pymrio Documentation

Release 0.3.dev1

Konstantin Stadler

Mar 12, 2018

Contents

Contents:

CHAPTER 1

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 occurr
- 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/konstantinstadler/pymrio

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

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

or

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

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_fp
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 and tutorials 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 and EORA26 as well as tutorials for the handling, aggregating

and analysis of these databases.

1.5 Contributing

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

1.6 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).

CHAPTER 2

Installation

pymrio is registered at PyPI. Install it by:

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

Alternativly, clone the source repository at: http://www.worldmrio.com/simplified/

CHAPTER 3

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
FY	factor production final demand	yF		thats kind of a weird name - maybe we can find a better: basically that are the extensions (e.g. emissions) of the final demand
S	factor production coefficients	D	stressor coefficients	
SY	factor production coefficients final demand	DY, yD		
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 accouts	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		
pxp	iosystem products x products	mxm		
ixi	iosystem industries x industries	nxn		

CHAPTER 4

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 prefered. This was generally the case when numpy broadcasting was available for a specific operation, resulting in a substaintal speed up of the calculations. In this cases the original formula remains as comment in the source code.

4.1 Basic MRIO calculations

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

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

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

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:

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

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

with the factor of production coefficients S given by

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

If the factor of production represent required environmental impacts, these can also occur during the final use phase. In that case G describe the impacts associated with final demand.

The production based accounts (direct territorial requirements) per country are than given by:

$$D_{pba} = Fe + Ge$$

Multipliers for F are obtained by

$$M = SL$$

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

$$D_{cba} = My$$

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

$$D_{imp} = SMY_t$$

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

$$D_{exp} = \widehat{SMY_te}$$

The total requirement for each country can be obtained by summing over the sectors for each account (D_{cba}, D_{imp} and D_{exp}). In case of D_{cba} any impacts associated with the use (G) must be added. Using that approach, footprints for each country i satisfy:

$$D^i_{cba} = D^i_{pba} + D^i_{imp} - D^i_{exp}$$

4.2 Aggregation

For the aggregation of the MRIO system the matrix S_k defines the aggregation matrix for regions and S_n the aggregation matrix for sectors.

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

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

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

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

$$S = S_k \otimes S_n$$

The aggregated IO system can than be obtained by

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

and

$$Y_{agg} = SY(S_k \otimes \mathrm{I})^{\mathrm{T}}$$

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

Factor of production are aggregated by

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

and final demand impacts by

$$G_{agg} = G(S_k \otimes \mathrm{I})^{\mathrm{T}}$$

CHAPTER 5

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 WIOD and Eora26.

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

5.1 WIOD download

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:

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

Define a folder for storing the data

```
In [2]: wiod_folder = '/tmp/mrios/WIOD2013'
```

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

```
In [3]: 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:

```
In [4]: print(wiod_meta)
```

```
Description: WIOD metadata file for pymrio
MRIO Name: WIOD
System: ixi
Version: data13
File: /tmp/mrios/WIOD2013/metadata.json
```

```
History:
20180111 10:11:06 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/water/wat
20180111 10:11:05 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/materials
20180111 10:11:05 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/land/lan_
20180111 10:11:04 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/AIR/AIR_m
20180111 10:11:03 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/CO2/CO2_m
20180111 10:11:02 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/EM/EM_may
20180111 10:11:02 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/EU/EU_may
20180111 10:11:01 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/SEA/WIOD_
20180111 10:11:00 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/update_se
20180111 10:10:58 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
 ... (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):

```
In [5]: res_years = [97,2004,'2005']
```

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

```
In [6]: from pymrio.tools.iodownloader import WIOD_CONFIG

In [7]: WIOD_CONFIG

Out[7]: {'satellite_urls': ['http://www.wiod.org/protected3/data13/SEA/WIOD_SEA_July14.x
           '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'],
          'url_db_content': 'http://www.wiod.org/',
          'url_db_view': 'http://www.wiod.org/database/wiots13'}
```

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

```
In [8]: sat_accounts = ['EU', 'CO2']
        res_satellite = [sat for sat in WIOD_CONFIG['satellite_urls']
                         if any(acc in sat for acc in sat_accounts)]

In [9]: res_satellite

Out[9]: ['http://www.wiod.org/protected3/data13/EU/EU_may12.zip',
         'http://www.wiod.org/protected3/data13/CO2/CO2_may12.zip']

In [10]: wiod_meta_res = pymrio.download_wiod2013(storage_folder='/tmp/foo_folder/WIOD20
                                                  years=res_years,
                                                  satellite_urls=res_satellite)

In [11]: 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:
20180111 10:22:41 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/CO2/CO2_m
20180111 10:22:41 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/EU/EU_may
```

```
20180111 10:22:40 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:38 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:37 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
```

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

```
In [12]: additional_years = [2000, 2001]
         wiod_meta_res = pymrio.download_wiod2013(storage_folder='/tmp/foo_folder/WIOD20
                                                  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.

```
In [13]: 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:
20180111 10:22:46 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:45 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:41 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/CO2/CO2_m
20180111 10:22:41 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/EU/EU_may
20180111 10:22:40 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:38 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
20180111 10:22:37 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/wiot_anal
```

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

5.2 Eora26 download

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

The Eora website (worldmrio) states that Eora is free for use at degree-granting academic institutions. All other users must license the data. Further information can be obtained by contacting info@worldmrio.com .

Prior to the access to the Eora database, you have to agree to this license agreement and commit to cite Lenzen et al. 2012 and Lenzen et al. 2013.

The same applies when using the automatic downloader and you have to explicitly confirm to these requirements during the download.

Setup the download with

```
In [14]: import pymrio
         eora_folder = '/tmp/mrios/eora26'
```

Start the download with (this can take some minutes). Before the download starts, the function will ask you to agree with the conditions stated on the Eora website (worldmrio).

```
In [15]: eora_meta = pymrio.download_eora26(storage_folder=eora_folder, prices=['bp'])
```

```
The Eora MRIO is free for academic (university or grant-funded) work at degree-granting

 When using Eora, the Eora authors ask you cite these publications:
```

```
 Lenzen, M., Kanemoto, K., Moran, D., Geschke, A. Mapping the Structure of the World Eco

 Lenzen, M., Moran, D., Kanemoto, K., Geschke, A. (2013) Building Eora: A Global Multi-r


Do you agree with these conditions [y/n]: y

In [16]: print(eora_meta)

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/Loc
20180111 10:26:26 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:26:17 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:25:57 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:25:47 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:25:37 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:25:21 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:25:08 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:24:58 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
20180111 10:24:46 - FILEIO -  Downloaded http://worldmrio.com/ComputationsM/Phase199/Loc
 ... (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.

In the default case, the tables in basic prices are downloaded. To catch the purchaser price tables pass `prices='pp'` or `prices=['bp', 'pp']` to catch both price systems.

5.3 EXIOBASE download

EXIOBASE 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.*

CHAPTER 6

Metadata and change 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':

```
In [1]: import pymrio
        io = pymrio.load_test()

In [2]: io.meta

Out[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:
        20180111 11:02:58 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/pym
        20171024 12:11:47 - FILEIO -  Created metadata file ../test_mrio/metadata.json

In [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:
20180111 11:02:58 - NOTE -  Loaded the pymrio test sytem
20180111 11:02:58 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/pymrio/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:

```
In [4]: io.calc_all()
        io.aggregate(region_agg = 'global')

Out[4]: <pymrio.core.mriosystem.IOSystem at 0x7fc4f1a4d518>

In [5]: io.meta
```

```
Out[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:
        20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissions
        20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor_in
        20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand
        20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
        20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
        20180111 11:03:02 - MODIFICATION -  Aggregate population vector
        20180111 11:03:02 - MODIFICATION -  Aggregate industry output x
        20180111 11:03:02 - MODIFICATION -  Aggregate transaction matrix Z
        20180111 11:03:02 - MODIFICATION -  Aggregate final demand y
        20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissions
         ... (more lines in history)
```

Notes can added at any time:

```
In [6]: io.meta.note('First round of calculations finished')

In [7]: io.meta

Out[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:
        20180111 11:03:09 - NOTE -  First round of calculations finished
        20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissions
        20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor_in
        20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand
        20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
        20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
        20180111 11:03:02 - MODIFICATION -  Aggregate population vector
        20180111 11:03:02 - MODIFICATION -  Aggregate industry output x
        20180111 11:03:02 - MODIFICATION -  Aggregate transaction matrix Z
        20180111 11:03:02 - MODIFICATION -  Aggregate final demand y
         ... (more lines in history)
```

In addition, all file io operations are recorde in the meta data:

```
In [8]: io.save_all('/tmp/foo')

Out[8]: <pymrio.core.mriosystem.IOSystem at 0x7fc4f1a4d518>

In [9]: io_new = pymrio.load_all('/tmp/foo')

In [10]: io_new.meta

Out[10]: Description: test mrio for pymrio
         MRIO Name: testmrio
         System: pxp
         Version: v1
         File: /tmp/foo/metadata.json
         History:
         20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/factor_inpu
         20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/emissions
         20180111 11:03:12 - FILEIO -  Loaded IO system from /tmp/foo
         20180111 11:03:12 - FILEIO -  Saved testmrio to /tmp/foo
```

```
          20180111 11:03:09 - NOTE -  First round of calculations finished
          20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissior
          20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor_i
          20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand
          20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
          20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
           ... (more lines in history)
```

The top level meta data can be changed as well. These changes will also be recorded in the history:

```
In [11]: io_new.meta.change_meta('Version', 'v2')

In [12]: io_new.meta

Out[12]: Description: test mrio for pymrio
         MRIO Name: testmrio
         System: pxp
         Version: v2
         File: /tmp/foo/metadata.json
         History:
         20180111 11:03:13 - METADATA_CHANGE -  Changed parameter "version" from "v1" to
         20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/factor_inpu
         20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/emissions
         20180111 11:03:12 - FILEIO -  Loaded IO system from /tmp/foo
         20180111 11:03:12 - FILEIO -  Saved testmrio to /tmp/foo
         20180111 11:03:09 - NOTE -  First round of calculations finished
         20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissior
         20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor_i
         20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand
         20180111 11:03:02 - MODIFICATION -  Aggregate extensions...
          ... (more lines in history)
```

To get the full history list, use:

```
In [13]: io_new.meta.history

Out[13]: ['20180111 11:03:13 - METADATA_CHANGE -  Changed parameter "version" from "v1"
          '20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/factor_in
          '20180111 11:03:12 - FILEIO -  Added satellite account from /tmp/foo/emissions
          '20180111 11:03:12 - FILEIO -  Loaded IO system from /tmp/foo',
          '20180111 11:03:12 - FILEIO -  Saved testmrio to /tmp/foo',
          '20180111 11:03:09 - NOTE -  First round of calculations finished',
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissi
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor
          '20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand',
          '20180111 11:03:02 - MODIFICATION -  Aggregate extensions...',
          '20180111 11:03:02 - MODIFICATION -  Aggregate extensions...',
          '20180111 11:03:02 - MODIFICATION -  Aggregate population vector',
          '20180111 11:03:02 - MODIFICATION -  Aggregate industry output x',
          '20180111 11:03:02 - MODIFICATION -  Aggregate transaction matrix Z',
          '20180111 11:03:02 - MODIFICATION -  Aggregate final demand y',
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissi
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor
          '20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand',
          '20180111 11:03:02 - MODIFICATION -  Leontief matrix L calculated',
          '20180111 11:03:02 - MODIFICATION -  Coefficient matrix A calculated',
          '20180111 11:03:02 - MODIFICATION -  Industry output x calculated',
          '20180111 11:02:58 - NOTE -  Loaded the pymrio test sytem',
          '20180111 11:02:58 - FILEIO -  Load test_mrio from /home/konstans/proj/pymrio/
          '20171024 12:11:47 - FILEIO -  Created metadata file ../test_mrio/metadata.jso
```

This can be restricted to one of the history types by:

```
In [14]: io_new.meta.modification_history
```

```
Out[14]: ['20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissi
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor
          '20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand',
          '20180111 11:03:02 - MODIFICATION -  Aggregate extensions...',
          '20180111 11:03:02 - MODIFICATION -  Aggregate extensions...',
          '20180111 11:03:02 - MODIFICATION -  Aggregate population vector',
          '20180111 11:03:02 - MODIFICATION -  Aggregate industry output x',
          '20180111 11:03:02 - MODIFICATION -  Aggregate transaction matrix Z',
          '20180111 11:03:02 - MODIFICATION -  Aggregate final demand y',
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension emissi
          '20180111 11:03:02 - MODIFICATION -  Calculating accounts for extension factor
          '20180111 11:03:02 - MODIFICATION -  Calculating aggregated final demand',
          '20180111 11:03:02 - MODIFICATION -  Leontief matrix L calculated',
          '20180111 11:03:02 - MODIFICATION -  Coefficient matrix A calculated',
          '20180111 11:03:02 - MODIFICATION -  Industry output x calculated']
```

or

```
In [15]: io_new.meta.note_history
```

```
Out[15]: ['20180111 11:03:09 - NOTE -  First round of calculations finished',
          '20180111 11:02:58 - NOTE -  Loaded the pymrio test sytem']
```

CHAPTER 7

Handling MRIO data

7.1 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

7.1.1 Preperation

Import pymrio

First import the pymrio module and other packages needed:

```
In [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 b 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.

```
In [2]: _sectors = ['sector1', 'sector2']
        _regions = ['reg1']
        _Z_multiindex = pd.MultiIndex.from_product(
                        [_regions, _sectors], names = [u'region', u'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.

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

```
In [4]: Z
```

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

Final demand is treated in the same way:

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

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

```
In [7]: Y
```

```
Out[7]: region                  reg1
        category       final demand
        region sector
        reg1   sector1          350
               sector2         1700
```

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

```
In [8]: F = pd.DataFrame(
            data = np.array([[650, 1400]]),
            index = ['Payments_sectors'],
            columns = _Z_multiindex)
```

```
In [9]: F
```

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

7.1.2 Include tables in the IOSystem object

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

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

Now we can add the tables to the IOSystem instance:

```
In [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):

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

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

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

```
In [14]: io.factor_input.unit = pd.DataFrame(data = ['USD'], index = F.index, columns =
```

We can check whats in the system:

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

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

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

7.1.3 Calculate the missing parts

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

Out[16]: <pymrio.core.mriosystem.IOSystem at 0x7ff1c473d0b8>
```

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):

```
In [17]: io.A

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

In [18]: io.L

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

7.1.4 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:

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

We copy the original IOSystem:

```
In [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)

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

```
Out[21]: <pymrio.core.mriosystem.IOSystem at 0x7ff1995a7390>
```

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

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

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

```
Out[23]: <pymrio.core.mriosystem.IOSystem at 0x7ff1995a7390>
```

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

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

```
Out[24]: region  sector
         reg1    sector1    2247.524752
                 sector2    3841.584158
         dtype: float64
```

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

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

```
Out[25]: Description: Metadata for pymrio
         MRIO Name: IO_copy
         System: None
         Version: None
         File: None
         History:
         20180119 09:33:33 - MODIFICATION -  Calculating accounts for extension factor_i
         20180119 09:33:33 - MODIFICATION -  Calculating aggregated final demand
         20180119 09:33:33 - MODIFICATION -  Flow matrix Z calculated
         20180119 09:33:33 - MODIFICATION -  Industry Output x calculated
         20180119 09:33:31 - MODIFICATION -  Reset full system to coefficients
         20180119 09:33:28 - NOTE -  IOSystem copy IO_copy based on IO
         20180119 09:33:15 - MODIFICATION -  Calculating accounts for extension factor_i
         20180119 09:33:15 - MODIFICATION -  Calculating aggregated final demand
         20180119 09:33:15 - MODIFICATION -  Leontief matrix L calculated
         20180119 09:33:15 - MODIFICATION -  Coefficient matrix A calculated
          ... (more lines in history)
```

7.2 Handling the WIOD EE MRIO database

7.2.1 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:

```
In [1]: import pymrio

In [2]: wiod_storage = '/tmp/mrios/WIOD2013'
```

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.

7.2.2 Parsing

Parsing a single year

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

```
In [3]: wiod2007 = pymrio.parse_wiod(year=2007, path=wiod_storage)
```

Which loads the specific year and extension data:

```
In [4]: wiod2007.Z.head()
```

```
Out[4]: 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
               19      0.495869    0.076813   0.185135   0.272204  0.000157
```

```
        [5 rows x 1435 columns]
```

```
In [5]: wiod2007.AIR.F
```

```
Out[5]: 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
        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:

```
In [6]: wiod2007.SEA.F
```

```
Out[6]: 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`:

```
In [7]: 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:
20180111 10:33:29 - FILEIO -  Extension wat parsed from /tmp/mrios/WIOD2013
20180111 10:33:28 - FILEIO -  Extension mat parsed from /tmp/mrios/WIOD2013
20180111 10:33:27 - FILEIO -  Extension lan parsed from /tmp/mrios/WIOD2013
20180111 10:33:26 - FILEIO -  Extension EU parsed from /tmp/mrios/WIOD2013
20180111 10:33:24 - FILEIO -  Extension EM parsed from /tmp/mrios/WIOD2013
20180111 10:33:23 - FILEIO -  Extension CO2 parsed from /tmp/mrios/WIOD2013
20180111 10:33:21 - FILEIO -  Extension AIR parsed from /tmp/mrios/WIOD2013
20180111 10:33:19 - FILEIO -  SEA file extension parsed from /tmp/mrios/WIOD2013
20180111 10:33:13 - FILEIO -  WIOD data parsed from /tmp/mrios/WIOD2013/wiot07_row_apr12
20180111 10:11:06 - FILEIO -  Downloaded http://www.wiod.org/protected3/data13/water/wat
```

```
... (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:

```
In [8]: wiod2007_full = pymrio.parse_wiod(year=2007, path=wiod_storage, names=('full', '
        wiod2007_full.Y.head()
```

```
Out[8]: region
        category                                          Final consumption expenditure
        region sector
        AUS    Agriculture, Hunting, Forestry and Fishing
               Mining and Quarrying
               Food, Beverages and Tobacco
               Textiles and Textile Products
               Leather, Leather and Footwear

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

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

        region
        category                                          Gross fixed capital formation
        region sector
        AUS    Agriculture, Hunting, Forestry and Fishing                  2924.034910
               Mining and Quarrying                                        4150.190757
               Food, Beverages and Tobacco                                  457.899386
               Textiles and Textile Products                                453.941827
               Leather, Leather and Footwear                                117.064525
```

```
region
category                                              Changes in inventories and val
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                          1280.
       Mining and Quarrying                                                -292.
       Food, Beverages and Tobacco                                          404.
       Textiles and Textile Products                                        -42.
       Leather, Leather and Footwear                                        -10.

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

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

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

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

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

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

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

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

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

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

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

region
category                                              Gross fixed capital formation
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 val
region sector
AUS    Agriculture, Hunting, Forestry and Fishing                              0.
       Mining and Quarrying                                                   -0.
       Food, Beverages and Tobacco                                             0.
       Textiles and Textile Products                                           0.
       Leather, Leather and Footwear                                          -0.

[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.

7.3 Working with the EXIOBASE EE MRIO database

7.3.1 Getting EXIOBASE

EXIOBASE 1 (developed in the fp6 project EXIOPOL) and EXIOBASE 2 (outcome of the fp7 project CREEA) are available on http://www.exiobase.eu

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

EXIOBASE 1

To download EXIOBASE 1 for the use with pymrio, navigate to https://www.exiobase.eu - 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 2 is available at ttp://www.exiobase.eu - 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 currently not publicly available. However, pymrio already includes a parser for the preliminary version. If you have access to this version, you can download the files as provided and use the preliminary pymrio exiobase3 parser. Manually adjustment might be needed depending on the available sub-version of EXIOBASE 3.

7.3.2 Parsing

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

For each publically available version of EXIOBASE pymrio provides a specific parser. To parse EXIOBASE 1 use:

```
In [2]: exio1 = pymrio.parse_exiobase1(path='/tmp/mrios/exio1/121016_EXIOBASE_pxp_ita_44
```

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

Similarly, EXIOBASE2 can be parsed by:

```
In [3]: exio2 = pymrio.parse_exiobase2(path='/tmp/mrios/exio2/mrIOT_PxP_ita_coefficient_
                                       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.

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

```
In [4]: del exio1
```

7.3.3 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:

```
In [5]: exio2.meta

Out[5]: Description: Metadata for pymrio
        MRIO Name: EXIOBASE
        System: pxp
        Version: 2.2.2
        File: None
        History:
        20180111 10:25:01 - FILEIO -  EXIOBASE data FY_materials parsed from /tmp/mrios/
        20180111 10:25:01 - FILEIO -  EXIOBASE data FY_emissions parsed from /tmp/mrios/
        20180111 10:25:01 - FILEIO -  EXIOBASE data S_resources parsed from /tmp/mrios/e
        20180111 10:25:00 - FILEIO -  EXIOBASE data S_materials parsed from /tmp/mrios/e
        20180111 10:24:59 - FILEIO -  EXIOBASE data S_emissions parsed from /tmp/mrios/e
        20180111 10:24:58 - FILEIO -  EXIOBASE data S_factor_inputs parsed from /tmp/mri
        20180111 10:24:58 - FILEIO -  EXIOBASE data Y parsed from /tmp/mrios/exio2/mrIOT
        20180111 10:24:57 - FILEIO -  EXIOBASE data A parsed from /tmp/mrios/exio2/mrIOT
```

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

```
In [6]: exio2.meta.note("First test run of EXIOBASE 2")
        exio2.meta

Out[6]: Description: Metadata for pymrio
        MRIO Name: EXIOBASE
        System: pxp
        Version: 2.2.2
        File: None
        History:
        20180111 10:25:02 - NOTE -  First test run of EXIOBASE 2
        20180111 10:25:01 - FILEIO -  EXIOBASE data FY_materials parsed from /tmp/mrios/
        20180111 10:25:01 - FILEIO -  EXIOBASE data FY_emissions parsed from /tmp/mrios/
        20180111 10:25:01 - FILEIO -  EXIOBASE data S_resources parsed from /tmp/mrios/e
        20180111 10:25:00 - FILEIO -  EXIOBASE data S_materials parsed from /tmp/mrios/e
        20180111 10:24:59 - FILEIO -  EXIOBASE data S_emissions parsed from /tmp/mrios/e
        20180111 10:24:58 - FILEIO -  EXIOBASE data S_factor_inputs parsed from /tmp/mri
        20180111 10:24:58 - FILEIO -  EXIOBASE data Y parsed from /tmp/mrios/exio2/mrIOT
        20180111 10:24:57 - FILEIO -  EXIOBASE data A parsed from /tmp/mrios/exio2/mrIOT
```

To check for sectors, regions and extensions:

```
In [7]: exio2.get_sectors()
```

```
Out[7]: 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)
```

```
In [8]: exio2.get_regions()
```

```
Out[8]: 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')
```

```
In [9]: list(exio2.get_extensions())
```

```
Out[9]: ['factor_inputs', 'emissions', 'materials', 'resources', 'impact']
```

7.3.4 Calculating the system and extension results

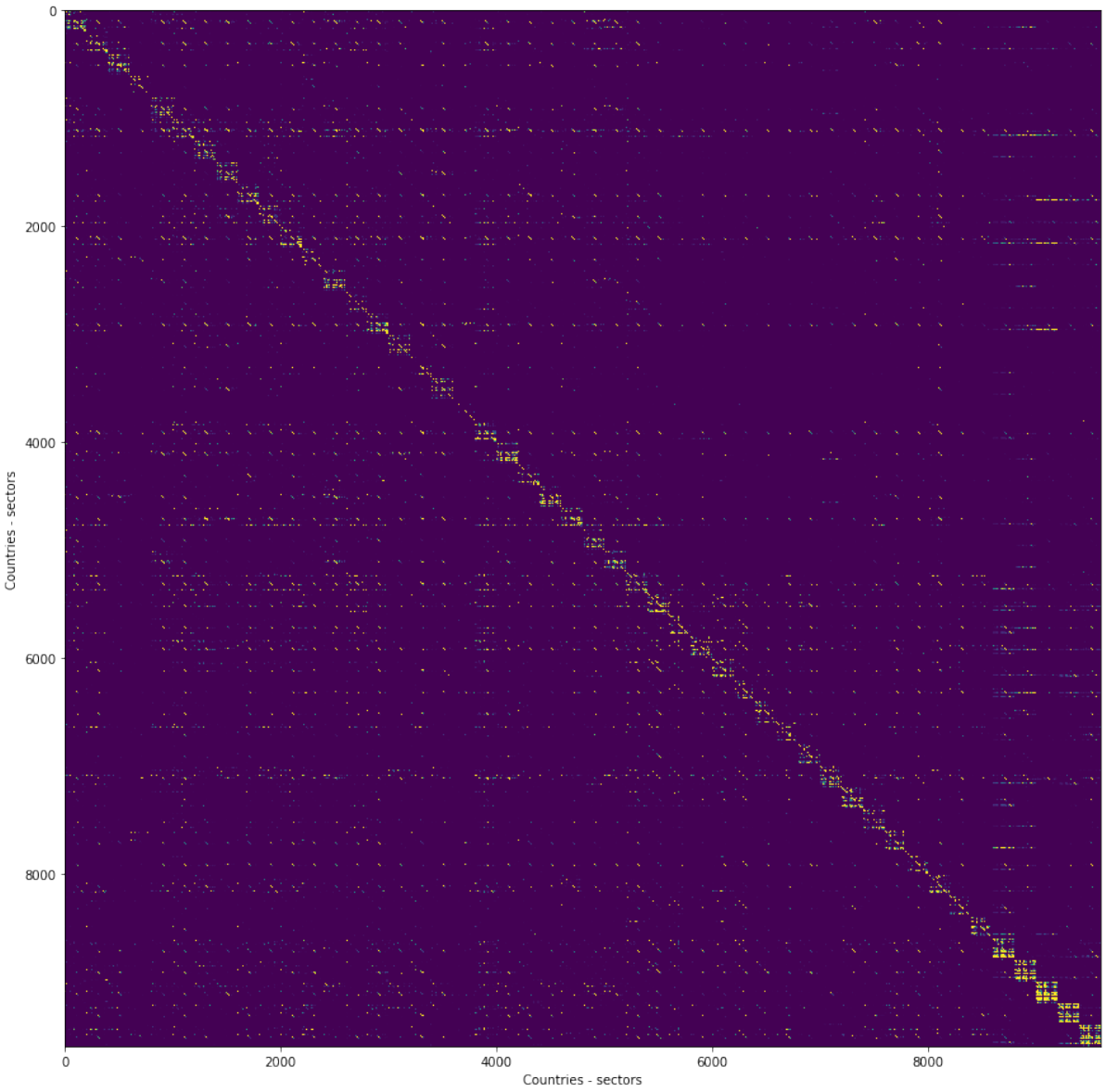
The following command checks for missing parts in the system and calculates them. In case of the parsed EXIOBASE this includes A, L, multipliers M, footprint accounts, ..

```
In [10]: exio2.calc_all()
```

```
Out[10]: <pymrio.core.mriosystem.IOSystem at 0x7f16909c1f98>
```

7.3.5 Exploring the results

```
In [11]: 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:

```
In [12]: list(exio2.impact.get_rows())
```

```
Out[12]: ['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)',
          '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)',
'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 ar
'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 ar
          '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 ar
          '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,
          'Marine aquatic ecotoxicity MAETP inf. (incl. PAH average, Xylene average, NMV
          'Terrestrial ecotoxicity TETP inf. (incl. PAH average, Xylene average, NMVOC a
          'global warming GWP100 (incl. NMVOC average)',
          'ozone layer depletion ODP steady state (incl. NMVOC average)',
          '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']
```

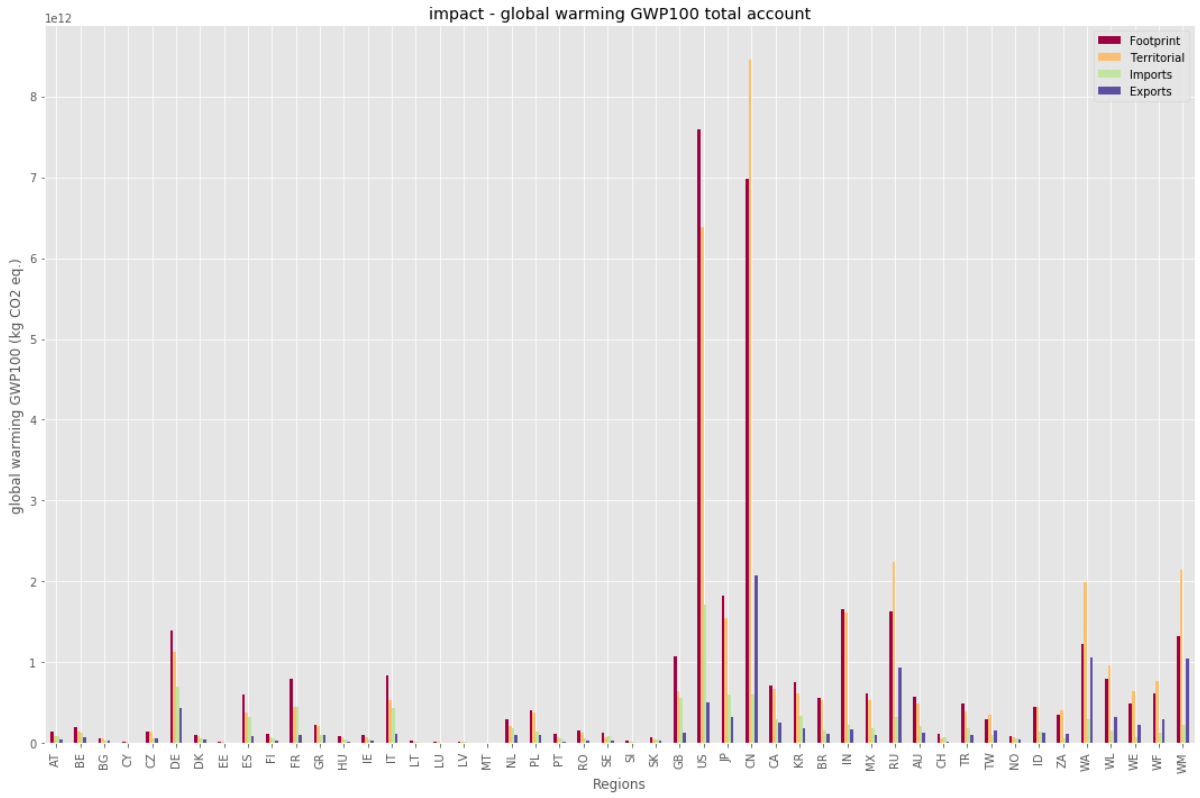
And to get for example the footprint of a specific impact do:

```
In [13]: 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

Out[13]: 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
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
```

7.3.6 Visualizing the data

```
In [14]: 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*.

7.4 Parsing the Eora26 EE MRIO database

7.4.1 Getting Eora26

The Eora 26 database is available at http://www.worldmrio.com . You can download these files with the pymrio automatic downloader as described at *Eora26 download*.

In the most simple case, you can get the full database in basic prices with (you need to agree to license conditions before download):

```
In [1]: import pymrio

In [2]: eora_storage = '/tmp/mrios/eora26'

In [3]: eora_meta = pymrio.download_eora26(storage_folder=eora_storage, prices=['bp'])
The Eora MRIO is free for academic (university or grant-funded) work at degree-granting

 When using Eora, the Eora authors ask you cite these publications:

 Lenzen, M., Kanemoto, K., Moran, D., Geschke, A. Mapping the Structure of the World Ecc

 Lenzen, M., Moran, D., Kanemoto, K., Geschke, A. (2013) Building Eora: A Global Multi-r


Do you agree with these conditions [y/n]: y
```

7.4.2 Parse

To parse a single year do:

```
In [4]: eora = pymrio.parse_eora26(year=2005, path=eora_storage)
```

```
/home/konstans/bin/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py:2530: Pe
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

7.4.3 Explore

Eora includes (almost) all countries:

```
In [5]: eora.get_regions()
```

```
Out[5]: 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.

```
In [6]: import country_converter as coco
```

```
In [7]: eora.aggregate(region_agg = coco.agg_conc(original_countries='Eora',
                                                  aggregates=['OECD'],
                                                  missing_countries='NON_OECD')
                      )
```

```
Out[7]: <pymrio.core.mriosystem.IOSystem at 0x7f4acfe64278>
```

```
In [8]: eora.get_regions()
```

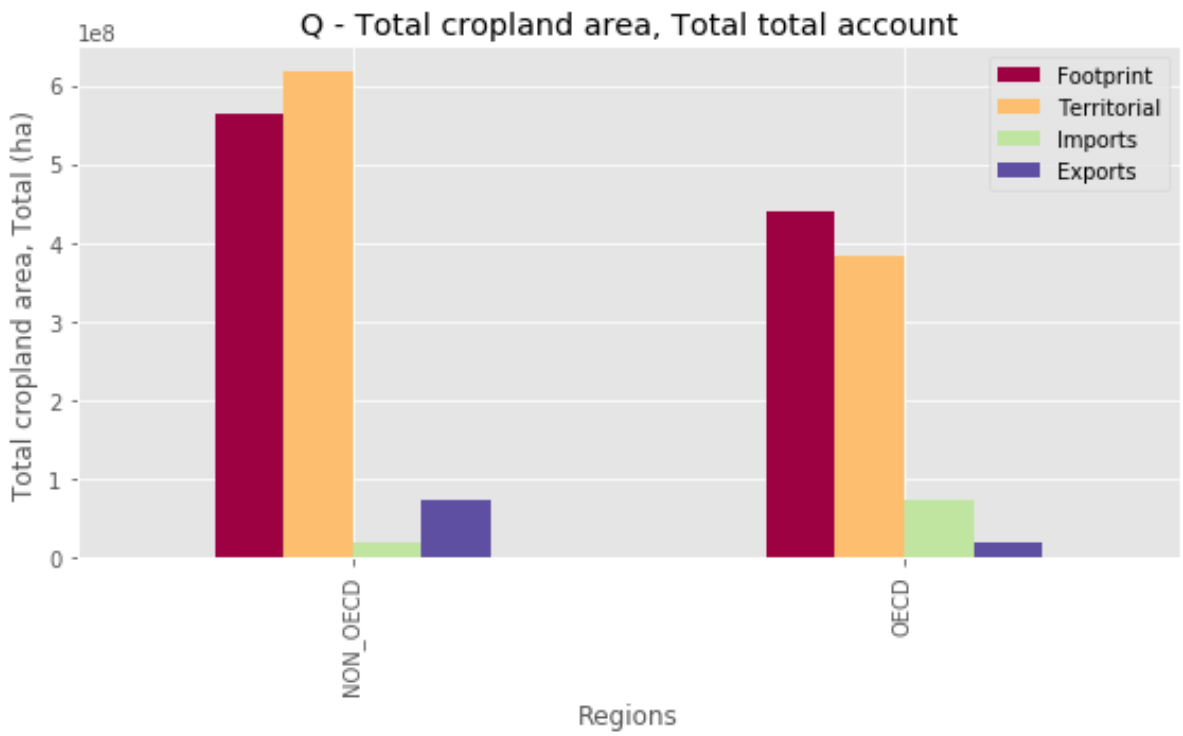
```
Out[8]: Index(['NON_OECD', 'OECD'], dtype='object', name='region')
```

```
In [9]: eora.calc_all()
```

```
Out[9]: <pymrio.core.mriosystem.IOSystem at 0x7f4acfe64278>
```

```
In [10]: import matplotlib.pyplot as plt
         with plt.style.context('ggplot'):
             eora.Q.plot_account(('Total cropland area', 'Total'), figsize=(8,5))
             plt.show()
```

```
/home/konstans/proj/pymrio/pymrio/core/mriosystem.py:886: PerformanceWarning: indexing p
  _data = pd.DataFrame(getattr(self, accounts[key]).ix[row].T)
```



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

7.5 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.

```
In [1]: import pymrio
        io = pymrio.load_test().calc_all()
```

7.5.1 Basic save and read

To save the full system, use:

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

```
Out[2]: <pymrio.core.mriosystem.IOSystem at 0x7fad28466390>
```

To read again from that folder do:

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

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

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

```
Out[4]: Description: test mrio for pymrio
        MRIO Name: testmrio
        System: pxp
        Version: v1
        File: /tmp/testmrio/full/metadata.json
        History:
        20180110 15:50:46 - FILEIO -  Added satellite account from /tmp/testmrio/full/fa
        20180110 15:50:46 - FILEIO -  Added satellite account from /tmp/testmrio/full/em
        20180110 15:50:46 - FILEIO -  Loaded IO system from /tmp/testmrio/full
        20180110 15:50:46 - FILEIO -  Saved testmrio to /tmp/testmrio/full
        20180110 15:50:46 - MODIFICATION -  Calculating accounts for extension emissions
        20180110 15:50:45 - MODIFICATION -  Calculating accounts for extension factor_in
        20180110 15:50:45 - MODIFICATION -  Calculating aggregated final demand
        20180110 15:50:45 - MODIFICATION -  Leontief matrix L calculated
        20180110 15:50:45 - MODIFICATION -  Coefficient matrix A calculated
        20180110 15:50:45 - MODIFICATION -  Industry output x calculated
         ... (more lines in history)
```

7.5.2 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.

```
In [5]: import os
        os.listdir(save_folder_full)
```

```
Out[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:

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

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

7.5.3 Storing or exporting a specific table or extension

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

```
In [7]: save_folder_em= '/tmp/testmrio/emissions'

In [8]: io.emissions.save(path=save_folder_em)

Out[8]: <pymrio.core.mriosystem.Extension at 0x7fad28485208>
```

This can than be loaded again as separate satellite account:

```
In [9]: emissions = pymrio.load(save_folder_em)

In [10]: emissions

Out[10]: <pymrio.core.mriosystem.Extension at 0x7fad18c9ecf8>

In [11]: emissions.D_cba
```

```
Out[11]: 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
         stressor       compartment                    ...
         emission_type1 air          19145.604911      ...       4.209505e+07
         emission_type2 water         3733.601474      ...       4.243738e+06

         region                                          reg6                \
         sector                               other      food        mining
         stressor       compartment
         emission_type1 air          1.138661e+07  1.517235e+07  1.345318e+06
         emission_type2 water        7.307208e+06  4.420574e+06  5.372216e+05

         region                                                                \
         sector                     manufactoring   electricity  construction
         stressor       compartment
         emission_type1 air          7.145075e+07  3.683167e+07  1.836696e+06
         emission_type2 water        1.068144e+07  5.728136e+05  9.069515e+05

         region
         sector                               trade     transport         other
         stressor       compartment
         emission_type1 air          4.241568e+07  4.805409e+07  3.602298e+07
         emission_type2 water        5.449044e+07  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:

```
In [12]: io.emissions.D_cba.to_excel('/tmp/testmrio/emission_footprints.xlsx')
```

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

7.6 Using the aggregation functionality of pymrio

Pymrio offers various possibilities to achieve an aggreation 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.

7.6.1 Loading the test mrio

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

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

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

```
Out[2]: <pymrio.core.mriosystem.IOSystem at 0x7fba581e2a20>
```

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

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

7.6.2 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.

```
In [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]
            ])
```

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

Out[5]: <pymrio.core.mriosystem.IOSystem at 0x7fba581e2a20>

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

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

In [7]: io.calc_all()

Out[7]: <pymrio.core.mriosystem.IOSystem at 0x7fba581e2a20>

In [8]: io.emissions.D_cba

Out[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:

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

In [10]: io.get_regions()

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

7.6.3 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:

```
In [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:

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

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

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

In [14]: io_vec_agg.emissions.D_cba_reg

Out[14]: region                          R1            R2
         stressor       compartment
```

```
emission_type1 air          6.690192e+08  1.686954e+09
emission_type2 water        5.337682e+08  5.902081e+08
```

7.6.4 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).

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

Independent aggregation

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

In [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')

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

Out[18]: <pymrio.core.mriosystem.IOSystem at 0x7fba500bb518>

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

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

This can be passed directly to pymrio:

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

Out[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:

```
In [21]: wiod_raw = '/tmp/mrios/WIOD2013'
```

We will parse the year 2000 and calculate the results:

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

and than 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:

```
In [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 than rename the regions to make the membership clearer:

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

Out[24]: <pymrio.core.mriosystem.IOSystem at 0x7fba395005f8>
```

To see the result for the air emission footprints:

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

Out[25]: region         OECDwoEU   EUwoGermany         Other           DEU
         stressor
         CO2       9.576199e+06  3.840406e+06  9.232742e+06  1.123772e+06
         CH4       6.066454e+07  3.134722e+07  1.487615e+08  7.953304e+06
         N2O       2.103103e+06  1.264400e+06  6.166586e+06  2.941486e+05
         NOX       3.730527e+07  1.385859e+07  5.103133e+07  3.164278e+06
         SOX       3.362054e+07  1.239562e+07  5.137882e+07  2.045926e+06
         CO        1.916016e+08  5.951296e+07  4.424992e+08  1.296816e+07
         NMVOC     3.423713e+07  1.536753e+07  8.186918e+07  2.870176e+06
         NH3       5.453330e+06  3.867825e+06  1.674807e+07  8.656818e+05
```

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

7.6.5 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.

```
In [26]: pymrio.load_test().calc_all().aggregate(region_agg='global', sector_agg='total'

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

7.6.6 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:

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

In [28]: io_pre.emissions.D_cba

Out[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

In [29]: io_post.emissions.D_cba

Out[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:

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

Out[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
```

7.7 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.

7.7.1 Basic example

First we load the test mrio:

```
In [1]: import pymrio
        io = pymrio.load_test()
```

The test mrio includes several extensions:

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

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

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

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

```
Out[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
        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
        stressor       compartment                           ...
        emission_type1 air          1697937.30  347378.150     ...     42299319
        emission_type2 water         204835.44   29463.944     ...      4199841

        region                                      reg6                            \
        sector                           other        food     mining manufactoring
        stressor       compartment
        emission_type1 air          10773826.0  15777996.0  6420955.5   113172450.0
        emission_type2 water         7191006.3   4826108.1  1865625.1    12700193.0

        region                                                                        \
        sector                    electricity construction     trade   transport
        stressor       compartment
        emission_type1 air          56022534.0    4861838.5  18195621  47046542.0
        emission_type2 water          753213.7    2699288.3  13892313   8765784.3

        region
        sector                          other
        stressor       compartment
        emission_type1 air          21632868
        emission_type2 water        16782553

        [2 rows x 48 columns]
```

```
In [4]: et1_diag = io.emissions.diag_stressor(('emission_type1', 'air'), name = 'emtype1
```

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:

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

```
Out[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
               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
       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:

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

Finally we can calulate the all stressor accounts with:

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

```
Out[7]: <pymrio.core.mriosystem.IOSystem at 0x7f251808f518>
```

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).

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

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

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

```
Out[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
```

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

```
Out[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:

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

```
Out[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

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

Out[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

         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
```

7.7.2 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:

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

Out[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
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:

```
In [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'))

Out[14]: <pymrio.core.mriosystem.IOSystem at 0x7f251808f518>

In [15]: io.et1_diag.D_cba

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

7.8 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.

7.8.1 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 material to one total account. We assume, that the WIOD system is available at

```
In [1]: wiod_folder = '/tmp/mrios/WIOD2013'
```

To get started we import pymrio

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

For the example here, we use the data from 2009:

```
In [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:

```
In [4]: wiod09.mat.F
```

```
Out[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
        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
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 specify the groups which should be grouped by Pandas. Pymrio contains a helper function which builds such matching dictionary. The matching can also include regular expression to simplify the build:

```
In [5]: groups = wiod09.mat.get_index(as_dict=True, grouping_pattern = {'.*_Used': 'Mate
                                                                        '.*_Unused': 'Ma
        groups
Out[5]: {'Biomass_animals_Unused': 'Material Unused',
```

```
'Biomass_animals_Used': 'Material Used',
'Biomass_feed_Unused': 'Material Unused',
'Biomass_feed_Used': 'Material Used',
'Biomass_food_Unused': 'Material Unused',
'Biomass_food_Used': 'Material Used',
'Biomass_forestry_Unused': 'Material Unused',
'Biomass_forestry_Used': 'Material Used',
'Biomass_other_Unused': 'Material Unused',
'Biomass_other_Used': 'Material Used',
'Fossil_coal_Unused': 'Material Unused',
'Fossil_coal_Used': 'Material Used',
'Fossil_gas_Unused': 'Material Unused',
'Fossil_gas_Used': 'Material Used',
'Fossil_oil_Unused': 'Material Unused',
'Fossil_oil_Used': 'Material Used',
'Fossil_other_Unused': 'Material Unused',
'Fossil_other_Used': 'Material Used',
'Minerals_construction_Unused': 'Material Unused',
'Minerals_construction_Used': 'Material Used',
'Minerals_industrial_Unused': 'Material Unused',
'Minerals_industrial_Used': 'Material Used',
'Minerals_metals_Unused': 'Material Unused',
'Minerals_metals_Used': 'Material Used',
'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:

```
In [6]: wiod09.mat_agg = wiod09.mat.copy(new_name='Aggregated matrial accounts')
```

Then, we use the pyrio 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.

```
In [7]: for df_name, df in zip(wiod09.mat_agg.get_DataFrame(data=False, with_unit=True,
                               wiod09.mat_agg.get_DataFrame(data=True, with_unit=True, w
            if df_name == 'unit':
                wiod09.mat_agg.__dict__[df_name] = df.groupby(groups).apply(lambda x: '
            else:
                wiod09.mat_agg.__dict__[df_name] = df.groupby(groups).sum()

In [8]: wiod09.mat_agg.F

Out[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]
```

```
In [9]: wiod09.mat_agg.unit
```

```
Out[9]: Material Unused    1000 tonnes
        Material Used      1000 tonnes
        Total              1000 tonnes
        dtype: object
```

7.8.2 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:

```
In [10]: tt = pymrio.load_test()
         tt.emissions.get_index(as_dict=True)
```

```
Out[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.:

```
In [11]: agg_groups = {('emis.*', '.*'): 'all emissions'}
```

```
In [12]: group_dict = tt.emissions.get_index(as_dict=True,
                                              grouping_pattern=agg_groups)
         group_dict
```

```
Out[12]: {('emission_type1', 'air'): 'all emissions',
          ('emission_type2', 'water'): 'all emissions'}
```

Which can than be used to aggregate the satellite account:

```
In [13]: for df_name, df in zip(tt.emissions.get_DataFrame(data=False, with_unit=True, w
                                tt.emissions.get_DataFrame(data=True, with_unit=True, wi
             if df_name == 'unit':
                 tt.emissions.__dict__[df_name] = df.groupby(group_dict).apply(lambda x:
             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:

```
In [14]: import pandas as pd
         tt.emissions.set_index(pd.Index(tt.emissions.get_index(), name='stressor'))
```

```
In [15]: tt.emissions.F
```

```
Out[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
stressor                        ...
all emissions  376842.094       ...     46499160  17964832.3  20604104.1

region                                                                    \
sector            mining manufactoring electricity construction     trade
stressor
all emissions  8286580.6   125872643.0  56775747.7    7561126.8  32087934

region
sector          transport     other
stressor
all emissions  55812326.3  38415421

[1 rows x 48 columns]
```

CHAPTER 8

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.

Independent of your contribution, please use pull requests to inform me about any improvements you did and make sure all tests pass (see below).

8.1 Working on the documentation

The documentation of pymrio is currently not complete, any contribution to the description of pymrio is of huge value! I also 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

If you plan any changes to the source code of this repo, please first discuss the change you wish to make via a filing an issue (labelled Enhancement or Bug) before making a change. All code contribution must be provided as pull requests connected to a filed issue. Use numpy style docstrings and follow pep8 style guide. The latter is a requirement to pass the tests before merging a pull request. 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 with the pytest-pep8 extension for testing. To run the tests install these two packages (and the Pandas dependency) and run

```
py.test -v -pep8
```

in the root of your local copy of pymrio.

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 acitive deveopment. Open points include:

- parser for other available MRIOs
  - OPEN:EU (http://www.oneplaneteconomynetwork.org/)
  - OECD MRIO
- 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

- improving the documentation (of course. . .)

CHAPTER 9

Changelog

9.1 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.

9.2 v0.3.5 (Jan 17, 2018)

Added xlrd to requirements

9.3 v0.3.4 (Jan 12, 2018)

9.3.1 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

9.4 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

9.4.1 Dependencies

- pandas minimal version changed to 0.22
- Optional (for aggregation): country converter coco >= 0.6.3

9.4.2 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.

9.5 v0.2.2 (May 27, 2016)

9.5.1 Dependencies

- pytest. For the unit tests.

9.5.2 Misc

- Fixed filename error for the test system.
- Various small bug fixes.
- Preliminary EXIOBASE 3 parser.
- Preliminary World Input-Output Database (WIOD) parser.

9.6 v0.2.1 (Nov 17, 2014)

9.6.1 Dependencies

- pandas version > 0.15. This required some change in the xls reading within the parser.
- pytest. For the unit tests.

9.6.2 Misc

- Unit testing for all mathematical functions and a first system wide check.
- Fixed some mistakes in the tutorials and readme

9.7 v0.2.0 (Sept 11, 2014)

9.7.1 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")

9.7.2 Misc

- Several bugfixes
- Mainmodule split into several packages and submodules
- Added 3rd tutorial
- Added CHANGELOG

9.8 v0.1.0 (June 20, 2014)

Initial version

CHAPTER 10

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, FY. 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

Returns

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 EORA26 database (EXIOBASE requires registration before downloading).

download_wiod2013(storage_folder[, years, ...])	Downloads the 2013 wiod release
download_eora26(storage_folder[, years, ...])	Downloads Eora 26

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

pymrio.download_eora26

pymrio.**download_eora26**(*storage_folder, years=None, prices=['bp'], overwrite_existing=False*)

Downloads Eora 26

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.

prices: list of str If bp (default), download basic price tables. If pp, download purchaser prices. ['bp', 'pp'] possible.

overwrite_existing: boolean, optional If False, skip download of file already existing in the storage folder (default). Set to True to replace files.

10.1.3 Raw data

parse_exiobase2(path[, charact, popvector])	Parse the exiobase 2.2.2 source files for the IOSystem
parse_wiod(path[, year, names, popvector])	Parse the wiod source files for the IOSystem

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*) – 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_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*) – 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.

- **TODO popvector** (*TO BE IMPLEMENTED (consistent with EXIOBASE)*)

Yields *IOSystem*

Raises `ParserError` – If the WIOD source file are not complete or inconsistent

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, ...])	Developing version for saving with json instead of ini for meta
IOSystem.save_all(path[, table_format, sep, ...])	Saves the system and all extensions

pymrio.IOSystem.save

`IOSystem.`**`save`**(*path*, *table_format='txt'*, *sep='\t'*, *table_ext=None*, *float_format='%.12g'*)

Developing version for saving with json instead of ini for meta

Parameters

- **path** (*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', 'bin', 'binary'
  - 'txt' : Text files (default), alias: 'text', 'csv'
- **table_ext** (*string, optional*) – File extension, default depends on table_format(.pkl for pickle, .txt for text)
- **sep** (*string, optional*) – Field delimiter for the output file, only for txt files. Default: tab (' ')
- **float_format** (*string, optional*) – Format for saving the DataFrames, default = '%.12g', only for txt files

pymrio.IOSystem.save_all

`IOSystem.`**`save_all`**(*path*, *table_format='txt'*, *sep='\t'*, *table_ext=None*, *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

10.1.5 Load processed data

This functions load IOSystems or individual extensions which have been saved with pymrio before.

load(path[, include_core])	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*)

Loads a IOSystem or Extension previously saved with pymrio

This function can be used to load a IOSystem or Extension specified in a ini file. DataFrames (tables) are loaded from text or binary pickle files. For the latter, the extension .pkl or .pickle is assumed, in all other case the tables are assumed to be in .txt format.

Parameters

- **path** (*string*) – path or ini file name for the data to load
- **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.

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*)

Loads a full IO system with all extension in path

By default, all subfolders 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 to subfolders giving the names of these.

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([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.

Returns

Return type DataFrames or string generator, depending on parameter data

pymrio.IOSystem.get_extensions

IOSystem.**get_extensions**(*data=False*)
Yields the extensions or their names

Parameters **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)

Returns

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

Parameters

- **row** (*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.

Returns

Return type dict object with the data (pandas DataFrame)for the specific rows

10.2 Exploring the IO System

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
Extension.get_rows()	Returns the name of the rows of the extension

10.2.1 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

10.2.2 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

10.2.3 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

10.2.4 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.

10.2.5 pymrio.IOSystem.set_index

IOSystem.**set_index**(*index*)

Sets the pd dataframe index of all dataframes in the system to index

10.2.6 pymrio.Extension.get_rows

Extension.**get_rows**()

Returns the name of the rows of the extension

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_agg[, L, population])	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, x, A, L and calculates all which are None

pymrio.Extension.calc_system

Extension.**calc_system**(*x*, *Y_agg*, *L=None*, *population=None*)
Calculates the missing part of the extension plus accounts

This method uses y aggregated across specified y categories

Calculates:

- **for each sector and country:** D, 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_agg** (*pandas.DataFrame or np.array*) – The final demand aggregated (one category per country)
- **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, nr_sectors)	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

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

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

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

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
- **TODO - this must be completely redone (D, check for dataframe, . . .)**

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*, *nr_sectors*)

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
- **nr_sectors** (*int*) – Number of sectors in the MRIO

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*, *logger_function=<function info>*)

__init__(*location=None*, *description=None*, *name=None*, *system=None*, *version=None*, *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.

- **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'.
- **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

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

MRIOMetaData.**history**

All recorded history

10.4.4 pymrio.MRIOMetaData.modification_history

MRIOMetaData.**modification_history**

All modification history entries

10.4.5 pymrio.MRIOMetaData.note_history

MRIOMetaData.**note_history**

All note history entries

10.4.6 pymrio.MRIOMetaData.file_io_history

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 **filename** (*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 based on various classifications

pymrio.IOSystem.aggregate

IOSystem.**aggregate**(*region_agg=None*, *sector_agg=None*, *region_names=None*, *sector_names=None*, *inplace=True*, *pre_aggregation=False*)
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 'orignal' 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 based on various 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.

```
>>> 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']
```

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

Return type list (aggregation vector)

10.5.2 Analysing the source of impacts

Extension.diag_stressor(stressor[, name])	Diagonalize one row of the stressor matrix for a flow analysis.

pymrio.Extension.diag_stressor

Extension.**diag_stressor**(*stressor*, *name=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 flow, direct household emissions (FY) 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)

Returns

Return type Extension

10.5.3 Changing extensions

IOSystem.remove_extension([ext])	Remove extension from IOSystem
parse_exio_ext(ext_file, index_col, name[, ...])	Parse an EXIOBASE like extension file into pymrio.Extension

pymrio.IOSystem.remove_extension

IOSystem.**remove_extension**(*ext=None*)

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 (the latter will be checked if no instance was found) If ext is None (default) all Extensions will be removed

pymrio.parse_exio_ext

pymrio.**parse_exio_ext**(*ext_file*, *index_col*, *name*, *drop_compartment=True*, *version=None*, *year=None*, *iosystem=None*, *sep=', '*)

Parse an EXIOBASE 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).

Notes

So far this only parses factor of production extensions F (not final demand extensions FY nor coeffiecents S).

Parameters

- **ext_file** (*string*) – 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

10.5.4 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

10.6 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

Returns

Return type Axis as given by pandas.DataFrame.plot, None in case of errors

10.8 Miscellaneous

IOSystem.reset_to_flows()	Keeps only the absolute values.
IOSystem.reset_to_coefficients()	Keeps only the coefficient.
IOSystem.copy([new_name])	Returns a deep copy of the system

10.8.1 pymrio.IOSystem.reset_to_flows

IOSystem.**reset_to_flows**()

Keeps only the absolute values.

This removes all attributes which can not be aggregated and must be recalculated after the aggre-

gation.

10.8.2 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.

10.8.3 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 11

Indices and tables

- genindex
- modindex
- search

Index

S