Euromod Connector

Release 0.2.4

Belousova Irina, Serruys Hannes

Sep 27, 2024

CONTENTS

The Euromod Connector for Python is built to facilitate and simplify the usage of the EUROMOD microsimulation model for research and analysis purposes.

EUROMOD is a tax-benefit microsimulation model for the European Union that enables researchers and policy analysts to calculate, in a comparable manner, the effects of taxes and benefits on household incomes and work incentives for the population of each country and for the EU as a whole. It is a static microsimulation model that applies user-defined tax and benefit policy rules to harmonised microdata on individuals and households, calculates the effects of these rules on household income.

CHAPTER

ONE

CONTENTS

1.1 Examples

1.1.1 Loading and navigating the model

We start with importing the euromod package and creating a Model object from a EUROMOD model.

See also:

See the installation guide on how to install the package and its dependencies.

```
from euromod import Model
mod=Model(r"C:\EUROMOD_RELEASES_I6.0+")
mod
```

```
------------------------------
Model
------------------------------
         countries: 28 elements
         extensions: 11 elements
         model_path: 'C:\\EUROMOD_RELEASES_I6.0+'
```

Note that every object that is related to the EUROMOD project comes with an informative description. Here we can see that the model has 3 relevant attributes to the user:

- countries
- extensions
- model_path

The countries and extensions attributes contain elements of the respective objects. If we take a look at `countries`:

```
mod.countries
```

```
0: AT
1: BE
2: BG
3: CY
4: CZ
5: DE
6: DK
7: EE
```

(continues on next page)

(continued from previous page)

```
8: EL
9: ES
10: FI
11: FR
12: HR
13: HU
14: IE
15: IT
16: LT
17: LU
18: LV
19: MT
20: NL
21: PL
22: PT
23: RO
24: SE
25: SI
26: SK
27: SL
```

We see indeed that the euromod model contains 28 countries. In a similar fashion we can look what kind of extensions are stored in the model. The countries container can be indexed by both the number of the element and the country shortcode. Let us take a look at Sweden.

```
mod.countries["SE"]
```

```
------------------------------
Country
------------------------------
         datasets: 27 elements
         extensions: 12 elements
         local_extensions: COVID
         name: 'SE'
         policies: 26 elements
         systems: 18 elements
```

Here we see again an informative representation of the Country object, which contains several attributes that can be accessed. We can for example take a look at the first 10 policies that are stored in the country.

```
mod.countries["SE"].policies[0:10]
```

```
0: setdefault_se     |                                    |    DEF: SET DEFAULT
1: uprate_se         |                                    |    DEF: UPRATING FACTORS
2: ConstDef_se       |                                    |    DEF: CONSTANTS
3: IlsDef_se         |                                    |    DEF: INCOME CONCEPTS␣
↪(standardized)
4: IlsUDBdef_se      |                                    |    DEF: INCOME CONCEPTS (UDB)
5: ildef_se          |                                    |    DEF: INCOME CONCEPTS (non-
↪standardized)
6: random_se         |                                    |    DEF: Random assignment
7: TransLMA_se       |                                    |    DEF: Modelling labour␣
```

(continues on next page)

(continued from previous page)

```
↪market transitions (DO NOT S ...
8: tudef_se          |                                  |    DEF: ASSESSMENT UNITS
9: yem_se            |  (with switch set for MWA)       |    DEF: minimum wage
```

1.1.2 Running a system with default configuration

Say that we are interested in running the tax system for the year 2021 of Sweden. Building further on the previous example we can look at the tax-systems contained in the model for Sweden.

```
mod.countries["SE"].systems
```

```
0: SE_2006
1: SE_2007
2: SE_2008
3: SE_2009
4: SE_2010
5: SE_2011
6: SE_2012
7: SE_2013
8: SE_2014
9: SE_2015
10: SE_2016
11: SE_2017
12: SE_2018
13: SE_2019
14: SE_2020
15: SE_2021
16: SE_2022
17: SE_2023
```

In order to run the tax system we need a dataset that fits the requirement to use. The model however provides us with a list of datasets that are configured already.

```
mod.countries["SE"].systems["SE_2021"].datasets
```

```
0: training_data     |
1: SE_2019_a1        |
2: SE_2020_b1        |
3: SE_2021_hhot      |
4: SE_2021_b1        | best match
```

Here we see that there are multiple datasets configured, but that PL_2021_b1 is set as the best match dataset for this taxsystem. Provided that you have the microdata stored somewhere, you can then load it as a `pandas.DataFrame` and run the model in the following way:

```
import pandas as pd
data=pd.read_csv(r"C:\EUROMOD_RELEASES_I6.0+\Input\SE_2021_b1.txt",sep="\t")
out = mod.countries["SE"].systems["SE_2021"].run(data,"SE_2021_b1")
out
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
------------------------------
Simulation
------------------------------
         constantsToOverwrite: {}
         errors: []
         output_filenames: ['se_2021_std.txt']
         outputs: Pandas DataFrame of 240 variables and 21671 observations.
```

This returns a Simulation object with multiple attributes. The one of interest here is outputs, which contains the outputdataset(s) returned by the microsimulation model.

```
outputdata_baseline = out.outputs[0]
outputdata_baseline
```

```
            idhh     idperson     idmother  ...  tu_bho_se_IsDependentChild  \
0          200.0      20001.0          0.0  ...                         0.0
1          300.0      30001.0          0.0  ...                         0.0
2          300.0      30002.0          0.0  ...                         0.0
3          500.0      50001.0          0.0  ...                         0.0
4          500.0      50002.0          0.0  ...                         0.0
...          ...          ...          ...  ...                         ...
21666  1936500.0  193650002.0          0.0  ...                         0.0
21667  1936800.0  193680001.0          0.0  ...                         0.0
21668  1936800.0  193680002.0          0.0  ...                         0.0
21669  1936800.0  193680003.0  193680002.0  ...                         1.0
21670  1936800.0  193680004.0  193680002.0  ...                         1.0

       tu_bho_se_IsLoneParent  tu_bho_se_IsPartner
0                         0.0                 0.0
1                         0.0                 0.0
2                         0.0                 1.0
3                         0.0                 0.0
4                         0.0                 1.0
...                       ...                 ...
21666                     0.0                 1.0
21667                     0.0                 1.0
21668                     0.0                 0.0
21669                     0.0                 0.0
21670                     0.0                 0.0

[21671 rows x 240 columns]
```

1.1.3 Running a system while changing a constant

One of the advantages of using the Python Connectors is the ability to run many counterfactual scenario's for the EUROMOD model. One can for example change the Tax Free income limit in Poland. There are multiple ways to do this via the euromod package in Python, but one very straightforward way is to use the `constantsToOverwrite` option which is a a dictionary, having the targetted constant as a key and the value to overwrite with as a value.

```
out=mod['SE']['SE_2021'].run(data,"SE_2021_b1",constantsToOverwrite={("$tinna_rate2",""):
↪'0.4'})
outputdata_changed= out.outputs[0]
sum(outputdata_changed.ils_dispy - outputdata_baseline.ils_dispy)
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
-12247774.761818277
```

The optional parameter `constantsToOverwrite` specifies which constants to overwrite in the policy spline. `constantsToOverwrite` must be a `dict`, where the keys are `tuples` of two `str` objects: the first string is the name of the constant and the second string is its group number (**Note:** Pass an empty string if the group number is `None`); the values are `str` with the new values of the constants. The default is `None`.

1.1.4 Run with add-ons

Run the simulation for the Swedish system SE_2021 including the Marginal Tax-Rate add-on 'MTR'.

```
out =mod['SE']['SE_2021'].run(data,"SE_2021_b1",addons=[("MTR","MTR")])
out
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
------------------------------
Simulation
------------------------------
        constantsToOverwrite: {}
        errors: []
        output_filenames: ['se_2021_base_mtr.txt', 'se_2021_mtr.txt']
        outputs: Pandas DataFrame of 246 variables and 21671 observations., Pandas␣
↪DataFrame of 39 variables and 21671 observations.
```

As one can see there are two datasets returned by the model. Both of them can be accessed. The average marginal tax rate for example can then be straightforwardly computed as

```
out.outputs['se_2021_mtr.txt'].mtrpc.mean()
```

```
19.419071885518324
```

The optional parameter `addons` that we passed to the run command is a `list` of EUROMOD Addons to be integrated in the spine . Each item of the `list` is a `tuple` with two `str` objects. The first `str` is the name of the Addon and the second `str` is the name of the system in the Addon to be integrated (typically, it is the name of the Addon _ two-letter country code, e.g. LMA_AT). The default value here is [].

1.1.5 Run with extensions

Run the simulation for the Swedish system SE_2021 switching on the Benefit Take-up Adjustment extension 'BTA'.

```
out_BTA =mod['SE']['SE_2021'].run(data,"SE_2021_b1",switches=[("BTA",True)])
out
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
------------------------------
Simulation
------------------------------
        constantsToOverwrite: {}
        errors: []
        output_filenames: ['se_2021_base_mtr.txt', 'se_2021_mtr.txt']
        outputs: Pandas DataFrame of 246 variables and 21671 observations., Pandas␣
↪DataFrame of 39 variables and 21671 observations.
```

```
out_BTA.outputs[0].ils_ben.mean() - outputdata_baseline.ils_ben.mean()
```

```
0.0
```

The optional parameter `switches` must define a `list` of the EUROMOD extensions to be switched on or off in the simulation. Each item in the `list` is a `tuple` with two objects. The first object is a `str` short name of the Extension. The second object is a `boolean`. The default is [].

1.2 User Guide

The Euromod Conector is a Python library providing tools for running simulations and interacting with the tax-benefit microsimulation model EUROMOD.

1.2.1 Installation

The Euromod Connector can be installed from PyPi using *pip*:

```
$ pip install euromod
```

Requirements

In order to run the model, we require two components: 1) the model (coded policy rules), and 2) the input microdata with the variables that respect the EUROMOD naming conventions. For more information, please, read the sections "Model" and "Input microdata" on the Download Euromod web page.

1.2.2 Running and navigating the model

The euromod package is object oriented and evolves around using the Model class that loads a representation of the EUROMOD model. This can be imported as follows:

```
from euromod import Model
```

Create an object of the `Model` class by passing the path to the EUROMOD project:

```
mod=Model(r"C:\EUROMOD_RELEASES_I6.0+")
mod
```

```
------------------------------
Model
------------------------------
         countries: 28 elements
         extensions: 11 elements
         model_path: 'C:\\EUROMOD_RELEASES_I6.0+'
```

Note that every object that is related to the EUROMOD project comes with an informative description. Here we can see that the model has 3 relevant attributes to the user:

- countries
- extensions
- model_path

The countries and extensions attributes contain elements of the respective objects. If we take a look at `countries`:

```
mod.countries
```

```
0: AT
1: BE
2: BG
3: CY
4: CZ
5: DE
6: DK
7: EE
8: EL
9: ES
10: FI
11: FR
12: HR
13: HU
14: IE
15: IT
16: LT
17: LU
18: LV
19: MT
20: NL
21: PL
22: PT
```

(continues on next page)

(continued from previous page)

```
23: RO
24: SE
25: SI
26: SK
27: SL
```

The `countries` attirbute is a container storing the Country objects nesting the country-specific tax-benefit `policies` and `systems`. We see indeed that the euromod model contains 28 countries.

Note: The elements in an attribute of type Container can be accessed by indexing the attribute with the element position.

In a similar fashion we can look what kind of extensions are stored in the model.

Tip: The countries container can be indexed by both the number of the element and the country shortcode.

Let us take a look at Sweden:

```
mod.countries["SE"]
```

```
------------------------------
Country
------------------------------
         datasets: 27 elements
         extensions: 12 elements
         local_extensions: COVID
         name: 'SE'
         policies: 26 elements
         systems: 18 elements
```

Here we see again an informative representation of the `Country` object, which contains several attributes that can be accessed. In order to simulate a system we run a specific System object. We can obtain the systems for Sweden as follows:

```
mod.countries["SE"].systems
```

```
0: SE_2006
1: SE_2007
2: SE_2008
3: SE_2009
4: SE_2010
5: SE_2011
6: SE_2012
7: SE_2013
8: SE_2014
9: SE_2015
10: SE_2016
11: SE_2017
12: SE_2018
13: SE_2019
```

(continues on next page)

(continued from previous page)

```
14: SE_2020
15: SE_2021
16: SE_2022
17: SE_2023
```

Tip: The systems container can be indexed by both the number of the element and the system name.

Running a simulation

In order to run the tax system we need a dataset that fits the requirement to use.

See also:

See Other important euromod objects on what datasets are configured and how.

If you know already which dataset to use you can simply load the data and run the model as follows:

```
import pandas as pd
data=pd.read_csv(r"C:\EUROMOD_RELEASES_I6.0+\Input\SE_2021_b1.txt",sep="\t")
out_baseline = mod.countries["SE"].systems["SE_2021"].run(data,"SE_2021_b1")
out_baseline
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
------------------------------
Simulation
------------------------------
         constantsToOverwrite: {}
         errors: []
         output_filenames: ['se_2021_std.txt']
         outputs: Pandas DataFrame of 240 variables and 21671 observations.
```

Note that the run function here takes the mandatory argument `dataset_id`, which in our case is SE_2021_b1. This is necessary such that EUROMOD can apply the dataset specific logic with respect to setting default values and uprating. This returned us a Simulation object with multiple attributes. The one of interest here is outputs, which contains the outputdataset(s) returned by the microsimulation model:

```
out_baseline.outputs[0]
```

```
            idhh     idperson     idmother  ...  tu_bho_se_IsDependentChild  \
0          200.0      20001.0          0.0  ...                         0.0
1          300.0      30001.0          0.0  ...                         0.0
2          300.0      30002.0          0.0  ...                         0.0
3          500.0      50001.0          0.0  ...                         0.0
4          500.0      50002.0          0.0  ...                         0.0
...          ...          ...          ...  ...                         ...
21666  1936500.0  193650002.0          0.0  ...                         0.0
21667  1936800.0  193680001.0          0.0  ...                         0.0
21668  1936800.0  193680002.0          0.0  ...                         0.0
21669  1936800.0  193680003.0  193680002.0  ...                         1.0
```

(continues on next page)

(continued from previous page)

```
21670  1936800.0  193680004.0  193680002.0  ...                          1.0

       tu_bho_se_IsLoneParent  tu_bho_se_IsPartner
0                         0.0                  0.0
1                         0.0                  0.0
2                         0.0                  1.0
3                         0.0                  0.0
4                         0.0                  1.0
...                       ...                  ...
21666                     0.0                  1.0
21667                     0.0                  1.0
21668                     0.0                  0.0
21669                     0.0                  0.0
21670                     0.0                  0.0

[21671 rows x 240 columns]
```

Navigating the model

The `Model` object actually contains a full representation of the model that can be accessed using it's attributes. The implementation in Python mimicks the hierarchical structure of the EUROMOD User Interface. A full description of the available types can be found in the API reference.

The spine

The spine of EUROMOD is what represents the series of calculations with to respect taxes and benefits. The spine consists out of three hierarchically ordered elements:

- Policy
  - Function
    - Parameter

The connector mimmicks this hierarchical implementation through an object-oriented representation. The three hierarchical elements are defined on the Country level and implemented on the System level.

Let us take a look at the policies, which are an attribute of the `Country` object:

```
mod.countries["SE"].policies
```

```
0: setdefault_se         |                                    |    DEF: SET DEFAULT
1: uprate_se             |                                    |    DEF: UPRATING FACTORS
2: ConstDef_se           |                                    |    DEF: CONSTANTS
3: IlsDef_se             |                                    |    DEF: INCOME CONCEPTS␣
↪(standardized)
4: IlsUDBdef_se          |                                    |    DEF: INCOME CONCEPTS␣
↪(UDB)
5: ildef_se              |                                    |    DEF: INCOME CONCEPTS␣
↪(non-standardized)
6: random_se             |                                    |    DEF: Random assignment
7: TransLMA_se           |                                    |    DEF: Modelling labour␣
```

(continues on next page)

(continued from previous page)

```
↪market transitions (DO NOT S ...
8: tudef_se              |                                    |    DEF: ASSESSMENT UNITS
9: yem_se                |  (with switch set for MWA)         |    DEF: minimum wage
10: neg_se               |                                    |    DEF: recode negative␣
↪self-employment income to zer ...
11: yemcomp_se           |                                    |    BEN: wage compensation␣
↪scheme COVID-19  (ONLY WORK ...
12: bunct_se             |                                    |    BEN: unemployment␣
↪benefit (contributory)
13: bfapl_se             |  (with switch set for PBE)         |    BEN: Parental leave␣
↪benefit
14: bpa_se               |  (with switch set for PBE)         |    BEN: Special leave␣
↪days other parent (10 days)
15: tscee_se             |                                    |    SIC: Employee Social␣
↪Insurance contribution
16: tscer_se             |                                    |    SIC: Employer Social␣
↪Insurance contribution
17: tscse_se             |                                    |    SIC: Self-employed␣
↪Social Insurance contribution
18: tin_se               |                                    |    TAX: Personal Income␣
↪tax
19: tinkt_se             |                                    |    TAX: Tax on Capital␣
↪Income
20: bch_se               |                                    |    BEN: Child benefit
21: bho_se               |                                    |    BEN: Housing allowance
22: bhope_se             |                                    |    BEN: Housing allowance␣
↪for pensioners
23: bsamt_se             |                                    |    BEN: Social Assistance␣
↪(means-tested)
24: output_std_se        |                                    |    DEF: STANDARD OUTPUT␣
↪INDIVIDUAL LEVEL
25: output_std_hh_se     |                                    |    DEF: STANDARD OUTPUT␣
↪HOUSEHOLD LEVEL
```

As one can see the `policies` attribute is a Container object therefor it's elements, which are of the type Policy here, are accessible by indexing:

```
mod.countries["SE"].policies[12]
```

```
------------------------------
Policy
------------------------------
        ID: '01f1c7ff-3b6d-4191-bc71-bb86db5603d6'
        comment: 'BEN: unemployment benefit (contributory)'
        extensions: 0 elements
        functions: 16 elements
        name: 'bunct_se'
        order: '13'
        private: 'no'
        spineOrder: '13'
```

The implementation of a policy is accessible through the System object.

```
mod.countries["SE"].systems["SE_2021"].policies
```

```
0: setdefault_se          | on                                        |    DEF: SET DEFAULT
1: uprate_se              | on                                        |    DEF: UPRATING␣
↪FACTORS
2: ConstDef_se            | on                                        |    DEF: CONSTANTS
3: IlsDef_se              | on                                        |    DEF: INCOME␣
↪CONCEPTS (standardized)
4: IlsUDBdef_se           | on                                        |    DEF: INCOME␣
↪CONCEPTS (UDB)
5: ildef_se               | on                                        |    DEF: INCOME␣
↪CONCEPTS (non-standardized)
6: random_se              | on                                        |    DEF: Random␣
↪assignment
7: TransLMA_se            | off                                       |    DEF: Modelling␣
↪labour market transitions (DO NOT S ...
8: tudef_se               | on                                        |    DEF: ASSESSMENT␣
↪UNITS
9: yem_se                 | off (with switch set for MWA)             |    DEF: minimum wage
10: neg_se                | on                                        |    DEF: recode␣
↪negative self-employment income to zer ...
11: yemcomp_se            | on                                        |    BEN: wage␣
↪compensation scheme COVID-19  (ONLY WORK ...
12: bunct_se              | off                                       |    BEN: unemployment␣
↪benefit (contributory)
13: bfapl_se              | off (with switch set for PBE)             |    BEN: Parental leave␣
↪benefit
14: bpa_se                | off (with switch set for PBE)             |    BEN: Special leave␣
↪days other parent (10 days)
15: tscee_se              | on                                        |    SIC: Employee␣
↪Social Insurance contribution
16: tscer_se              | on                                        |    SIC: Employer␣
↪Social Insurance contribution
17: tscse_se              | on                                        |    SIC: Self-employed␣
↪Social Insurance contribution
18: tin_se                | on                                        |    TAX: Personal␣
↪Income tax
19: tinkt_se              | on                                        |    TAX: Tax on Capital␣
↪Income
20: bch_se                | on                                        |    BEN: Child benefit
21: bho_se                | on                                        |    BEN: Housing␣
↪allowance
22: bhope_se              | on                                        |    BEN: Housing␣
↪allowance for pensioners
23: bsamt_se              | on                                        |    BEN: Social␣
↪Assistance (means-tested)
24: output_std_se         | on                                        |    DEF: STANDARD␣
↪OUTPUT INDIVIDUAL LEVEL
25: output_std_hh_se      | off                                       |    DEF: STANDARD␣
↪OUTPUT HOUSEHOLD LEVEL
```

Here we see that some policies are turned **off** by default. Note that the behaviour of the policies can be controlled from the connector. We can for example switch the policy bunct_se to **on**. Let us first look at the policy:

```
mod.countries["SE"].systems["SE_2021"].policies[12]
```

```
------------------------------
PolicyInSystem
------------------------------
        ID: 'bde78132-3f44-4d2e-a1ea-4849f88c277601f1c7ff-3b6d-4191-bc71-bb86db5603d6'
        comment: 'BEN: unemployment benefit (contributory)'
        extensions: 0 elements
        functions: 16 elements
        name: 'bunct_se'
        order: '13'
        polID: '01f1c7ff-3b6d-4191-bc71-bb86db5603d6'
        private: 'no'
        spineOrder: '13'
        switch: 'off'
        sysID: 'bde78132-3f44-4d2e-a1ea-4849f88c2776'
```

We see here the attribute switch that is part of the PolicyInSystem class. This attribute, and similarly the other attributes of the object, is modifiable and the changes that you will make will be passed to the EUROMOD software when simulating.

Important: Currently, the python connector does not support the save option. Therefore, changes implemented during a Python session cannot be saved.

To apply permanent changes to the model, we recommend using the User Interface of EUROMOD.

Attention: Note that the python connector is not checking what kind of modifications you make to the model. Changing values of attributes like ID's are definitely not recommended.

```
mod.countries["SE"].systems["SE_2021"].policies[13].switch = 'on'
out_with_bunct_se = mod.countries["SE"].systems["SE_2021"].run(data,"SE_2021_b1")
# Here we see that the average benefit, which is represented by ils_ben,  is indeed␣
↪larger by switching on the bunct_se policy.
out_baseline.outputs[0].ils_ben.mean() - out_with_bunct_se.outputs[0].ils_ben.mean()
```

```
Simulation for system SE_2021 with dataset SE_2021_b1 finished.
```

```
0.0
```

```
mod.countries["SE"].systems["SE_2021"].policies
```

```
0: setdefault_se         | on                                          |    DEF: SET DEFAULT
1: uprate_se             | on                                          |    DEF: UPRATING␣
↪FACTORS
2: ConstDef_se           | on                                          |    DEF: CONSTANTS
3: IlsDef_se             | on                                          |    DEF: INCOME␣
↪CONCEPTS (standardized)
4: IlsUDBdef_se          | on                                          |    DEF: INCOME␣
↪CONCEPTS (UDB)
```

(continues on next page)

(continued from previous page)

```
5: ildef_se              | on                                    |    DEF: INCOME␣
↪CONCEPTS (non-standardized)
6: random_se             | on                                    |    DEF: Random␣
↪assignment
7: TransLMA_se           | off                                   |    DEF: Modelling␣
↪labour market transitions (DO NOT S ...
8: tudef_se              | on                                    |    DEF: ASSESSMENT␣
↪UNITS
9: yem_se                | off (with switch set for MWA)         |    DEF: minimum wage
10: neg_se               | on                                    |    DEF: recode␣
↪negative self-employment income to zer ...
11: yemcomp_se           | on                                    |    BEN: wage␣
↪compensation scheme COVID-19  (ONLY WORK ...
12: bunct_se             | off                                   |    BEN: unemployment␣
↪benefit (contributory)
13: bfapl_se             | on (with switch set for PBE)          |    BEN: Parental leave␣
↪benefit
14: bpa_se               | off (with switch set for PBE)         |    BEN: Special leave␣
↪days other parent (10 days)
15: tscee_se             | on                                    |    SIC: Employee␣
↪Social Insurance contribution
16: tscer_se             | on                                    |    SIC: Employer␣
↪Social Insurance contribution
17: tscse_se             | on                                    |    SIC: Self-employed␣
↪Social Insurance contribution
18: tin_se               | on                                    |    TAX: Personal␣
↪Income tax
19: tinkt_se             | on                                    |    TAX: Tax on Capital␣
↪Income
20: bch_se               | on                                    |    BEN: Child benefit
21: bho_se               | on                                    |    BEN: Housing␣
↪allowance
22: bhope_se             | on                                    |    BEN: Housing␣
↪allowance for pensioners
23: bsamt_se             | on                                    |    BEN: Social␣
↪Assistance (means-tested)
24: output_std_se        | on                                    |    DEF: STANDARD␣
↪OUTPUT INDIVIDUAL LEVEL
25: output_std_hh_se     | off                                   |    DEF: STANDARD␣
↪OUTPUT HOUSEHOLD LEVEL
```

As mentioned earlier, the connector mimicks the hierarchical structure of the UI. Hence, the definition of functions and parameters are defined on the country level, and their actual implementation are here also accessible via the Tax System. Note that also here, the values of a Parameter and the switch of a Function can be manipulated through the Python Connector without saving the changes permanently:

```
print("Overview of the functions defined in the bho_se policy:")
print(mod.countries["SE"].policies[22].functions)
print("System specific Implementation of functions:")
print(mod.countries["SE"].systems["SE_2021"].policies[22].functions)
```

```
Overview of the functions defined in the bho_se policy:
```

(continues on next page)

(continued from previous page)

```
0: DefVar       |    Temporary variables for Housing Allowance for pens ...
1: Elig         |    Living with partner
2: ArithOp      |    Wealth to be included in the means
3: Elig         |    Living without partner
4: ArithOp      |    Wealth to be included in the means
5: Allocate     |    Allocation of wealth to the partners
6: Elig         |    Elderly or disabled adult (i.e. head or partner)
7: BenCalc      |    "Reserved amount (""income disregard"")"
8: BenCalc      |    Change in definition of Income Means for for Housi ...
9: DefIl        |    Income Means for Housing Allowance for pensioners
10: BenCalc     |    Income of children is not take into account in the ...
11: BenCalc     |    Maximum housing allowance
12: BenCalc     |
13: BenCalc     |    Maximum housing allowance
14: BenCalc     |    Maximum housing allowance
15: Allocate    |    Allocation of income to the partners
16: BenCalc     |    Final housing allowance for pensioners
17: Allocate    |    Sharing housing cost
18: BenCalc     |    Housing allowance for pensioners

System specific Implementation of functions:
0: DefVar       | on        |    Temporary variables for Housing Allowance for pens ...
1: Elig         | on        |    Living with partner
2: ArithOp      | on        |    Wealth to be included in the means
3: Elig         | on        |    Living without partner
4: ArithOp      | on        |    Wealth to be included in the means
5: Allocate     | on        |    Allocation of wealth to the partners
6: Elig         | on        |    Elderly or disabled adult (i.e. head or partner)
7: BenCalc      | on        |    "Reserved amount (""income disregard"")"
8: BenCalc      | on        |    Change in definition of Income Means for for Housi ...
9: DefIl        | on        |    Income Means for Housing Allowance for pensioners
10: BenCalc     | on        |    Income of children is not take into account in the ...
11: BenCalc     | on        |    Maximum housing allowance
12: BenCalc     | off       |
13: BenCalc     | off       |    Maximum housing allowance
14: BenCalc     | on        |    Maximum housing allowance
15: Allocate    | on        |    Allocation of income to the partners
16: BenCalc     | on        |    Final housing allowance for pensioners
17: Allocate    | on        |    Sharing housing cost
18: BenCalc     | on        |    Housing allowance for pensioners
```

```
print("Overview of the parameters defined in the bho_se policy:")
print(mod.countries["SE"].policies[22].functions[0].parameters)
print("Implementation of parameters:")
print(mod.countries["SE"].systems["SE_2021"].policies[22].functions[0].parameters)
```

```
Overview of the parameters defined in the bho_se policy:
0: i_means_bhope_prel

Implementation of parameters:
0: i_means_bhope_prel     | 0
```

Other Important EUROMOD Objects

Central to the EUROMOD project, next to the coding of the policies is the microdata. How datasets should be treated by the model is configured in the model already. The attributes of the datasets are just like the spine-elements accessible and modifiable.

```
mod.countries["SE"].datasets
```

```
0: SE_2007_a4
1: SE_2008_a3
2: training_data
3: SE_2010_a1
4: SE_2012_a2
5: SE_2015_a1
6: SE_2009_hhot
7: SE_2010_hhot
8: SE_2011_hhot
9: SE_2012_hhot
10: SE_2013_hhot
11: SE_2014_hhot
12: SE_2015_hhot
13: SE_2016_hhot
14: SE_2017_hhot
15: SE_2016_a1
16: SE_2018_hhot
17: SE_2019_hhot
18: SE_2018_a2
19: SE_2017_a3
20: SE_2020_hhot
21: SE_2019_a1
22: SE_2020_b1
23: SE_2021_hhot
24: SE_2022_hhot
25: SE_2021_b1
26: SE_2023_hhot
```

In the previous section we used SE_2021_b1. Let us have a look at it.

```
mod.countries["SE"].datasets["SE_2021_b1"]
```

```
------------------------------
Dataset
------------------------------
         ID: 'c7b651ed-b311-4e39-80b4-18ca19957ce7'
         coicopVersion: ''
         comment: ''
         currency: 'national'
         decimalSign: '.'
         name: 'SE_2021_b1'
         private: 'no'
         readXVariables: 'no'
         useCommonDefault: 'no'
         yearCollection: '2021'
```

(continues on next page)

(continued from previous page)

```
        yearInc: '2020'
```

Similarly to the attributes in the `Policy`, `Function` and `Parameter` objects, the attributes of the Dataset can be modified here.

We can further check what datasets are implemented for a given system, for example SE_2021, as follows:

```
mod.countries["SE"]["SE_2021"].datasets
```

```
0: training_data     |
1: SE_2019_a1        |
2: SE_2020_b1        |
3: SE_2021_hhot      |
4: SE_2021_b1        | best match
```

Another important concept in euromod are extensions that are defined globally on the `Model` level:

```
mod.extensions
```

```
0: Benefit Take-up Adjustments
1: Tax Compliance Adjustments
2: Full Year Adjustments
3: Uprating by Average Adjustment
4: Extended Policy Simulation
5: Parental leave benefits
6: Minimum Wage Adjustments
7: HHoT unemployment extension
8: EUROMOD JRC-Interface
9: HHoT - Extended Simulation
10: HHoT - Non Compulsory Payments
```

Or locally on the `Country` level:

```
mod.countries["SE"].extensions
```

```
0: COVID benefit
1: Benefit Take-up Adjustments
2: Tax Compliance Adjustments
3: Full Year Adjustments
4: Uprating by Average Adjustment
5: Extended Policy Simulation
6: Parental leave benefits
7: Minimum Wage Adjustments
8: HHoT unemployment extension
9: EUROMOD JRC-Interface
10: HHoT - Extended Simulation
11: HHoT - Non Compulsory Payments
```

The `extensions` attribute is of type Container. If we want to access the information stored in the Minimum Wage Adjustments extension for example, we can simply use the following command:

```
mod.countries["SE"].extensions[7]
```

```
------------------------------
Extension
------------------------------
        ID: '557c232a-9ce6-4808-b52f-ca5e02fe8cf4'
        look: '|BUTTON_COLOR=-16744384|'
        name: 'Minimum Wage Adjustments'
        shortName: 'MWA'
```

1.3 API Reference

This reference guide lists the main public objects of the Euromod Connector package. The *euromod.core* module contains most of the public classes of the library. It provides useful functionalities that allow the user to interact with EUROMOD[1] and run simulations. The *euromod.container* module defines a storage class for the model objects accessible by indexing.

Please, refer to the User Guide and Examples for futher readings.

1.3.1 euromod

Below are listed the main public classes of the euromod module.

euromod.container

Below are listed the main public classes of the euromod.container module.

Table 1: **Classes**

Container	This class is a container for objects that allow for indexing and representation in multiple ways:

class euromod.container.**Container**(*idDict=False*)

This class is a container for objects that allow for indexing and representation in multiple ways:
- via keys that are the name of the objects or,
- via integer indexing as in a list.

Overview

Table 2: Methods

find(key, pattern, return_children, case_insentive)	Find objects that match pattern.
items()	Get items of the *Container*.
keys()	Get keys of the *Container*.
values()	Get values of the *Container*.

[1] See the documetation for the EUROMOD tax-benefit microsimulation model on the official webpage and in the resources page.

Methods

find(*key*, *pattern*, *return_children=False*, *case_insentive=True*)

Find objects that match pattern.

Parameters

- **key** (str) – Name of the attribute or the attribute of a child element that you want to look for One can search child elements by using the dot-notation. E.g.: mod["BE"]["BE_2023"].policies.find("functions.name","BenCalc")
- **pattern** (str) – pattern that you want to match.
- **return_children** (*bool, optional*) – When True, the return type will be a Container containing elements of the type for which the find method was used When False, the return type will be a Container of the elements of the deepest level specified by the pattern keyword. E.g.: mod["BE"]["BE_2023"].policies.find("function) The default is False.
- **case_insentive** (bool, optional) – DESCRIPTION. The default is True.

Returns

An object that matches the pattern.

Return type

Container

items()

Get items of the *Container*.

Returns

Object items.

Return type

Container.items

keys()

Get keys of the *Container*.

Returns

Names of the attribute or the attribute of a child element.

Return type

Container.keys

values()

Get values of the *Container*.

Returns

Value of the object attribute.

Return type

Container.values

euromod.core

Below are listed the main public classes of the euromod.core module.

Table 3: **Classes**

Country	Country-specific EUROMOD tax-benefit model.
Dataset	Dataset available in a country model.
DatasetInSystem	Datasets available in a system model.
Extension	EUROMOD extensions.
ExtensionSwitch	A class containing the extension switches of an object.
Function	Functions implemented in a country policy.
FunctionInSystem	Functions implemented in a policy for a specific system.
Model	Base class of the Euromod Connector instantiating the microsimulation model
Parameter	Parameters set up in a function.
ParameterInSystem	Parameters set up in a function for a specific system.
Policy	Policy rules modeled in a country.
PolicyInSystem	Policy rules modeled in a system.
ReferencePolicy	Object storing the reference policies.
Simulation	Object storing the simulation results.
System	A EUROMOD tax-benefit system.

class euromod.core.**Country**(*country: str*, *model:* Model)

Country-specific EUROMOD tax-benefit model.

This class instantiates the EUROMOD tax benefit model for a given country. A class instance is automatically generated and stored in the attribute `countries` of the base class *Model*.

This class contains subclasses of type *System*, *Policy*, *Dataset* and *Extension*.

Parameters

- **country** (*str*) – Name of the country. Must be a two-letter country codes, see the Eurostat Glossary:Country codes.
- **model** (*Model*) – A class containing the EUROMOD base model.

Returns

A class containing the EUROMOD country models.

Return type

Country

Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> mod.countries[0]
```

Overview

Table 4: Attributes

datasets	A Container with *Dataset* objects.
extensions	A Container with *Extension* objects. These are the local + model extensions defined.
local_extensions	A Container with *Extension* objects. These are the local extensions defined for the country.
model	"*Model* Returns the base *Model* object.
name	Two-letters country code.
policies	A Container with *Policy* objects.
systems	A Container with *System* objects.

Table 5: Methods

get_switch_value(ext_name, dataset_name, sys_name)	Get the configuration of the switch.
load_data(ID_DATASET, PATH_DATA)	Load data as a pandas.DataFrame object.

Attributes

datasets: *container.Container*[*Dataset*] | None = None

A Container with *Dataset* objects.

extensions: *container.Container*[*Extension*] | None = None

A Container with *Extension* objects. These are the local + model extensions defined.

local_extensions: *container.Container*[*Extension*] | None = None

A Container with *Extension* objects. These are the local extensions defined for the country.

model: *Model*

"*Model* Returns the base *Model* object.

name: str

Two-letters country code.

policies: *container.Container*[*Policy*] | None = None

A Container with *Policy* objects.

systems: *container.Container*[*System*] | None = None

A Container with *System* objects.

Methods

get_switch_value(*ext_name: str | None = None*, *dataset_name: str | None = None*, *sys_name: str | None = None*)

Get the configuration of the switch.

Parameters

- **ext_name** (str , optional) – Name of the extension. The default is None.
- **dataset_name** (str , optional) – Name of the dataset. The default is None.

- **sys_name** (str, optional) – Name of the system. The default is None.

Raises
KeyError – Is raised if ext_name, dataset_name or sys_name, but is not configured in the model.

Returns
Object containing information how the switch is configured. Note that there is only a value returned if the switch is either explicitly 'off' or 'on'. When it's configured as n/a in the model no value will be included.

Return type
Container[*ExtensionSwitch*]

load_data(*ID_DATASET*, *PATH_DATA=None*)

Load data as a pandas.DataFrame object.

Parameters

- **ID_DATASET** (str) – Name of the dataset excluding extension (Note: must be a *txt* file).
- **PATH_DATA** (str, optional) – Path to the dataset. Default is the folder "PATH_TO_EUROMOD_PROJECT/Input".

Returns
Dataset is returned as a pandas.DataFrame object.

Return type
pandas.DataFrame

class euromod.core.**Dataset**(**args*)

Dataset available in a country model.

This class contains the relevant information about a dataset.

Returns
A class with the country-specific dataset.

Return type
Dataset

Overview

Table 6: Attributes

ID	Dataset identifier number.
coicopVersion	COICOP version.
comment	Comment about the dataset.
currency	Currency of the monetary values in the dataset.
decimalSign	Decimal sign
name	Name of the dataset.
parent	The country-specific class.
private	Access type.
readXVariables	Read variables.
useCommonDefault	Use default.
yearCollection	Year of the dataset collection.
yearInc	Reference year for the income variables.

Attributes

ID: str
Dataset identifier number.

coicopVersion: str = ''
COICOP version.

comment: str = ''
Comment about the dataset.

currency: str = ''
Currency of the monetary values in the dataset.

decimalSign: str = ''
Decimal sign

name: str
Name of the dataset.

parent: *Country*
The country-specific class.

private: str = 'no'
Access type.

readXVariables: str = 'no'
Read variables.

useCommonDefault: str = 'no'
Use default.

yearCollection: str
Year of the dataset collection.

yearInc: str
Reference year for the income variables.

class euromod.core.**DatasetInSystem**
Datasets available in a system model.

Returns
A class with the system-specific dataset.

Return type
DatasetInSystem

Overview

Table 7: Attributes

ID	Dataset identifier number.
bestMatch	If yes, the current dataset is a best match for the specific system.
coicopVersion	COICOP version.
comment	Comment about the dataset.
currency	Currency of the monetary values in the dataset.
dataID	Identifier number of the reference dataset at the country level.
decimalSign	Decimal sign
name	Name of the dataset.
parent	The country specific class.
private	Access type.
readXVariables	Read variables.
sysID	Identifier number of the reference system.
useCommonDefault	Use default.
yearCollection	Year of the dataset collection.
yearInc	Reference year for the income variables.

Attributes

ID: str

Dataset identifier number.

bestMatch: str

If yes, the current dataset is a best match for the specific system.

coicopVersion: str

COICOP version.

comment: str

Comment about the dataset.

currency: str

Currency of the monetary values in the dataset.

dataID: str

Identifier number of the reference dataset at the country level.

decimalSign: str

Decimal sign

name: str

Name of the dataset.

parent: *Country*

The country specific class.

private: str

Access type.

readXVariables: str

Read variables.

sysID: str
Identifier number of the reference system.

useCommonDefault: str
Use default.

yearCollection: str
Year of the dataset collection.

yearInc: str
Reference year for the income variables.

class euromod.core.**Extension**(**arg*)
EUROMOD extensions.

Returns
A class with the model extensions.

Return type
Extension

Overview

Table 8: Attributes

name	Long name of the extension.
parent	The model base class.
shortName	Short name of the extension.

Attributes

name: str = None
Long name of the extension.

parent: *Model*
The model base class.

shortName: str = None
Short name of the extension.

class euromod.core.**ExtensionSwitch**(*info*, *ctry*)
A class containing the extension switches of an object.

This class is returned by *get_switch_value()* method and should not be used by the user as a stand alone.

Returns
A class with relevant information on the extension switch.

Return type
ExtensionSwitch

Overview

Table 9: Attributes

data_name	Name of the applicable dataset.
extension_name	Short name of the extension.
parent	The country-specific class.
sys_name	Name of the applicable system.
value	Value of the switch as configured in EUROOMOD.

Attributes

data_name: *str*

Name of the applicable dataset.

extension_name: *str*

Short name of the extension.

parent: *Country*

The country-specific class.

sys_name: *str*

Name of the applicable system.

value: *str* = ''

Value of the switch as configured in EUROOMOD.

class euromod.core.**Function**(**arg*)

Functions implemented in a country policy.

Returns

A class with country-specific function.

Return type

Function

Overview

Table 10: Attributes

ID	Identifier number of the function.
comment	Comment specific to the function.
extensions	A `Container` of *Extension* objects in a country.
name	Name of the function.
order	Order of the function in the specific spine.
parameters	A `Container` of *Parameter* objects in a country.
parent	The class of the country-specific policy.
polID	Identifier number of the reference policy.
private	Access type.
spineOrder	Order of the function in the spine.

Attributes

ID: str
Identifier number of the function.

comment: str
Comment specific to the function.

extensions: *container.Container*[*Extension*] | None = None
A Container of *Extension* objects in a country.

name: str
Name of the function.

order: str
Order of the function in the specific spine.

parameters: *container.Container*[*Parameter*] | None = None
A Container of *Parameter* objects in a country.

parent: *Policy*
The class of the country-specific policy.

polID: str
Identifier number of the reference policy.

private: str
Access type.

spineOrder: str
Order of the function in the spine.

class euromod.core.**FunctionInSystem**(**arg*)
Functions implemented in a policy for a specific system.

Returns
A class with the system-specific function.

Return type
FunctionInSystem

Overview

Table 11: Attributes

ID	Identifier number of the function.
comment	Comment specific to the function.
extensions	A `Container` of *Extension* objects in a system.
funID	Identifier number of the reference function at country level.
name	Name of the function.
order	Order of the function in the specific spine.
parameters	A `Container` with *ParameterInSystem* objects specific to a function.
parent	The class of the country-specific policy.
polID	Identifier number of the reference policy.
private	Access type.
spineOrder	Order of the function in the spine.
switch	Policy switch action.
sysID	Identifier number of the reference policy.

Attributes

ID: str
Identifier number of the function.

comment: str
Comment specific to the function.

extensions: *container.Container*[*Extension*]
A `Container` of *Extension* objects in a system.

funID: str
Identifier number of the reference function at country level.

name: str
Name of the function.

order: str
Order of the function in the specific spine.

parameters: *container.Container*[*ParameterInSystem*] | None = None
A `Container` with *ParameterInSystem* objects specific to a function.

parent: *Policy*
The class of the country-specific policy.

polID: str
Identifier number of the reference policy.

private: str
Access type.

spineOrder: str
Order of the function in the spine.

switch: str
Policy switch action.

sysID: str

Identifier number of the reference policy.

class euromod.core.**Model**(*model_path: str*)

Base class of the Euromod Connector instantiating the microsimulation model EUROMOD.

Parameters
model_path (str) – Path to the EUROMOD project.

Returns
A class containing the EUROMOD base model.

Return type
Model

Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
```

Overview

Table 12: Attributes

countries	A Container with *Country* objects.
extensions	A Container with *Model* extensions.
model_path	Path to the EUROMOD project.

Attributes

countries: *container.Container*[*Country*]

A Container with *Country* objects.

extensions: *container.Container*[*Extension*]

A Container with *Model* extensions.

model_path: str

Path to the EUROMOD project.

class euromod.core.**Parameter**(**arg*)

Parameters set up in a function.

Returns
A class with country-specific parameter.

Return type
Parameter

Overview

Table 13: Attributes

ID	Identifier number of the parameter.
comment	Comment specific to the parameter.
extensions	A Container with *Extension* objects.
funID	Identifier number of the reference function at country level.
group	Parameter group value.
name	Name of the parameter.
order	Order of the parameter in the specific spine.
parent	The class of the country-specific function.
spineOrder	Order of the parameter in the spine.

Attributes

ID: str

Identifier number of the parameter.

comment: str

Comment specific to the parameter.

extensions: *container.Container*[*Extension*] | None = None

A Container with *Extension* objects.

funID: str

Identifier number of the reference function at country level.

group: str = ''

Parameter group value.

Type
str

name: str

Name of the parameter.

order: str

Order of the parameter in the specific spine.

parent: *Function*

The class of the country-specific function.

spineOrder: str

Order of the parameter in the spine.

class euromod.core.**ParameterInSystem**

Parameters set up in a function for a specific system.

Returns
A class with the system-specific function parameter.

Return type
ParameterInSystem

Overview

Table 14: Attributes

ID	Identifier number of the parameter.
comment	Comment specific to the parameter.
extensions	A `Container` with *Extension* objects.
funID	Identifier number of the reference function at country level.
group	Parameter group number.
name	Name of the parameter.
order	Order of the parameter in the specific spine.
parID	Identifier number of the reference parameter at country level.
parent	The class of the country-specific function.
spineOrder	Order of the parameter in the spine.
sysID	Identifier number of the reference system.
value	Value of the parameter.

Attributes

ID: str

Identifier number of the parameter.

comment: str

Comment specific to the parameter.

extensions: *container.Container*

A `Container` with *Extension* objects.

funID: str

Identifier number of the reference function at country level.

group: str

Parameter group number.

name: str

Name of the parameter.

order: str

Order of the parameter in the specific spine.

parID: str

Identifier number of the reference parameter at country level.

parent: *Function*

The class of the country-specific function.

spineOrder: str

Order of the parameter in the spine.

sysID: str

Identifier number of the reference system.

value: str

Value of the parameter.

class euromod.core.**Policy**(**arg*)

Policy rules modeled in a country.

Returns
A class with the country-specific policies.

Return type
Policy

Overview

Table 15: Attributes

ID	Identifier number of the policy.
comment	Comment specific to the policy.
extensions	A Container of policy-specific *Extension* objects.
functions	A Container of policy-specific *Function* objects.
name	Name of the policy.
order	Order of the policy in the specific spine.
parent	The country-specific class.
private	Access type. Default is 'no'.
spineOrder	Order of the policy in the spine.

Attributes

ID: str

Identifier number of the policy.

comment: str

Comment specific to the policy.

extensions: *container.Container*[*Extension*] | None = None

A Container of policy-specific *Extension* objects.

functions: *container.Container*[*Function*] | None = None

A Container of policy-specific *Function* objects.

name: str

Name of the policy.

order: str

Order of the policy in the specific spine.

parent: *Country*

The country-specific class.

private: str = 'no'

Access type. Default is 'no'.

spineOrder: str

Order of the policy in the spine.

class euromod.core.**PolicyInSystem**(**arg*)

Policy rules modeled in a system.

Returns
A class with system-specific policies.

Return type
PolicyInSystem

Overview

Table 16: Attributes

ID	Identifier number of the policy.
comment	Comment specific to the policy.
extensions	A Container of policy-specific *Extension* objects.
functions	A Container with *FunctionInSystem* objects specific to the system
name	Name of the policy.
order	Order of the policy in the specific spine.
parent	The country-specific class.
polID	Identifier number of the reference policy at country level.
private	Access type. Default is 'no'.
spineOrder	Order of the policy in the spine.
switch	Policy switch action.
sysID	Identifier number of the reference system.

Attributes

ID: str
Identifier number of the policy.

comment: str
Comment specific to the policy.

extensions: *container.Container*[*Extension*]
A Container of policy-specific *Extension* objects.

functions: *container.Container*[*FunctionInSystem*] | None = None
A Container with *FunctionInSystem* objects specific to the system

name: str
Name of the policy.

order: str
Order of the policy in the specific spine.

parent: *Country*
The country-specific class.

polID: str
Identifier number of the reference policy at country level.

private: str
Access type. Default is 'no'.

spineOrder: str

Order of the policy in the spine.

switch: str

Policy switch action.

sysID: str

Identifier number of the reference system.

class euromod.core.**ReferencePolicy**(*info*, *parent*)

Object storing the reference policies.

Returns
A class with the country-specific reference policies.

Return type
ReferencePolicy

Overview

Table 17: Attributes

extensions	A Container of reference policy-specific *Extension* objects.
name	Name of the reference policy.
parent	The country-specific class.

Attributes

extensions: *container.Container*[*Extension*] | None = None

A Container of reference policy-specific *Extension* objects.

name: str

Name of the reference policy.

parent: *Country*

The country-specific class.

class euromod.core.**Simulation**(*out*, *constantsToOverwrite*)

Object storing the simulation results.

This is a class containing results from the simulation *run()* and other related configuration information.

Returns
A class with simulation output.

Return type
Simulation

Overview

Table 18: Attributes

constantsToOverwrite	A dict-type object with user-defined constants.
errors	A list with errors and warnings from the simulation run.
output_filenames	A list of file-names of simulation output.
outputs	A Container with pandas.DataFrame-type simulation results.

Attributes

constantsToOverwrite: dict[tuple(str, str), str]

A dict-type object with user-defined constants.

errors: list[str]

A list with errors and warnings from the simulation run.

output_filenames: list[str] | [] = []

A list of file-names of simulation output.

outputs: *container.Container*[pandas.DataFrame]

A Container with pandas.DataFrame-type simulation results. For indexing use an integer or a label from *output_filenames*.

class euromod.core.**System**(**arg*)

A EUROMOD tax-benefit system.

This class represents a EUROMOD tax system. Instances of this class are generated when loading the EUROMOD base model. These are collected in a Container as attribute *systems* of the *Country*.

Returns

A class with country systems.

Return type

System

Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> mod.countries[0].systems[-1]
```

Overview

Table 19: Attributes

ID	Identifier number of the system.
bestmatch_datasets	A Container with best-match *Dataset* objects in the system.
comment	Comment specific to the system.
currencyOutput	Currency of the simulation results.
currencyParam	Currency of the monetary parameters in the system.
datasets	A Container of *DatasetInSystem* objects in the system.
headDefInc	Main income definition.
name	Name of the system.
order	System order in the spine.
parent	The country-specific class.
policies	A Container of *PolicyInSystem* objects in the system.
private	Access type.
year	System year.

Table 20: Methods

run(data, dataset_id, constantsToOverwrite, verbose, outputpath, addons, switches, nowarnings, euro, public_components_only)	Run the simulation of a EUROMOD tax-benefit system.

Attributes

ID: str
Identifier number of the system.

bestmatch_datasets: *container.Container*[*Dataset*] | None = None
A Container with best-match *Dataset* objects in the system.

comment: str
Comment specific to the system.

currencyOutput: str
Currency of the simulation results.

currencyParam: str
Currency of the monetary parameters in the system.

datasets: *container.Container*[*DatasetInSystem*] | None = None
A Container of *DatasetInSystem* objects in the system.

headDefInc: str
Main income definition.

name: str
Name of the system.

order: str
System order in the spine.

parent: *Country*
The country-specific class.

policies: *container.Container*[*PolicyInSystem*] | None = None

A Container of *PolicyInSystem* objects in the system.

private: str

Access type.

year: str

System year.

Methods

run(*data: pandas.DataFrame*, *dataset_id: str*, *constantsToOverwrite: Dict[Tuple[str, str], str] | None = None*, *verbose: bool = True*, *outputpath: str = ''*, *addons: List[Tuple[str, str]] = []*, *switches: List[Tuple[str, bool]] = []*, *nowarnings=False*, *euro=False*, *public_components_only=False*)

Run the simulation of a EUROMOD tax-benefit system.

Parameters

- **data** (pandas.DataFrame) – input dataframe passed to the EUROMOD model.
- **dataset_id** (str) – ID of the dataset.
- **constantsToOverwrite** (dict [tuple [str, str], str], optional) – A dict with constants to overwrite. Note that the key is a tuple of two strings, for which the first element is the name of the constant and the second is the groupnumber. Note that the values must be defined as strings. Default is None.
- **verbose** (bool, optional) – If True then information on the output will be printed. Default is True.
- **outputpath** (str, optional) – When the output path is provided, there will be anoutput file generated. Default is "".
- **addons** (list [tuple [str, str]], optional) – List of tuples with addons to be integrated in the spine. The first element of the tuple is the name of the addon and the second element is the name of the system in the Addon to be integrated. Default is [].
- **switches** (list [tuple [str, bool]], optional) – List of tuples with extensions to be switched on or of. The first element of the tuple is the short name of the extension. The second element is a boolean Default is [].
- **nowarnings** (bool, optional) – If True, the warning messages resulting from the simulations will be suppressed. Default is False.
- **euro** (bool, optional) – If True, the monetary variables will be converted to euro for the simulation. Default value is False.
- **public_compoments_only** (bool, optional) – If True, the the model will be on with only the public compoments. Default value is False.

Raises

Exception – Exception when simulation does not finish succesfully, i.e. without errors.

Returns

A class containing simulation output and error messages.

Return type

Simulation

Example

```
>>> # Load the dataset
>>> import pandas as pd
>>> data = pd.read_csv("C:\EUROMOD_RELEASES_I6.0+\Input\sl_demo_v4.txt",sep="
↪")
>>> # Load EUROMOD
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> # Run simulation
>>> out=mod.countries['SL'].systems['SL_1996'].run(data,'sl_demo_v4')
```

1.4 License

```
                EUROPEAN UNION PUBLIC LICENCE v. 1.2
                 EUPL © the European Union 2007, 2016
```

This European Union Public Licence (the 'EUPL') applies to the Work (as defined below) which is provided under the terms of this Licence. Any use of the Work, other than as authorised under this Licence is prohibited (to the extent such use is covered by a right of the copyright holder of the Work).

The Work is provided under the terms of this Licence when the Licensor (as defined below) has placed the following notice immediately following the copyright notice for the Work:

```
   Licensed under the EUPL
```

or has expressed by any other means his willingness to license under the EUPL.

1. Definitions

In this Licence, the following terms have the following meaning:

- 'The Licence': this Licence.
- 'The Original Work': the work or software distributed or communicated by the Licensor under this Licence, available as Source Code and also as Executable Code as the case may be.
- 'Derivative Works': the works or software that could be created by the Licensee, based upon the Original Work or modifications thereof. This Licence does not define the extent of modification or dependence on the Original Work required in order to classify a work as a Derivative Work; this extent is determined by copyright law applicable in the country mentioned in Article 15.
- 'The Work': the Original Work or its Derivative Works.
- 'The Source Code': the human-readable form of the Work which is the most convenient for people to study and modify.
- 'The Executable Code': any code which has generally been compiled and which is meant to be interpreted by a computer as a program.
- 'The Licensor': the natural or legal person that distributes or communicates the Work under the Licence.
- 'Contributor(s)': any natural or legal person who modifies the Work under the Licence, or otherwise contributes to the creation of a Derivative Work.
- 'The Licensee' or 'You': any natural or legal person who makes any usage of the Work under the terms of the Licence.

- 'Distribution' or 'Communication': any act of selling, giving, lending, renting, distributing, communicating, transmitting, or otherwise making available, online or offline, copies of the Work or providing access to its essential functionalities at the disposal of any other natural or legal person.

2. Scope of the rights granted by the Licence

The Licensor hereby grants You a worldwide, royalty-free, non-exclusive, sublicensable licence to do the following, for the duration of copyright vested in the Original Work:

- use the Work in any circumstance and for all usage,
- reproduce the Work,
- modify the Work, and make Derivative Works based upon the Work,
- communicate to the public, including the right to make available or display the Work or copies thereof to the public and perform publicly, as the case may be, the Work,
- distribute the Work or copies thereof,
- lend and rent the Work or copies thereof,
- sublicense rights in the Work or copies thereof.

Those rights can be exercised on any media, supports and formats, whether now known or later invented, as far as the applicable law permits so.

In the countries where moral rights apply, the Licensor waives his right to exercise his moral right to the extent allowed by law in order to make effective the licence of the economic rights here above listed.

The Licensor grants to the Licensee royalty-free, non-exclusive usage rights to any patents held by the Licensor, to the extent necessary to make use of the rights granted on the Work under this Licence.

3. Communication of the Source Code

The Licensor may provide the Work either in its Source Code form, or as Executable Code. If the Work is provided as Executable Code, the Licensor provides in addition a machine-readable copy of the Source Code of the Work along with each copy of the Work that the Licensor distributes or indicates, in a notice following the copyright notice attached to the Work, a repository where the Source Code is easily and freely accessible for as long as the Licensor continues to distribute or communicate the Work.

4. Limitations on copyright

Nothing in this Licence is intended to deprive the Licensee of the benefits from any exception or limitation to the exclusive rights of the rights owners in the Work, of the exhaustion of those rights or of other applicable limitations thereto.

5. Obligations of the Licensee

The grant of the rights mentioned above is subject to some restrictions and obligations imposed on the Licensee. Those obligations are the following:

Attribution right: The Licensee shall keep intact all copyright, patent or trademarks notices and all notices that refer to the Licence and to the disclaimer of warranties. The Licensee must include a copy of such notices and a copy of the Licence with every copy of the Work he/she distributes or communicates. The Licensee must cause any Derivative Work to carry prominent notices stating that the Work has been modified and the date of modification.

Copyleft clause: If the Licensee distributes or communicates copies of the Original Works or Derivative Works, this Distribution or Communication will be done under the terms of this Licence or of a later version of this Licence unless the Original Work is expressly distributed only under this version of the Licence — for example by communicating 'EUPL v. 1.2 only'. The Licensee (becoming Licensor) cannot offer or impose any additional terms or conditions on the Work or Derivative Work that alter or restrict the terms of the Licence.

Compatibility clause: If the Licensee Distributes or Communicates Derivative Works or copies thereof based upon both the Work and another work licensed under a Compatible Licence, this Distribution or Communication can be done under the terms of this Compatible Licence. For the sake of this clause, 'Compatible Licence' refers to the licences listed in the appendix attached to this Licence. Should the Licensee's obligations under the Compatible Licence conflict with his/her obligations under this Licence, the obligations of the Compatible Licence shall prevail.

Provision of Source Code: When distributing or communicating copies of the Work, the Licensee will provide a machine-readable copy of the Source Code or indicate a repository where this Source will be easily and freely available for as long as the Licensee continues to distribute or communicate the Work.

Legal Protection: This Licence does not grant permission to use the trade names, trademarks, service marks, or names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the copyright notice.

6. Chain of Authorship

The original Licensor warrants that the copyright in the Original Work granted hereunder is owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each Contributor warrants that the copyright in the modifications he/she brings to the Work are owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each time You accept the Licence, the original Licensor and subsequent Contributors grant You a licence to their contributions to the Work, under the terms of this Licence.

7. Disclaimer of Warranty

The Work is a work in progress, which is continuously improved by numerous Contributors. It is not a finished work and may therefore contain defects or 'bugs' inherent to this type of development.

For the above reason, the Work is provided under the Licence on an 'as is' basis and without warranties of any kind concerning the Work, including without limitation merchantability, fitness for a particular purpose, absence of defects or errors, accuracy, non-infringement of intellectual property rights other than copyright as stated in Article 6 of this Licence.

This disclaimer of warranty is an essential part of the Licence and a condition for the grant of any rights to the Work.

8. Disclaimer of Liability

Except in the cases of wilful misconduct or damages directly caused to natural persons, the Licensor will in no event be liable for any direct or indirect, material or moral, damages of any kind, arising out of the Licence or of the use of the Work, including without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, loss of data or any commercial damage, even if the Licensor has been advised of the possibility of such damage. However, the Licensor will be liable under statutory product liability laws as far such laws apply to the Work.

9. Additional agreements

While distributing the Work, You may choose to conclude an additional agreement, defining obligations or services consistent with this Licence. However, if accepting obligations, You may act only on your own behalf and on your sole responsibility, not on behalf of the original Licensor or any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against such Contributor by the fact You have accepted any warranty or additional liability.

10. Acceptance of the Licence

The provisions of this Licence can be accepted by clicking on an icon 'I agree' placed under the bottom of a window displaying the text of this Licence or by affirming consent in any other similar way, in accordance with the rules of applicable law. Clicking on that icon indicates your clear and irrevocable acceptance of this Licence and all of its terms and conditions.

Similarly, you irrevocably accept this Licence and all of its terms and conditions by exercising any rights granted to You by Article 2 of this Licence, such as the use of the Work, the creation by You of a Derivative Work or the Distribution

or Communication by You of the Work or copies thereof.

11. Information to the public

In case of any Distribution or Communication of the Work by means of electronic communication by You (for example, by offering to download the Work from a remote location) the distribution channel or media (for example, a website) must at least provide to the public the information requested by the applicable law regarding the Licensor, the Licence and the way it may be accessible, concluded, stored and reproduced by the Licensee.

12. Termination of the Licence

The Licence and the rights granted hereunder will terminate automatically upon any breach by the Licensee of the terms of the Licence.

Such a termination will not terminate the licences of any person who has received the Work from the Licensee under the Licence, provided such persons remain in full compliance with the Licence.

13. Miscellaneous

Without prejudice of Article 9 above, the Licence represents the complete agreement between the Parties as to the Work.

If any provision of the Licence is invalid or unenforceable under applicable law, this will not affect the validity or enforceability of the Licence as a whole. Such provision will be construed or reformed so as necessary to make it valid and enforceable.

The European Commission may publish other linguistic versions or new versions of this Licence or updated versions of the Appendix, so far this is required and reasonable, without reducing the scope of the rights granted by the Licence. New versions of the Licence will be published with a unique version number.

All linguistic versions of this Licence, approved by the European Commission, have identical value. Parties can take advantage of the linguistic version of their choice.

14. Jurisdiction

Without prejudice to specific agreement between parties,

- any litigation resulting from the interpretation of this License, arising between the European Union institutions, bodies, offices or agencies, as a Licensor, and any Licensee, will be subject to the jurisdiction of the Court of Justice of the European Union, as laid down in article 272 of the Treaty on the Functioning of the European Union,
- any litigation arising between other parties and resulting from the interpretation of this License, will be subject to the exclusive jurisdiction of the competent court where the Licensor resides or conducts its primary business.

15. Applicable Law

Without prejudice to specific agreement between parties,

- this Licence shall be governed by the law of the European Union Member State where the Licensor has his seat, resides or has his registered office,
- this licence shall be governed by Belgian law if the Licensor has no seat, residence or registered office inside a European Union Member State.

Appendix

'Compatible Licences' according to Article 5 EUPL are:

- GNU General Public License (GPL) v. 2, v. 3
- GNU Affero General Public License (AGPL) v. 3
- Open Software License (OSL) v. 2.1, v. 3.0
- Eclipse Public License (EPL) v. 1.0

- CeCILL v. 2.0, v. 2.1
- Mozilla Public Licence (MPL) v. 2
- GNU Lesser General Public Licence (LGPL) v. 2.1, v. 3
- Creative Commons Attribution-ShareAlike v. 3.0 Unported (CC BY-SA 3.0) for works other than software
- European Union Public Licence (EUPL) v. 1.1, v. 1.2
- Québec Free and Open-Source Licence — Reciprocity (LiLiQ-R) or Strong Reciprocity (LiLiQ-R+).

The European Commission may update this Appendix to later versions of the above licences without producing a new version of the EUPL, as long as they provide the rights granted in Article 2 of this Licence and protect the covered Source Code from exclusive appropriation.

All other changes or additions to this Appendix require the production of a new EUPL version.

https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12

PYTHON MODULE INDEX

e

INDEX

H

I

K

L

M

N

O

P

R

S

U

V

Y