pymrio.generic_exiobase12_parser

pymrio.**generic_exiobase12_parser**(*exio_files*, *system=None*)

Generic EXIOBASE version 1 and 2 parser

This is used internally by parse_exiobase1 / 2 functions to parse exiobase files. In most cases, these top-level functions should just work, but in case of archived exiobase versions it might be necessary to use low-level function here.

Parameters

- **exio_files** (*dict of dict*) –
- **system** (*str (pxp or ixi)*) – Only used for the metadata

pymrio.parse_wiod

pymrio.**parse_wiod**(*path*, *year=None*, *names=('isic', 'c_codes')*, *popvector=None*)

Parse the wiod source files for the IOSystem

WIOD provides the MRIO tables in excel - format (xlsx) at http://www.wiod.org/new_site/database/wiots.htm (release November 2013). To use WIOD in pymrio these (for the year of analysis) must be downloaded. The interindustry matrix of these files gets parsed in IOSystem.Z, the additional information is included as factor_input extension (value added,...)

The folder with these xslx must than be passed to the WIOD parsing function. This folder may contain folders with the extension data. Every folder within the wiod root folder will be parsed for extension data and will be added to the IOSystem. The WIOD database offers the download of the environmental extensions as zip files. These can be read directly by the parser. In case a zip file and a folder with the same name are available, the data is read from the folder. If the zip files are extracted into folder, the folders must have the same name as the corresponding zip file (without the 'zip' extension).

If a WIOD SEA file is present (at the root of path or in a folder named 'SEA' - only one file!), the labor data of this file gets included in the factor_input extension (calculated for the the three skill levels available). The monetary data in this file is not added because it is only given in national currency.

Since the "World Input-Output Tables in previous years' prices" are still under construction (20141129), no parser for these is provided.

Some of the meta-parameter of the IOSystem are set automatically based on the values given in the first four cells and the name of the WIOD data files (base year, version, price, iosystem). These can be overwritten afterwards if needed.

Parameters

- **path** (*string or pathlib.Path*) – Path to the folder with the WIOD source files. In case that the path to a specific file is given, only this will be parsed irrespective of the values given in year.

- **year** (*int or str*) – Which year in the path should be parsed. The years can be given with four or two digits (eg [2012 or 12]). If the given path contains a specific file, the value of year will not be used (but inferred from the meta data)- otherwise it must be given For the monetary data the parser searches for files with 'wiot - two digit year'.
- **names** (*string or tuple, optional*) – WIOD provides three different sector/final demand categories naming schemes. These can can be specified for the IOSystem. Pass:

  1) 'isic': ISIC rev 3 Codes - available for interindustry flows and final demand rows.

  2) 'full': Full names - available for final demand rows and final demand columns (categories) and interindustry flows.

  3) 'c_codes' : WIOD specific sector numbers, available for final demand rows and columns (categories) and interindustry flows.

  Internally, the parser relies on 1) for the interindustry flows and 3) for the final demand categories. This is the default and will also be used if just 'isic' gets passed ('c_codes' also replace 'isic' if this was passed for final demand categories). To specify different finial consumption category names, pass a tuple with (sectors/interindustry classification, fd categories), eg ('isic', 'full'). Names are case insensitive and passing the first character is sufficient.

- **popvector** (*TODO*) –

Return type
IOSystem

Raises
ParserError – If the WIOD source file are not complete or inconsistent

pymrio.parse_eora26

pymrio.**parse_eora26**(*path*, *year=None*, *price='bp'*, *country_names='eora'*)

Parse the Eora26 database

Note: This parser deletes the statistical discrepancy columns from the parsed Eora system (reports the amount of loss in the meta records).

Eora does not provide any information on the unit of the monetary values. Based on personal communication the unit is set to Mill USD manually.

Parameters

- **path** (*string or pathlib.Path*) – Path to the Eora raw storage folder or a specific eora zip file to parse. There are several options to specify the data for parsing:

  1) Pass the name of Eora zip file. In this case the parameters 'year' and 'price' will not be used

2) Pass a folder which either contains Eora zip files or unpacked Eora data. In that case, a year must be given

3) Pass a folder which contains subfolders in the format 'YYYY', e.g. '1998' This subfolder can either contain an Eora zip file or an unpacked Eora system

- **year** (*int or str*) – 4 digit year spec. This will not be used if a zip file is specified in 'path'
- **price** (*str, optional*) – 'bp' or 'pp'
- **country_names** (*str, optional*) – Which country names to use: 'eora' = Eora flavoured ISO 3 varian 'full' = Full country names as provided by Eora Passing the first letter suffice.

pymrio.parse_oecd

pymrio.**parse_oecd**(*path*, *year=None*)

Parse the OECD ICIO tables

This function works for both, the 2016 and 2018 release. The OECd webpage provides the data as csv files in zip compressed archives. This function works with both, the compressed archives and the unpacked csv files.

Note: I) The original OECD ICIO tables provide some disaggregation of the Mexican and Chinese tables for the interindustry flows. The pymrio parser automatically aggregates these into Chinese And Mexican totals. Thus, the MX1, MX2, .. and CN1, CN2, … entries are aggregated into MEX and CHN.

II) If a given storage folder contains both releases, the datafile must be specified in the 'path' parameter.

Parameters

- **path** (*str or pathlib.Path*) – Either the full path to one specific OECD ICIO file or the path to a storage folder with several OECD files. In the later case, a specific year needs to be specified.
- **year** (*str or int, optional*) – Year to parse if 'path' is given as a folder. If path points to a specific file, this parameter is not used.

Return type
IOSystem

Raises

- **ParserError** – If the file to parse could not be definitely identified.
- **FileNotFoundError** – If the specified data file could not be found.

10.1.4 Save data

Currently, the full MRIO system can be saved in txt or the python specific binary format ('pickle'). Both formats work with the same API interface:

IOSystem.save(path[, table_format, sep, ...])	Saving the system to path
IOSystem.save_all(path[, table_format, sep, ...])	Saves the system and all extensions

pymrio.IOSystem.save

IOSystem.**save**(*path*, *table_format='txt'*, *sep='\t'*, *float_format='%.12g'*)

Saving the system to path

Parameters

- **path** (*pathlib.Path or string*) – path for the saved data (will be created if necessary, data within will be overwritten).
- **table_format** (*string*) – Format to save the DataFrames:
  - **'pkl'**
    [Binary pickle files,] alias: 'pickle'
  - **'parquet': Parquet format**
    alias: 'par'
  - 'txt' : Text files (default), alias: 'text', 'csv', 'tsv'

sep

[string, optional] Field delimiter for the output file, only for txt files. Default: tab (' ')

float_format

[string, optional] Format for saving the DataFrames, only for txt files. default = '%.12g'

pymrio.IOSystem.save_all

IOSystem.**save_all**(*path*, *table_format='txt'*, *sep='\t'*, *float_format='%.12g'*)

Saves the system and all extensions

Extensions are saved in separate folders (names based on extension)

Parameters are passed to the .save methods of the IOSystem and Extensions. See parameters description there.

Already saved MRIO databases can be archived with

archive(source, archive[, path_in_arc, ...])	Archives a MRIO database as zip file

pymrio.archive

pymrio.**archive**(*source*, *archive*, *path_in_arc=None*, *remove_source=False*, *compression=8*, *compresslevel=-1*)

Archives a MRIO database as zip file

This function is a wrapper around zipfile.write, to ease the writing of an archive and removing the source data.

Note: In contrast to zipfile.write, this function raises an error if the data (path + filename) are identical in the zip archive. Background: the zip standard allows that files with the same name and path are stored side by side in a zip file. This becomes an issue when unpacking this files as they overwrite each other upon extraction.

Parameters

- **source** (*str or pathlib.Path or list of these*) – Location of the mrio data (folder). If not all data should be archived, pass a list of all files which should be included in the archive (absolute path)
- **archive** (*str or pathlib.Path*) – Full path with filename for the archive.
- **path_in_arc** (*string, optional*) – Path within the archive zip file where data should be stored. 'path_in_arc' must be given without leading dot and slash. Thus to point to the data in the root of the compressed file pass '', for data in e.g. the folder 'mrio_v1' pass 'mrio_v1/'. If None (default) data will be stored in the root of the archive.
- **remove_source** (*boolean, optional*) – If True, deletes the source file from the disk (all files specified in 'source' or the specified directory, depending if a list of files or directory was passed). If False, leaves the original files on disk. Also removes all empty directories in source including source.
- **compression** (*ZIP compression method, optional*) – This is passed to zipfile.write. By default it is set to ZIP_DEFLATED. NB: This is different from the zipfile default (ZIP_STORED) which would not give any compression. See https://docs.python.org/3/library/zipfile.html#zipfile-objects for further information. Depending on the value given here additional modules might be necessary (e.g. zlib for ZIP_DEFLATED). Futher information on this can also be found in the zipfile python docs.
- **compresslevel** (*int, optional*) – This is passed to zipfile.write and specifies the compression level. Acceptable values depend on the method specified at the parameter 'compression'. By default, it is set to -1 which gives a compromise between speed and size for the ZIP_DEFLATED compression (this is internally interpreted as 6 as described here: https://docs.python.org/3/library/zlib.html#zlib.compressobj) NB: This is only used if python version >= 3.7

Raises

- **FileExistsError** – In case a file to be archived already present in the:
- **archive.** –

10.1.5 Load processed data

This functions load IOSystems or individual extensions which have been saved with pymrio before.

load(path[, include_core, path_in_arc])	Loads a IOSystem or Extension previously saved with pymrio
load_all(path[, include_core, subfolders, ...])	Loads a full IO system with all extension in path

pymrio.load

pymrio.**load**(*path*, *include_core=True*, *path_in_arc=''*)

Loads a IOSystem or Extension previously saved with pymrio

This function can be used to load a IOSystem or Extension specified in a metadata file (as defined in DEFAULT_FILE_NAMES['filepara']: metadata.json)

The format of DataFrames (tables) are determined by the file extension. Avaialble formats are defined in constant.py - STORAGE_FORMAT. If using the save/save_all functionality from the mriosystem it just works.

Parameters

- **path** (*pathlib.Path or string*) – Path or path with para file name for the data to load. This must either point to the directory containing the uncompressed data or the location of a compressed zip file with the data. In the later case the parameter 'path_in_arc' need to be specific to further indicate the location of the data in the compressed file.
- **include_core** (*boolean, optional*) – If False the load method does not include A, L and Z matrix. This significantly reduces the required memory if the purpose is only to analyse the results calculated beforehand.
- **path_in_arc** (*string, optional*) – Path to the data in the zip file (where the fileparameters file is located). path_in_arc must be given without leading dot and slash; thus to point to the data in the root of the compressed file pass '', for data in e.g. the folder 'emissions' pass 'emissions/'. Only used if parameter 'path' points to an compressed zip file.

Returns

- *IOSystem or Extension class depending on systemtype in the json file*
- *None in case of errors*

pymrio.load_all

pymrio.**load_all**(*path*, *include_core=True*, *subfolders=None*, *path_in_arc=None*)

Loads a full IO system with all extension in path

Parameters

- **path** (*pathlib.Path or string*) – Path or path with para file name for the data to load. This must either point to the directory containing the uncompressed data or the location of a compressed zip file with the data. In the later

case and if there are several mrio's in the zip file the parameter 'path_in_arc' need to be specifiec to further indicate the location of the data in the compressed file.

- **include_core** (*boolean, optional*) – If False the load method does not include A, L and Z matrix. This significantly reduces the required memory if the purpose is only to analyse the results calculated beforehand.
- **subfolders** (*list of pathlib.Path or string, optional*) – By default (subfolders=None), all subfolders in path containing a json parameter file (as defined in DEFAULT_FILE_NAMES['filepara']: metadata.json) are parsed. If only a subset should be used, pass a list of names of subfolders. These can either be strings specifying direct subfolders of path, or absolute/relative path if the extensions are stored at a different location. Both modes can be mixed. If the data is read from a zip archive the path must be given as described below in 'path_in_arc', relative to the root defined in the paramter 'path_in_arc'. Extensions in a different zip archive must be read separately by calling the function 'load' for this extension.
- **path_in_arc** (*string, optional*) – Path to the data in the zip file (where the fileparameters file is located). path_in_arc must be given without leading dot and slash; thus to point to the data in the root of the compressed file pass '', for data in e.g. the folder 'emissions' pass 'emissions/'. Only used if parameter 'path' points to an compressed zip file. Can be None (default) if there is only one mrio database in the zip archive (thus only one file_parameter file as the systemtype entry 'IOSystem'.

10.1.6 Accessing

pymrio stores all tables as pandas DataFrames. This data can be accessed with the usual pandas methods. On top of that, the following functions return (in fact yield) several tables at once:

IOSystem.get_DataFrame([data, with_unit, ...])	Yields all panda.DataFrames or there names
IOSystem.get_extensions([names, data, ...])	Yields the extensions or their names

pymrio.IOSystem.get_DataFrame

IOSystem.**get_DataFrame**(*data=False*, *with_unit=True*, *with_population=True*)

Yields all panda.DataFrames or there names

Notes

For IOSystem this does not include the DataFrames in the extensions.

Parameters

- **data** (*boolean, optional*) – If True, returns a generator which yields the DataFrames. If False, returns a generator which yields only the names of the DataFrames
- **with_unit** (*boolean, optional*) – If True, includes the 'unit' DataFrame If False, does not include the 'unit' DataFrame. The method than only yields the numerical data tables
- **with_population** (*boolean, optional*) – If True, includes the 'population' vector If False, does not include the 'population' vector.

Return type

DataFrames or string generator, depending on parameter data

pymrio.IOSystem.get_extensions

IOSystem.**get_extensions**(*names=None*, *data=False*, *instance_names=True*)

Yields the extensions or their names

Parameters

- **like** (*names = str or list*) – Extension names to yield. If None (default), all extensions are yielded. This can be used to convert from set names to instance names and vice versa or to harmonize a list of extensions.
- **optional** – Extension names to yield. If None (default), all extensions are yielded. This can be used to convert from set names to instance names and vice versa or to harmonize a list of extensions.
- **data** (*boolean, optional*) – If True, returns a generator which yields the extensions. If False, returns a generator which yields the names of the extensions (default)
- **instance_names** (*boolean, optional*) – If True, returns the name of the instance, otherwise the set (custom) name of the extension. (default: True) For example, the test mrio has a extension named 'Factor Inputs' (get it with mrio.factor_inputs.name), and an instance name 'factor_inputs'.

Return type

Generator for Extension or string

For the extensions, it is also possible to receive all data (F, S, M, D_cba, ...) for one specified row.

Extension.get_row_data(row[, name])	Returns a dict with all available data for a row in the extension.

pymrio.Extension.get_row_data

Extension.**get_row_data**(*row*, *name=None*)

Returns a dict with all available data for a row in the extension.

If you need a new extension, see the extract method.

Parameters

- **row** (*index, tuple, list, string*) – A valid index for the extension DataFrames
- **name** (*string, optional*) – If given, adds a key 'name' with the given value to the dict. In that case the dict can be used directly to build a new extension.

Return type

dict object with the data (pandas DataFrame) for the specific rows

10.2 Exploring and Search

10.2.1 Structure

The following functions provide informations about the structure of the IO System and the extensions. The methods work on the IOSystem as well as directly on the Extensions.

IOSystem.get_regions([entries])	Returns the names of regions in the IOSystem as unique names in order
IOSystem.get_sectors([entries])	Names of sectors in the IOSystem as unique names in order
IOSystem.get_Y_categories([entries])	Returns names of y cat.
IOSystem.get_index([as_dict, grouping_pattern])	Returns the index of the DataFrames in the system
IOSystem.set_index(index)	Sets the pd dataframe index of all dataframes in the system to index
IOSystem.get_gross_trade()	Returns the gross bilateral trade flows and totals
Extension.get_rows()	Returns the name of the rows of the extension

pymrio.IOSystem.get_regions

IOSystem.**get_regions**(*entries=None*)

Returns the names of regions in the IOSystem as unique names in order

Parameters

entries (*List, optional*) – If given, retuns an list with None for all values not in entries.

Returns

List of regions, None if now attribute to determine list is available

Return type

Index

pymrio.IOSystem.get_sectors

IOSystem.**get_sectors**(*entries=None*)

Names of sectors in the IOSystem as unique names in order

Parameters

entries (*List, optional*) – If given, retuns an list with None for all values not in entries.

Returns

List of sectors, None if no attribute to determine the list is available

Return type

Index

pymrio.IOSystem.get_Y_categories

IOSystem.**get_Y_categories**(*entries=None*)

Returns names of y cat. of the IOSystem as unique names in order

Parameters

entries (*List, optional*) – If given, retuns an list with None for all values not in entries.

Returns

List of categories, None if no attribute to determine list is available

Return type

Index

pymrio.IOSystem.get_index

IOSystem.**get_index**(*as_dict=False*, *grouping_pattern=None*)

Returns the index of the DataFrames in the system

Parameters

- **as_dict** (*boolean, optional*) – If True, returns a 1:1 key-value matching for further processing prior to groupby functions. Otherwise (default) the index is returned as pandas index.
- **grouping_pattern** (*dict, optional*) – Dictionary with keys being regex patterns matching index and values the name for the grouping. If the index is a pandas multiindex, the keys must be tuples of length levels in the multiindex, with a valid regex expression at each position. Otherwise, the keys need to be strings. Only relevant if as_dict is True. NOTE: This is broken if index is multiindex

pymrio.IOSystem.set_index

IOSystem.**set_index**(*index*)

Sets the pd dataframe index of all dataframes in the system to index

pymrio.IOSystem.get_gross_trade

IOSystem.**get_gross_trade**() → gross_trade

Returns the gross bilateral trade flows and totals

These are the entries of Z and Y with the domestic blocks set to 0.

Returns

A namedTuple with two fields:

- bilat_flows: df with rows: exporting country and sector, columns: importing countries
- totals: df with gross total imports and exports per sector and region

Return type
namedtuple (with two pandas DataFrames)

pymrio.Extension.get_rows

Extension.**get_rows**()

Returns the name of the rows of the extension

10.2.2 Finding accounts

IOSystem.find(term)	Looks for term in index, sectors, regions, Y_categories
IOSystem.contains([find_all])	Check if index of the system contains the regex pattern
IOSystem.match([find_all])	Check if index of the system match the regex pattern
IOSystem.fullmatch([find_all])	Check if a index row of the system is a full match to the regex pattern
Extension.contains([find_all])	Check if index of the system contains the regex pattern
Extension.match([find_all])	Check if index of the system match the regex pattern
Extension.fullmatch([find_all])	Check if a index row of the system is a full match to the regex pattern
IOSystem.extension_contains([find_all, ...])	Get a dict of extension index dicts which contains a search pattern
IOSystem.extension_fullmatch([find_all, ...])	Get a dict of extension index dicts with full match of a search pattern.
IOSystem.extension_match([find_all, extensions])	Get a dict of extension index dicts which match a search pattern
index_contains(df_ix[, find_all])	Check if index contains a regex pattern.

pymrio.IOSystem.find

IOSystem.**find**(*term*)

Looks for term in index, sectors, regions, Y_categories

Mostly useful for a quick check if entry is present.

Internally that uses pd.str.contains as implemented in ioutil.index_contains

For a multiindex, all levels of the multiindex are searched.

Parameters
term (*string*) – String to search for

Returns
With keys 'index', 'region', 'sector', 'Y_category' and values the found index entries. Empty keys are ommited. The values can be used directly on one of the DataFrames with .loc

Return type
dict of (multi)index

pymrio.IOSystem.contains

IOSystem.**contains**(*find_all=None*, ***kwargs*)

Check if index of the system contains the regex pattern

Similar to pandas str.contains, thus the index string must contain the regex pattern. Uses ioutil.index_contains

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex which should be contained. The keys are the index names, the values are the regex. If the entry is not in index name, it is ignored silently.

Returns

The matched rows/index

Return type

pd.Index or pd.MultiIndex

pymrio.IOSystem.match

IOSystem.**match**(*find_all=None*, ***kwargs*)

Check if index of the system match the regex pattern

Similar to pandas str.match, thus the start of the index string must match. Uses ioutil.index_match

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index

Return type
pd.Index or pd.MultiIndex

pymrio.IOSystem.fullmatch

`IOSystem.fullmatch(find_all=None, **kwargs)`

Check if a index row of the system is a full match to the regex pattern

Similar to pandas str.fullmatch, thus the whole string of the index must match. Uses ioutil.index_fullmatch

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index

Return type
pd.Index or pd.MultiIndex

pymrio.Extension.contains

`Extension.contains(find_all=None, **kwargs)`

Check if index of the system contains the regex pattern

Similar to pandas str.contains, thus the index string must contain the regex pattern. Uses ioutil.index_contains

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex which should be contained. The keys are the index names, the values are the regex. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index

Return type
pd.Index or pd.MultiIndex

pymrio.Extension.match

Extension.**match**(*find_all=None*, ***kwargs*)

Check if index of the system match the regex pattern

Similar to pandas str.match, thus the start of the index string must match. Uses ioutil.index_match

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index

Return type
pd.Index or pd.MultiIndex

pymrio.Extension.fullmatch

Extension.**fullmatch**(*find_all=None*, ***kwargs*)

Check if a index row of the system is a full match to the regex pattern

Similar to pandas str.fullmatch, thus the whole string of the index must match. Uses ioutil.index_fullmatch

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index

Return type
pd.Index or pd.MultiIndex

pymrio.IOSystem.extension_contains

IOSystem.**extension_contains**(*find_all=None*, *extensions=None*, ***kwargs*)

Get a dict of extension index dicts which contains a search pattern

This calls the extension.contains for all extensions.

Similar to pandas str.contains, thus the index string must contain the regex pattern.

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **extensions** (*str, list of str, list of extensions, None*) – Which extensions to consider, default (None): all extensions
- **kwargs** (*dict*) – The regex which should be contained. The keys are the index names, the values are the regex. If the entry is not in index name, it is ignored silently.

Returns
A dict with the extension names as keys and an Index/MultiIndex of the matched rows as values

Return type
dict

pymrio.IOSystem.extension_fullmatch

IOSystem.**extension_fullmatch**(*find_all=None*, *extensions=None*, ***kwargs*)

Get a dict of extension index dicts with full match of a search pattern.

This calls the extension.fullmatch for all extensions.

Similar to pandas str.fullmatch, thus the start of the index string must match.

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **extensions** (*str, list of str, list of extensions, None*) – Which extensions to consider, default (None): all extensions
- **kwargs** (*dict*) – The regex which should be contained. The keys are the index names, the values are the regex. If the entry is not in index name, it is ignored silently.

Returns

A dict with the extension names as keys and an Index/MultiIndex of the matched rows as values

Return type

dict

pymrio.IOSystem.extension_match

IOSystem.**extension_match**(*find_all=None*, *extensions=None*, ***kwargs*)

Get a dict of extension index dicts which match a search pattern

This calls the extension.match for all extensions.

Similar to pandas str.match, thus the start of the index string must match.

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **find_all** (*None or str*) – If str (regex pattern) search in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **extensions** (*str, list of str, list of extensions, None*) – Which extensions to consider, default (None): all extensions

- **kwargs** (*dict*) – The regex which should be contained. The keys are the index names, the values are the regex. If the entry is not in index name, it is ignored silently.

Returns
A dict with the extension names as keys and an Index/MultiIndex of the matched rows as values

Return type
dict

pymrio.index_contains

pymrio.**index_contains**(*df_ix*, *find_all=None*, ***kwargs*)

Check if index contains a regex pattern.

Similar to pandas str.contains, thus the index string must contain the regex pattern.

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **df_ix** (*pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex*) – Method works on Index directly or extract from DataFrame/Series
- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index, same type as _dfs_idx

Return type
pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex

10.3 Calculations

10.3.1 Top level methods

The top level level function calc_all checks the IO System and its extensions for missing parts and calculate these. This function calls the specific calculation method for the core system and for the extensions.

IOSystem.calc_all()	Calculates missing parts of the IOSystem and all extensions.
IOSystem.calc_system()	Calculates the missing part of the core IOSystem
Extension.calc_system(x, Y[, Y_agg, L, ...])	Calculates the missing part of the extension plus accounts

pymrio.IOSystem.calc_all

IOSystem.**calc_all**()

Calculates missing parts of the IOSystem and all extensions.

This method call calc_system and calc_extensions

pymrio.IOSystem.calc_system

IOSystem.**calc_system**()

Calculates the missing part of the core IOSystem

The method checks Z, A, x, L and calculates all which are None

The possible cases are:

Case Provided Calculated 1) Z A, x, L 2) A, x Z, L 3) A, Y L, x, Z

pymrio.Extension.calc_system

Extension.**calc_system**(*x*, *Y*, *Y_agg=None*, *L=None*, *population=None*)

Calculates the missing part of the extension plus accounts

This method allows to specify an aggregated Y_agg for the account calculation (see Y_agg below). However, the full Y needs to be specified for the calculation of F_Y or S_Y.

Calculates:

- **for each sector and country:**
  S, S_Y (if F_Y available), M, D_cba, D_pba_sector, D_imp_sector, D_exp_sector
- **for each region:**
  D_cba_reg, D_pba_reg, D_imp_reg, D_exp_reg,
- **for each region (if population vector is given):**
  D_cba_cap, D_pba_cap, D_imp_cap, D_exp_cap

Notes

Only attributes which are not None are recalculated (for D_* this is checked for each group (reg, cap, and w/o appendix)).

Parameters

- **x** (*pandas.DataFrame or numpy.array*) – Industry output column vector
- **Y** (*pandas.DataFrame or numpy.arry*) – Full final demand array
- **Y_agg** (*pandas.DataFrame or np.array, optional*) – The final demand aggregated (one category per country). Can be used to restrict the calculation of CBA of a specific category (e.g. households). Default: y is aggregated over all categories
- **L** (*pandas.DataFrame or numpy.array, optional*) – Leontief input output table L. If this is not given, the method recalculates M based on D_cba (must be present in the extension).
- **population** (*pandas.DataFrame or np.array, optional*) – Row vector with population per region

10.3.2 Low level matrix calculations

The top level functions work by calling the following low level functions. These can also be used independently from the IO System for pandas DataFrames and numpy array.

calc_x(Z, Y)	Calculate the industry output x from the Z and Y matrix
calc_Z(A, x)	calculate the Z matrix (flows) from A and x
calc_A(Z, x)	Calculate the A matrix (coefficients) from Z and x
calc_L(A)	Calculate the Leontief L from A
calc_S(F, x)	Calculate extensions/factor inputs coefficients
calc_F(S, x)	Calculate total direct impacts from the impact coefficients
calc_M(S, L)	Calculate multipliers of the extensions
calc_e(M, Y)	Calculate total impacts (footprints of consumption Y)
calc_accounts(S, L, Y)	Calculate sector specific cba and pba based accounts, imp and exp accounts

pymrio.calc_x

pymrio.**calc_x**(*Z*, *Y*)

Calculate the industry output x from the Z and Y matrix

industry output (x) = flows (sum_columns(Z)) + final demand (sum_columns(Y))

Parameters

- **Z** (*pandas.DataFrame or numpy.array*) – Symmetric input output table (flows)
- **Y** (*pandas.DataFrame or numpy.array*) – final demand with categories (1.order) for each country (2.order)

Returns

Industry output x as column vector The type is determined by the type of Z. If DataFrame index as Z

Return type

pandas.DataFrame or numpy.array

pymrio.calc_Z

pymrio.**calc_Z**(*A*, *x*)

calculate the Z matrix (flows) from A and x

A = Z / x[None, :] => Z = A * x[None, :]

By definition, the coefficient matrix A is basically the normalized flows So Z is just derived from A by un-normalizing using the industrial output x

Parameters

- **A** (*pandas.DataFrame or numpy.array*) – Symmetric input output table (coefficients)
- **x** (*pandas.DataFrame or numpy.array*) – Industry output column vector

Returns

Symmetric input output table (flows) Z The type is determined by the type of A. If DataFrame index/columns as A

Return type

pandas.DataFrame or numpy.array

pymrio.calc_A

pymrio.**calc_A**(*Z*, *x*)

Calculate the A matrix (coefficients) from Z and x

A is a normalized version of the industrial flows Z

Parameters

- **Z** (*pandas.DataFrame or numpy.array*) – Symmetric input output table (flows)

- **x** (*pandas.DataFrame or numpy.array*) – Industry output column vector

Returns
Symmetric input output table (coefficients) A The type is determined by the type of Z. If DataFrame index/columns as Z

Return type
pandas.DataFrame or numpy.array

pymrio.calc_L

pymrio.**calc_L**(*A*)

Calculate the Leontief L from A

L = inverse matrix of (I - A)

Where I is an identity matrix of same shape as A

Comes from:
x = Ax + y => (I-A)x = y

Where:
A: coefficient input () - output () table x: output vector y: final demand vector

Hence, L allows to derive a required output vector x for a given demand y

Parameters
A (*pandas.DataFrame or numpy.array*) – Symmetric input output table (coefficients)

Returns
Leontief input output table L The type is determined by the type of A. If DataFrame index/columns as A

Return type
pandas.DataFrame or numpy.array

pymrio.calc_S

pymrio.**calc_S**(*F*, *x*)

Calculate extensions/factor inputs coefficients

Parameters

- **F** (*pandas.DataFrame or numpy.array*) – Total direct impacts
- **x** (*pandas.DataFrame or numpy.array*) – Industry output column vector

Returns
Direct impact coefficients S The type is determined by the type of F. If DataFrame index/columns as F

Return type
pandas.DataFrame or numpy.array

pymrio.calc_F

pymrio.**calc_F**(*S*, *x*)

Calculate total direct impacts from the impact coefficients

Parameters

- **S** (*pandas.DataFrame or numpy.array*) – Direct impact coefficients S
- **x** (*pandas.DataFrame or numpy.array*) – Industry output column vector

Returns

Total direct impacts F The type is determined by the type of S. If DataFrame index/columns as S

Return type

pandas.DataFrame or numpy.array

pymrio.calc_M

pymrio.**calc_M**(*S*, *L*)

Calculate multipliers of the extensions

Parameters

- **L** (*pandas.DataFrame or numpy.array*) – Leontief input output table L
- **S** (*pandas.DataFrame or numpy.array*) – Direct impact coefficients

Returns

Multipliers M The type is determined by the type of D. If DataFrame index/columns as D

Return type

pandas.DataFrame or numpy.array

pymrio.calc_e

pymrio.**calc_e**(*M*, *Y*)

Calculate total impacts (footprints of consumption Y)

Parameters

- **M** (*pandas.DataFrame or numpy.array*) – Multipliers
- **Y** (*pandas.DataFrame or numpy.array*) – Final consumption
- **(D** (*TODO - this must be completely redone*) –
- **dataframe** (*check for*) –
- **...)** –

Returns

- *pandas.DataFrame or numpy.array* – Multipliers m The type is determined by the type of M. If DataFrame index/columns as M
- *The calcubased on multipliers M and finald demand Y*

pymrio.calc_accounts

pymrio.**calc_accounts**(*S*, *L*, *Y*)

Calculate sector specific cba and pba based accounts, imp and exp accounts

The total industry output x for the calculation is recalculated from L and y

Parameters

- **L** (*pandas.DataFrame*) – Leontief input output table L
- **S** (*pandas.DataFrame*) – Direct impact coefficients
- **Y** (*pandas.DataFrame*) – Final demand: aggregated across categories or just one category, one column per country. Shape: rows as L, Y with just region names

Returns

(D_cba, D_pba, D_imp, D_exp)

Format: D_row x L_col (=nr_countries*nr_sectors)

- D_cba Footprint per sector and country
- D_pba Total factur use per sector and country
- **D_imp Total global factor use to satisfy total final demand in** the country per sector
- **D_exp Total factor use in one country to satisfy final demand** in all other countries (per sector)

Return type
Tuple

10.4 Metadata and history recording

Each pymrio core system object contains a field 'meta' which stores meta data as well as changes to the MRIO system. This data is stored as json file in the root of a saved MRIO data and accessible through the attribute '.meta'.

MRIOMetaData([location, description, name, ...])	
MRIOMetaData.note(entry)	Add the passed string as note to the history
MRIOMetaData.history	All recorded history
MRIOMetaData.modification_history	All modification history entries
MRIOMetaData.note_history	All note history entries
MRIOMetaData.file_io_history	All fileio history entries
MRIOMetaData.save([location])	Saves the current status of the metadata

10.4.1 pymrio.MRIOMetaData

class pymrio.**MRIOMetaData**(*location=None*, *description=None*, *name=None*, *system=None*, *version=None*, *path_in_arc=''*, *logger_function=<function info>*)

__init__(*location=None*, *description=None*, *name=None*, *system=None*, *version=None*, *path_in_arc=''*, *logger_function=<function info>*)

Organzises the MRIO meta data

The meta data is stored in a json file.

Note: The parameters 'description', 'name', 'system', and 'version' should be set during the establishment of the meta data file. If the meta data file already exists and they are given again, the corresponding entry will be overwritten if replace_existing_meta_content is set to True (with a note in the 'History' field.)

Parameters

- **location** (*str, valid path, optional*) – Path or file for loading a previously saved metadata file and/or saving additional metadata (the method 'save' will use this location by default). This can be the full file path or just the storage folder. In the latter case, the filename defined in DEFAULT_FILE_NAMES['metadata'] (currently 'metadata.json') is assumed. The metadata can also be read from an archived zip file. In this case the data is 'read only' and can not be stored back to the same archive directly (writing data as zip archive not implemented yet). This means, that a new file has to be specified when saving the metadata.
- **description** (*str, optional*) – Description of the metadata file purpose and mrio, default set to 'Metadata file for pymrio'. Will be set the first time the metadata file gets established; subsequent changes are recorded in 'history'.
- **name** (*str, optional*) – Name of the mrio (e.g. wiod, exiobase) Will be set the first time the metadata file gets established; subsequent changes are recorded in 'history'.
- **system** (*str, optional*) – For example 'industry by industry', 'ixi', ... Will be set the first time the metadata file gets established; subsequent changes are recorded in 'history'.
- **version** (*str, int, float, optional*) – Version number Will be set the first time the metadata file gets established; subsequent changes are recorded in 'history'.
- **path_in_arc** (*string, optional*) – Path to the meta data in a zip file. path_in_arc must be given without leading dot or slash; thus to point to the data in the root of the compressed file pass '', for data in e.g. the folder 'emissions' pass 'emissions/'. Only used if parameter 'location' points to a compressed zip file.
- **logger_function** (*func, optional*) – Function accepting strings. The info string written to the metadata is also passed to this function. By default, the funtion is set to logging.info. Set to None for no output.

Methods

__init__([location, description, name, ...])	Organzises the MRIO meta data
change_meta(para, new_value[, log])	Changes the meta data
note(entry)	Add the passed string as note to the history
save([location])	Saves the current status of the metadata

Attributes

description	
file_io_history	All fileio history entries
history	All recorded history
metadata	
modification_history	All modification history entries
name	
note_history	All note history entries
system	
version	

10.4.2 pymrio.MRIOMetaData.note

MRIOMetaData.**note**(*entry*)

Add the passed string as note to the history

If log is True (default), also log the string by logging.info

10.4.3 pymrio.MRIOMetaData.history

property MRIOMetaData.**history**

All recorded history

10.4.4 pymrio.MRIOMetaData.modification_history

property MRIOMetaData.**modification_history**

All modification history entries

10.4.5 pymrio.MRIOMetaData.note_history

property MRIOMetaData.**note_history**

All note history entries

10.4.6 pymrio.MRIOMetaData.file_io_history

property MRIOMetaData.**file_io_history**

All fileio history entries

10.4.7 pymrio.MRIOMetaData.save

MRIOMetaData.**save**(*location=None*)

Saves the current status of the metadata

This saves the metadata at the location of the previously loaded metadata or at the file/path given in location.

Specify a location if the metadata should be stored in a different location or was never stored before. Subsequent saves will use the location set here.

Parameters

location (*str, optional*) – Path or file for saving the metadata. This can be the full file path or just the storage folder. In the latter case, the filename defined in DEFAULT_FILE_NAMES['metadata'] (currently 'metadata.json') is assumed.

10.5 Modifiying the IO System and its Extensions

10.5.1 Aggregation

The IO System method 'aggregate' accepts concordance matrices and/or aggregation vectors. The latter can be generated automatically for various aggregation levels for the test system and EXIOBASE 2.

IOSystem.aggregate([region_agg, sector_agg, ...])	Aggregates the IO system.
build_agg_vec(agg_vec, **source)	Builds an combined aggregation vector considering diff classifications

pymrio.IOSystem.aggregate

IOSystem.**aggregate**(*region_agg=None*, *sector_agg=None*, *region_names=None*, *sector_names=None*, *inplace=True*)

Aggregates the IO system.

Aggregation can be given as vector (use pymrio.build_agg_vec) or aggregation matrix. In the case of a vector this must be of length self.get_regions() / self.get_sectors() respectively with the new position as integer or a string of the new name. In the case of strings the final output order can be specified in region_dict and sector_dict in the format {str1 = int_pos, str2 = int_pos, ... }.

If the sector / region concordance is given as matrix or numerical vector, generic names will be used for the new sectors/regions. One can define specific names by defining the aggregation as string vector

Parameters

- **region_agg** (*list, array or string, optional*) – The aggregation vector or matrix for the regions (np.ndarray or list). If string: aggregates to one total region and names is to the given string. Pandas DataFrame with columns 'original' and 'aggregated'. This is the output from the country_converter.agg_conc
- **sector_agg** (*list, arrays or string, optional*) – The aggregation vector or matrix for the sectors (np.ndarray or list).If string: aggregates to one total region and names is to the given string.
- **region_names** (*list, optional*) – Names for the aggregated regions. If concordance matrix - in order of rows in this matrix If concordance vector - in order or num. values in this vector If string based - same order as the passed string Not considered if passing a DataFrame - in this case give the names in the column 'aggregated'
- **sector_names** (*list, optional*) – Names for the aggregated sectors. Same behaviour as 'region_names'
- **inplace** (*boolean, optional*) – If True, aggregates the IOSystem in place (default), otherwise aggregation happens on a copy of the IOSystem. Regardless of the setting, the IOSystem is returned to allow for chained operations.

Returns
Aggregated IOSystem (if inplace is False)

Return type
IOSystem

pymrio.build_agg_vec

pymrio.**build_agg_vec**(*agg_vec*, ***source*)

Builds an combined aggregation vector considering diff classifications

This function build an aggregation vector based on the order in agg_vec. The naming and actual mapping is given in source, either explicitly or by pointing to a folder with the mapping.

Parameters

- **agg_vec** (*list*) – A list of sector or regions to which the IOSystem shall be aggregated. The order in agg_vec is important: If a string was assigned to one specific entry it will not be overwritten if it is given in the next vector, e.g. ['EU', 'OECD'] would aggregate first into EU and the remaining one into OECD, whereas ['OECD', 'EU'] would first aggregate all countries into OECD and than the remaining countries into EU.
- **source** (*list or string*) – Definition of the vectors in agg_vec. The input vectors (either in the file or given as list for the entries in agg_vec) must be as long as the desired output with a string for every position which should be aggregated and None for position which should not be used.

Special keywords:

- **path**
  [Path to a folder with concordance matrices.] The files in the folder can have any extension but must be in text format (tab separated) with one entry per row. The last column in the file will be taken as aggregation vectors (other columns can be used for documentation). Values must be given for every entry in the original classification (string None for all values not used) If the same entry is given in source and as text file in path than the one in source will be used.

  Two special path entries are available so far:

  * 'exio2' Concordance matrices for EXIOBASE 2.0
  * 'test' Concordance matrices for the test IO system

  If a entry is not found in source and no path is given the current directory will be searched for the definition.
- miss : Entry to use for missing values, default: 'REST'

Returns
The aggregation vector

Return type
list

Examples

```
>>> build_agg_vec(['EU', 'OECD'], path = 'test')
['EU', 'EU', 'EU', 'OECD', 'REST', 'REST']
```

```
>>> build_agg_vec(['OECD', 'EU'], path = 'test', miss='RoW')
['OECD', 'EU', 'OECD', 'OECD', 'RoW', 'RoW']
```

```
>>> build_agg_vec(['EU', 'orig_regions'], path = 'test')
['EU', 'EU', 'EU', 'reg4', 'reg5', 'reg6']
```

```
>>> build_agg_vec(['supreg1', 'other'], path = 'test',
>>>        other = [None, None, 'other1', 'other1', 'other2', 'other2'])
['supreg1', 'supreg1', 'other1', 'other1', 'other2', 'other2']
```

10.5.2 Characterizing stressors

Extension.characterize(factors[, ...])	Characterize stressors

pymrio.Extension.characterize

Extension.**characterize**(*factors*, *characterized_name_column='impact'*, *characterization_factors_column='factor'*, *characterized_unit_column='impact_unit'*, *name=None*, *return_char_matrix=False*, *_meta=None*)

Characterize stressors

Characterizes the extension with the characterization factors given in factors. Factors can contain more characterization factors which depend on stressors not present in the Extension - these will be automatically removed.

The dataframe passed for the characterization must be in a long format. It must contain columns with the same names as in the index of the extension.

The routine can also handle region or sector specific characterization factors. In that case, the passed dataframe must also include columns for sector and/or region. The names must be the same as the column names of the extension.

Other column names can be specified in the parameters, see below for the default values.

Note: Accordance of units is not checked - you must ensure that the characterization factors correspond to the units of the extension to be characterized.

Parameters

- **factors** (*pd.DataFrame*) – A dataframe in long format with numerical index and columns named index.names of the extension to be characterized and 'characterized_name_column', 'characterization_factors_column', 'characterized_unit_column'
- **characterized_name_column** (*str (optional)*) – Name of the column with the names of the characterized account (default: "impact")
- **characterization_factors_column** (*str (optional)*) – Name of the column with the factors for the characterization (default: "factor")
- **characterized_unit_column** (*str (optional)*) – Name of the column with the units of the characterized accounts characterization (default: "impact_unit")
- **name** (*string (optional)*) – The new name for the extension, if None (default): name of the current extension with suffix '_characterized'
- **return_char_matrix** (*boolean (optional)*) – If False (default), returns just the characterized extension. If True, returns a namedtuple with extension and the actually used characterization matrix.
- **_meta** (MRIOMetaData, *optional*) – Metadata handler for logging, optional. Internal

Returns

- *pymrio.Extensions (when return_char_matrix==False, default)* – or
- *namedtuple with* –

- .extension: pymrio.Extension
- .factors: pd.DataFrame (when return_char_matrix==True)

10.5.3 Analysing the source of impacts

Extension.diag_stressor(stressor[, name, _meta])	Diagonalize one row of the stressor matrix for a flow analysis.

pymrio.Extension.diag_stressor

Extension.**diag_stressor**(*stressor*, *name=None*, *_meta=None*)

Diagonalize one row of the stressor matrix for a flow analysis.

This method takes one row of the F matrix and diagonalize to the full region/sector format. Footprints calculation based on this matrix show the flow of embodied stressors from the source region/sector (row index) to the final consumer (column index).

Note: Since the type of analysis based on the disaggregated matrix is based on flows, direct household emissions (F_Y) are not included.

Parameters

- **stressor** (*str or int - valid index for one row of the F matrix*) – This must be a tuple for a multiindex, a string otherwise. The stressor to diagonalize.
- **name** (*string (optional)*) – The new name for the extension, if None (default): string based on the given stressor (row name)
- **_meta** (*MRIOMetaData, optional*) – Metadata handler for logging, optional. Internal

Return type
pymrio.Extension

10.5.4 Changing extensions

IOSystem.remove_extension(ext)	Remove extension from IOSystem
concate_extension(*extensions, name)	Concatenate extensions
parse_exio12_ext(ext_file, index_col, name)	Parse an EXIOBASE version 1 or 2 like extension file into pymrio.Extension

pymrio.IOSystem.remove_extension

IOSystem.**remove_extension**(*ext*)

Remove extension from IOSystem

For single Extensions the same can be achieved with del IOSystem_name.Extension_name

Parameters

ext (*string or list, optional*) – The extension to remove, this can be given as the name of the instance or of Extension.name. instance was found)

pymrio.concate_extension

pymrio.**concate_extension**(**extensions*, *name*)

Concatenate extensions

Notes

The method assumes that the first index is the name of the stressor/impact/input type. To provide a consistent naming this is renamed to 'indicator' if they differ. All other index names ('compartments', ...) are added to the concatenated extensions and set to NaN for missing values.

Notes

Attributes which are not DataFrames will be set to None if they differ between the extensions

Parameters

- **extensions** (*Extensions*) – The Extensions to concatenate as multiple parameters
- **name** (*string*) – Name of the new extension

Return type

Concatenated extension

pymrio.parse_exio12_ext

pymrio.**parse_exio12_ext**(*ext_file*, *index_col*, *name*, *drop_compartment=True*, *version=None*, *year=None*, *iosystem=None*, *sep=','*)

Parse an EXIOBASE version 1 or 2 like extension file into pymrio.Extension

EXIOBASE like extensions files are assumed to have two rows which are used as columns multi-index (region and sector) and up to three columns for the row index (see Parameters).

For EXIOBASE 3 - extension can be loaded directly with pymrio.load

Notes

So far this only parses factor of production extensions F (not final demand extensions F_Y nor coeffiecents S).

Parameters

- **ext_file** (*string or pathlib.Path*) – File to parse
- **index_col** (*int*) – The number of columns (1 to 3) at the beginning of the file to use as the index. The order of the index_col must be - 1 index column: ['stressor'] - 2 index columns: ['stressor', 'unit'] - 3 index columns: ['stressor', 'compartment', 'unit'] - > 3: everything up to three index columns will be removed
- **name** (*string*) – Name of the extension
- **drop_compartment** (*boolean, optional*) – If True (default) removes the compartment from the index.
- **version** (*string, optional*) – see pymrio.Extension
- **iosystem** (*string, optional*) – see pymrio.Extension
- **year** (*string or int*) – see pymrio.Extension
- **sep** (*string, optional*) – Delimiter to use; default ','

Returns
with F (and unit if available)

Return type
pymrio.Extension

10.5.5 Adjusting and Renaming

IOSystem.rename_regions(regions)	Sets new names for the regions
IOSystem.rename_sectors(sectors)	Sets new names for the sectors
IOSystem.rename_Y_categories(Y_categories)	Sets new names for the Y_categories

pymrio.IOSystem.rename_regions

IOSystem.**rename_regions**(*regions*)

Sets new names for the regions

Parameters

regions (*list or dict*) –

In case of dict: {'old_name'
['new_name'} with a] entry for each old_name which should be renamed

In case of list: List of new names in order and complete
without repetition

pymrio.IOSystem.rename_sectors

IOSystem.**rename_sectors**(*sectors*)

Sets new names for the sectors

Parameters

sectors (*list or dict*) –

In case of dict: {'old_name'
['new_name'} with an] entry for each old_name which should be renamed

In case of list: List of new names in order and
complete without repetition

pymrio.IOSystem.rename_Y_categories

IOSystem.**rename_Y_categories**(*Y_categories*)

Sets new names for the Y_categories

Parameters

Y_categories (*list or dict*) –

In case of dict: {'old_name'
['new_name'} with an] entry for each old_name which should be renamed

In case of list: List of new names in order and
complete without repetition

10.5.6 Getting predefined Classifications of MRIOs

get_classification([mrio_name])	Get predefined classifications included in pymrio
ClassificationData(...)	Classification data for MRIOs

pymrio.get_classification

pymrio.**get_classification**(*mrio_name: Optional[str] = None*)

Get predefined classifications included in pymrio

Parameters

- **mrio_name** (*str*) – MRIO for which to get the classification. Pass None (default) for a list of available classifications.
- **Returns** – pymrio.ClassificationData

pymrio.ClassificationData

class pymrio.**ClassificationData**(*mrio_classification_folder: Union[str, Path]*, *sectors_file_name: str*, *sectors_sep: str*, *finaldemand_file_name: str*, *finaldemand_sep: str*)

Classification data for MRIOs

Typically includes .sectors and .finaldemand

__init__(*mrio_classification_folder: Union[str, Path]*, *sectors_file_name: str*, *sectors_sep: str*, *finaldemand_file_name: str*, *finaldemand_sep: str*) → None

Methods

__init__(mrio_classification_folder, ...)	
get_sector_dict(orig, new)	Returns sector rename dict based

Attributes

mrio_classification_folder
sectors_file_name
sectors_sep
finaldemand_file_name
finaldemand_sep
sectors
finaldemand

10.6 Report

The following method works on the IO System (generating reports for every extension available) or at individual extensions.

IOSystem.report_accounts(path[, per_region, ...])	Generates a report to the given path for all extension

10.6.1 pymrio.IOSystem.report_accounts

IOSystem.**report_accounts**(*path*, *per_region=True*, *per_capita=False*, *pic_size=1000*, *format='rst'*, ***kwargs*)

Generates a report to the given path for all extension

This method calls .report_accounts for all extensions

Notes

This looks prettier with the seaborn module (import seaborn before calling this method)

Parameters

- **path** (*string*) – Root path for the report
- **per_region** (*boolean, optional*) – If true, reports the accounts per region
- **per_capita** (*boolean, optional*) – If true, reports the accounts per capita If per_capita and per_region are False, nothing will be done
- **pic_size** (*int, optional*) – size for the figures in px, 1000 by default
- **format** (*string, optional*) – file format of the report: 'rst'(default), 'html', 'latex', ... except for rst all depend on the module docutils (all writer_name from docutils can be used as format)
- **ffname** (*string, optional*) – root file name (without extension, per_capita or per_region will be attached) and folder names If None gets passed (default), self.name with be modified to get a valid name for the operation system without blanks
- ****kwargs** (*key word arguments, optional*) – This will be passed directly to the pd.DataFrame.plot method (through the self.plot_account method)

10.7 Visualization

Extension.plot_account(row[, per_capita, ...])	Plots D_pba, D_cba, D_imp and D_exp for the specified row (account)

10.7.1 pymrio.Extension.plot_account

Extension.**plot_account**(*row*, *per_capita=False*, *sector=None*, *file_name=False*, *file_dpi=600*, *population=None*, ***kwargs*)

Plots D_pba, D_cba, D_imp and D_exp for the specified row (account)

Plot either the total country accounts or for a specific sector, depending on the 'sector' parameter.

Per default the accounts are plotted as bar charts. However, any valid keyword for the pandas.DataFrame.plot method can be passed.

Notes

This looks prettier with the seaborn module (import seaborn before calling this method)

Parameters

- **row** (*string, tuple or int*) – A valid index for the row in the extension which should be plotted (one(!) row - no list allowed)
- **per_capita** (*boolean, optional*) – Plot the per capita accounts instead of the absolute values default is False
- **sector** (*string, optional*) – Plot the results for a specific sector of the IO table. If None is given (default), the total regional accounts are plotted.
- **population** (*pandas.DataFrame or np.array, optional*) – Vector with population per region. This must be given if values should be plotted per_capita for a specific sector since these values are calculated on the fly.
- **file_name** (*path string, optional*) – If given, saves the plot to the given filename
- **file_dpi** (*int, optional*) – Dpi for saving the figure, default 600
- ****kwargs** (*key word arguments, optional*) – This will be passed directly to the pd.DataFrame.plot method

Return type

Axis as given by pandas.DataFrame.plot, None in case of errors

10.8 Utility

Routines developed for pymrio which can also be used independently.

index_match(df_ix[, find_all])	Match regex on index of df_ix
index_contains(df_ix[, find_all])	Check if index contains a regex pattern.
index_fullmatch(df_ix[, find_all])	Fullmatch regex on index of df_ix
build_agg_matrix(agg_vector[, pos_dict])	Aggregation based on mapping given in input as numerical or str vector.
build_agg_vec(agg_vec, **source)	Builds an combined aggregation vector considering diff classifications
convert_to_long(df[, value_name])	Converts the pymrio matrix df format to a long format

10.8.1 pymrio.index_match

pymrio.**index_match**(*df_ix*, *find_all=None*, ***kwargs*)

Match regex on index of df_ix

Similar to pandas str.match, thus the start of the index string must match.

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **df_ix** (*pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex*) – Method works on Index directly or extract from DataFrame/Series
- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns

The matched rows/index, same type as _dfs_idx

Return type

pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex

10.8.2 pymrio.index_fullmatch

pymrio.**index_fullmatch**(*df_ix*, *find_all=None*, ***kwargs*)

Fullmatch regex on index of df_ix

Similar to pandas str.fullmatch, thus the whole string of the index must match.

The index levels need to be named (df.index.name needs to be set for all levels).

Note: Arguments are set to case=True, flags=0, na=False, regex=True. For case insensitive matching, use (?i) at the beginning of the pattern.

See the pandas/python.re documentation for more details.

Parameters

- **df_ix** (*pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex*) – Method works on Index directly or extract from DataFrame/Series

- **find_all** (*None or str*) – If str (regex pattern) search for all matches in all index levels. All matching rows are returned. The remaining kwargs are ignored.
- **kwargs** (*dict*) – The regex to match. The keys are the index names, the values are the regex to match. If the entry is not in index name, it is ignored silently.

Returns
The matched rows/index, same type as _dfs_idx

Return type
pd.DataFrame, pd.Series, pd.Index or pd.MultiIndex

10.8.3 pymrio.build_agg_matrix

pymrio.**build_agg_matrix**(*agg_vector*, *pos_dict=None*)

Aggregation based on mapping given in input as numerical or str vector.

Parameters

- **agg_vector** (*list or vector like numpy ndarray*) – This can be row or column vector. Length m with position given for n and -1 if values should not be included or length m with id_string for the aggregation
- **pos_dict** (*dictionary*) – (only possible if agg_vector is given as string) output order for the new matrix must be given as dict with 'string in agg_vector' = pos (as int, -1 if value should not be included in the aggregation)

Returns
agg_matrix – Aggregation matrix with shape (n, m) with n (rows) indicating the new classification and m (columns) the old classification

Return type
numpy ndarray

Examples

Assume an input vector with either

```
>>> inp1 = np.array([0, 1, 1, 2])
```

or

```
>>> inp2 = ['a', 'b', 'b', 'c']
```

inp1 and inp2 are equivalent, the entries are just indicators.

```
>>> build_agg_matrix(inp1)
>>> build_agg_matrix(inp2)
```

Returns

```
>>> array([[1., 0., 0., 0.],
>>>        [0., 1., 1., 0.],
>>>        [0., 0., 0., 1.]])
```

The order can be defined by a dictionary, thus

```
>>> pymrio.build_agg_matrix(np.array([1, 0, 0, 2]))
```

is equivalent to

```
>>> pymrio.build_agg_matrix(['b', 'a', 'a', 'c'], dict(a=0,b=1,c=2))
```

```
>>> array([[0., 1., 1., 0.],
>>>        [1., 0., 0., 0.],
>>>        [0., 0., 0., 1.]])
```

10.8.4 pymrio.convert_to_long

pymrio.**convert_to_long**(*df*, *value_name='value'*)

Converts the pymrio matrix df format to a long format

FIX: All index and columns become separate columns (not index!)

Parameters

- **df** (*pymrio matrix*) – The pymrio matrix (e.g. Z, F, etc) to convert
- **value_name** (*str, optional*) – The name of the value column, default: 'value' as defined in constants.py

Return type

pd.DataFrame

10.9 Miscellaneous

IOSystem.reset_to_flows([force])	Keeps only the absolute values.
IOSystem.reset_to_coefficients()	Keeps only the coefficient.
IOSystem.copy([new_name])	Returns a deep copy of the system

10.9.1 pymrio.IOSystem.reset_to_flows

IOSystem.**reset_to_flows**(*force=False*)

Keeps only the absolute values.

This removes all attributes which can not be aggregated and must be recalculated after the aggregation.

Parameters

force (*boolean, optional*) – If True, reset to flows although the system can not be recalculated. Default: False

10.9.2 pymrio.IOSystem.reset_to_coefficients

IOSystem.**reset_to_coefficients**()

Keeps only the coefficient.

This can be used to recalculate the IO tables for a new finald demand.

Note: The system can not be reconstructed after this steps because all absolute data is removed. Save the Y data in case a reconstruction might be necessary.

10.9.3 pymrio.IOSystem.copy

IOSystem.**copy**(*new_name=None*)

Returns a deep copy of the system

Parameters

new_name (*str, optional*) – Set a new meta name parameter. Default: <old_name>_copy

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

__init__() (*pymrio.ClassificationData method*), 170
__init__() (*pymrio.MRIOMetaData method*), 160

A

aggregate() (*pymrio.IOSystem method*), 162
archive() (*in module pymrio*), 140

B

build_agg_matrix() (*in module pymrio*), 174
build_agg_vec() (*in module pymrio*), 163

C

calc_A() (*in module pymrio*), 156
calc_accounts() (*in module pymrio*), 159
calc_all() (*pymrio.IOSystem method*), 154
calc_e() (*in module pymrio*), 158
calc_F() (*in module pymrio*), 158
calc_L() (*in module pymrio*), 157
calc_M() (*in module pymrio*), 158
calc_S() (*in module pymrio*), 157
calc_system() (*pymrio.Extension method*), 154
calc_system() (*pymrio.IOSystem method*), 154
calc_x() (*in module pymrio*), 156
calc_Z() (*in module pymrio*), 156
characterize() (*pymrio.Extension method*), 165
ClassificationData (*class in pymrio*), 170
concate_extension() (*in module pymrio*), 167
contains() (*pymrio.Extension method*), 149
contains() (*pymrio.IOSystem method*), 148
convert_to_long() (*in module pymrio*), 175
copy() (*pymrio.IOSystem method*), 176

D

diag_stressor() (*pymrio.Extension method*), 166
download_exiobase3() (*in module pymrio*), 132
download_oecd() (*in module pymrio*), 133
download_wiod2013() (*in module pymrio*), 133

E

extension_contains() (*pymrio.IOSystem method*), 151
extension_fullmatch() (*pymrio.IOSystem method*), 152
extension_match() (*pymrio.IOSystem method*), 152

F

file_io_history (*pymrio.MRIOMetaData property*), 162
find() (*pymrio.IOSystem method*), 147
fullmatch() (*pymrio.Extension method*), 150
fullmatch() (*pymrio.IOSystem method*), 149

G

generic_exiobase12_parser() (*in module pymrio*), 136
get_classification() (*in module pymrio*), 169
get_DataFrame() (*pymrio.IOSystem method*), 143
get_extensions() (*pymrio.IOSystem method*), 143
get_gross_trade() (*pymrio.IOSystem method*), 146
get_index() (*pymrio.IOSystem method*), 145
get_regions() (*pymrio.IOSystem method*), 144
get_row_data() (*pymrio.Extension method*), 144
get_rows() (*pymrio.Extension method*), 146
get_sectors() (*pymrio.IOSystem method*), 145
get_Y_categories() (*pymrio.IOSystem method*), 145

H

history (*pymrio.MRIOMetaData property*), 161

I

index_contains() (*in module pymrio*), 153

index_fullmatch() (*in module pymrio*), 173
index_match() (*in module pymrio*), 173

L

load() (*in module pymrio*), 141
load_all() (*in module pymrio*), 141
load_test() (*in module pymrio*), 131

M

match() (*pymrio.Extension method*), 150
match() (*pymrio.IOSystem method*), 148
modification_history (*pymrio.MRIOMetaData property*), 161
MRIOMetaData (*class in pymrio*), 160

N

note() (*pymrio.MRIOMetaData method*), 161
note_history (*pymrio.MRIOMetaData property*), 162

P

parse_eora26() (*in module pymrio*), 137
parse_exio12_ext() (*in module pymrio*), 167
parse_exiobase1() (*in module pymrio*), 134
parse_exiobase2() (*in module pymrio*), 135
parse_exiobase3() (*in module pymrio*), 135
parse_oecd() (*in module pymrio*), 138
parse_wiod() (*in module pymrio*), 136
plot_account() (*pymrio.Extension method*), 171

R

remove_extension() (*pymrio.IOSystem method*), 167
rename_regions() (*pymrio.IOSystem method*), 168
rename_sectors() (*pymrio.IOSystem method*), 169
rename_Y_categories() (*pymrio.IOSystem method*), 169
report_accounts() (*pymrio.IOSystem method*), 171
reset_to_coefficients() (*pymrio.IOSystem method*), 176
reset_to_flows() (*pymrio.IOSystem method*), 175

S

save() (*pymrio.IOSystem method*), 139
save() (*pymrio.MRIOMetaData method*), 162
save_all() (*pymrio.IOSystem method*), 139
set_index() (*pymrio.IOSystem method*), 146