Euromod Connector

build unknown

The Euromod Connector package for Python is a user-friendly tool that implements the tax-benefit microsimulation model EUROMOD in Python. It provides functionalities that allow the user to

- inspect the tax-benefit policy systems of the countries modelled in EUROMOD
- define and modify policy rules and parameters
- perform multiple simulations in a loop

Installation

Requirements

The Euromod Connector package requires two **EUROMOD** components: 1) the model files that contain the country-specific coded policy rules, and 2) the input microdata files 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.

Download and install the Euromod Connector package for Python using the *pip* command:

```
pip install euromod
```

Import the Euromod Connector *Model* module:

```
from euromod import Model
```

Managing Errors

Note: If the module *Model* is not found, add to %SYSTEMPATH the root folder where the Euromod Connector package is installed (for example, `r"C:\Users\YOUR_USER_NAME\AppData\Roaming\Python\PythonXXX\site-packages"`) then retry:

```
import sys
sys.path.insert(0, ROOT_PATH_TO_EUROMOD_PACKAGE)
from euromod import Model
```

Note: If you get a "missing attribute" error message regarding the *clr* module, then there is probably a conflict with a different *clr* package, which is installed on your device. Try to uninstall the *clr* package and, if necessary, re-install the *pythonnet* package:

```
pip uninstall clr
pip install pythonnet
```

Equivalent commands in **cmd**:

```
py -m pip install euromod
py -m euromod
```

```
set PATH=%PATH%; "C:\...\AppData\Roaming\Python\PythonXXX\site-packages"
py -m pip uninstall clr
py -m pip install pythonnet
```

Note: Depending on the Python editor, running the code multiple times can cause an import error of the *clr* module from *pythonnet*, which looks as follows:

```
RuntimeError: Failed to initialize Python.Runtime.dll

Failed to initialize pythonnet: System.InvalidOperationException: This property must
be set before runtime is initialized
   at Python.Runtime.Runtime.set_PythonDLL(String value)
   at Python.Runtime.Loader.Initialize(IntPtr data, Int32 size)
   at Python.Runtime.Runtime.set_PythonDLL(String value)
   at Python.Runtime.Loader.Initialize(IntPtr data, Int32 size)
```

For a one-time solution start a new console window. To solve the problem temporarely (or permanently) for a specific console, disable the option **User Module Reloader (UMR)** in the `Tools` bar (this prevents python from automatically reloading modules whenever they are re-imported). Depending on the Python editor/version, go to:

- Tools -> Preferences -> Python Interpreter, OR
- Tools -> Console -> Advaned setting

  Disable the User Module Reloader (UMR) option then press `Apply` and `Ok`. Start a new Console window. Note: this console will load properly the *clr* module even if the UMR option has been reactivated in the meantime. However, if you open a new console the UMR option must be again disabled and the console re-stared.

Working with the Euromod Connector

The Euromod Connector Model object

Initializing the Euromod Connector object by calling the main module `Model`, which takes on one required input of type `str`, i.e. the full path to the root directory where the EUROMOD model is located.

```
In [1]: mod=Model(r"C:\...\EUROMOD_RELEASES_I6.0+")
In [2]: mod
Out[2]: <core.Model at 0x28e67633cd0>
```

The new object `mod` is an array object of the Euromod Connector `Model` class with two attributes: `model_path`, and `countries` that stores the initialized `Country` objects for the **EUROMOD** default countries:

```
In [3]: mod.countries
Out[3]:
AT
BE
BG
...
```

The Euromod Connector Country object

List of the country-object attributes:

Name	Description
model	Returns the `Model` object of the Euromod Connector (of type *class*).
name	Get the two-letter country name of the `Country` object (of type *char*).
systems	Array of `System` objects (of type *class*). ***Note**: only available after calling `load()`.

List of the country-object methods:

Name	Description
get_system_info	***Note**: only applies after the call to `load()`.
load	Load the **EUROMOD** country model in the `Country` object.
load_data	Load data from a .csv file in a Pandas DataFrame.

> ***Note:*** *The `Country` object attributes and methods can be accessed directly from the object `mod` or from the attribute `countries` of the object `mod`. In both the cases, use the array notation either with the index number or with the two-letter country name, and the `.` attribute specification.*

For example, the following commands all produce the same output:

```
In [4]: # mod[21].name
In [4]: # mod['PL'].name
In [4]: # mod.countries[21].name
In [4]: mod.countries['PL'].name
Out[4]: 'PL'
```

Loading the **EUROMOD** country model (for country PL) by using `load()`:

```
In [5]: mod[21].load()
```

The Euromod Connector System object

List of the system-object attributes:

Name	Description
bestMatchDatasets	Get a list of dataset names with best match for the system (of type *list*).
comment	Get any comment related to the system (of type *char*).
country	Returns the country-specific `Model` object.
currencyOutput	Get the currency of the simulation output (of type *char*).
currencyParam	Get the currency of the system parameters (of type *char*).
datasets	Get a list of dataset names that match the system (of type *list*).

headDefInc	Get the main income definition for tax base (of type *char*).
iD	Get the system ID (of type *char*).
name	Get the system name (of type *char*).
order	Get the system order (of type *char*).
private	Get the system ID (of type *char*).
year	Get the system year (of type *char*).

List of the system-object methods:

Name	Description
run	Get the system ID (of type *char*).

When calling the method `load()`, the system models of the country are loaded in the Euromod Connector object as an array of `System` objects.

> ***Note:*** *The `System` object attributes and methods can be accessed directly from the `Country` object or from the attribute `systems` of the object `Country`. In both the cases, use the array notation either with the index number or with the two-letter country name, and the `.` attribute specification.*

For example, the following commands all produce the same output:

```
In [6]: # mod[21][17]
In [6]: # mod[21].systems[17]
In [6]: # mod.countries[21].systems[17]
In [6]: mod.countries['PL'].systems['PL_2022']
Out[6]:
System PL_2022
```

Displaying the datasets that best match the systems by using the attribute `bestMatchDatasets`:

```
In [7]: for sys in mod[21].systems:
            print([sys.name, sys.bestMatchDatasets, sys.currencyParam])
Out[7]:
['PL_2005', ['pl_2007_b3'], 'national']
['PL_2006', ['pl_2007_b3'], 'national']
['PL_2007', ['PL_2008_b4'], 'national']
['PL_2008', ['PL_2008_b4'], 'national']
['PL_2009', ['PL_2010_b5'], 'national']
['PL_2010', ['PL_2010_b5'], 'national']
['PL_2011', ['PL_2012_b6'], 'national']
['PL_2012', ['PL_2012_b6'], 'national']
...
['PL_2021', ['PL_2020_b2'], 'national']
['PL_2022', ['PL_2020_b2'], 'national']
```

Simulating the EUROMOD tax-benefit models

The simulation can be performed by calling the method `run()` from the `System` object. It returns a Euromod Connector `Simulation` object with datasets of the simulation result and additional related information. The main attributes of the `Simulation` object are: `configSettings`, `constantsToOverwrite`, `errors`, `name`, `outputs`.

List of the parameters of method run():

Name	Description
data	*pandas.DataFrame*. Input dataframe passed to the EUROMOD model.
ID_DATASET	*char* or *str*. Name of the dataset. ***Note:** The name of the dataset determines the year of the uprating factros to use in the simulation.
constantsToOverwrite	(**Optional**) *Dict[Tuple[str, str], str]*. A list of constants to overwrite. Note that the key is a tuple for which the first element is the name of the constant and the second string the groupnumber. *Default*: None.
verbose	(**Optional**) *bool*. If True then information on the output will be printed. *Default*: True.
outputpath	(**Optional**) *str*. When an output path is provided, there will be an output file generated. *Default*: "".
addons	(**Optional**) *List[Tuple[str, str]]*. List of addons to be integrated in the spine, where the first element of the tuple is the name of the Addon (available addons are: LMA, MTR, NRR, TCA) and the second element is the name of the system in the Addon to be integrated (typically, it is the name of the Addon _ two-letter country name, e.g. LMA_AT). *Default*: [].
switches	(**Optional**) *List[Tuple[str, bool]]*. List of Extensions to be switched on or off. The first element of the tuple is the short name of the Addon. The second element is a boolean. *Default*: [].

Simulating with default optional parameters: Running the simulation for two systems by using the method `run()` with two required input parameters, after loading the data as a *pandas.DataFrame*:

```
In [8]: data=pd.read_csv("PL_2020_b2.txt",sep="\t")
In [9]: out=[]
In [10]: for sysnam in ['PL_2021','PL_2022']:
             out.append(mod['PL'][sysnam].run(data,"PL_2020_b2.txt"))
Out[10]:
Simulation: Sim1, System: PL_2021, Data: PL_2020_b2.txt  .. done!   Time to
simulate16.4692988395690925

Simulation: Sim2, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate13.6837193965911875

In [11]: out
Out[11]: [
```

```
name:                 Sim1
output:               pl_2021_std.txt ,

name:                 Sim2
output:               pl_2022_std.txt
]
```

Displaying the simulation results by calling the attribute `outputs`, that can be indexed by an `int` or the name of the dataset as `str` :

```
In [12]: out1 = out[1]
In [13]: out1.outputs['pl_2022_std.txt']
Out[13]:
idhh     idperson  ...      il_bsamt      il_bsatm
0          100.0      10001.0  ...  14504.920877  14504.920877
1          100.0      10002.0  ...   6297.556928   6297.556928
          ...          ...  ...           ...           ...
38640  2047300.0  204730001.0  ...   1476.410557   1476.410557
38641  2047500.0  204750001.0  ...   2733.061980   2733.061980

[38642 rows x 454 columns]
```

Displaying the configuration settings used in the simulation (e.g. the names of the system and dataset or the user's configuration of addons and extensions) by calling the attribute `configSettings`:

```
In [14]: out1.configSettings
Out[14]:
{'PATH_EUROMODFILES': 'C:\\...\\EUROMOD_RELEASES_I6.0+',
 'PATH_DATA': 'C:\\...\\EUROMOD_RELEASES_I6.0+\\Input',
 'PATH_OUTPUT': '',
 'ID_DATASET': 'PL_2020_b2.txt',
 'COUNTRY': 'PL',
 'ID_SYSTEM': 'PL_2022'}
```

The attribute `constantsToOverwrite` stores the user's configuration of modified constants. The attribute `errors` collects the error/warning messages, if any, produced by **EUROMOD** during the simulation.

Simulating changing the values of constants: Running the simulation of a system by using the method `run()` with two required input parameters and one optional input parameter, `constantsToOverwrite`. This parameter overwrites the **EUROMOD** default values of the constants with the ones specified bu the user. Define the parameter as a `dict`, where the key is a 1x2 `tuple` of `str` with the first string specifying the name of the constant and the second string its groupnumber, and the value is a `str` with the new constant value. The default is None.:

```
In [15]: out=mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",constantsToOverwrite=
{("$f_h_cpi","2022"):'10000'})
Out[15]:
Simulation: Sim3, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate15.7604472637176515
```

```
In [16]: out.constantsToOverwrite
Out[16]: {('$f_h_cpi', '2022'): '10000'}
```

Simulating using the EUROMOD addons: Running the simulation of a system by using the method `run()` with two required input parameters and one optional input parameter, `addons`. This optional parameter is a `list` of 1x2 `tuple` of `str` defining the addons to be integrated in the spine. The first element of the `tuple` is the `str` name of the Addon (available addons are: LMA, MTR, NRR, TCA) and the second element is the `str` name of the system in the Addon to be integrated (typically, it is the name of the Addon `_` two-letter country name, e.g. LMA_AT). The default is [].

```
In [17]: out =mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",addons=
[("LMA","LMA_PL")])
Out[17]:
Simulation: Sim4, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate18.564006567001343s

In [18]: out
Out[18]:
name:                 Sim4
output:               pl_2022_lma.txt

In [19]: out.configSettings
Out[19]:
{'PATH_EUROMODFILES': 'C:\\...\\EUROMOD_RELEASES_I6.0+',
 'PATH_DATA': 'C:\\...\\EUROMOD_RELEASES_I5.0+\\Input',
 'PATH_OUTPUT': '',
 'ID_DATASET': 'PL_2020_b2.txt',
 'COUNTRY': 'PL',
 'ID_SYSTEM': 'PL_2022',
 'ADDON0': 'LMA|LMA_PL'}
```

Simulating switching on/off the extensions: Running the simulation of a system by using the method `run()` with two required input parameters and one optional input parameter, `switches`. This optional parameter is a `list` of 1x2 `tuple` defining the extensions to be switched on or off. The first element of the `tuple` is the `str` short name of the Addon. The second element is a `boolean`. The default is [].

```
In [20]: out =mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",switches=[("LMA",True)])
Out[20]:
Simulation: Sim5, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate18.564006567001343s

In [21]: out
Out[21]:
name:                 Sim5
output:               pl_2022_lma.txt

In [22]: out.configSettings
Out[22]:
{'PATH_EUROMODFILES': 'C:\\...\\EUROMOD_RELEASES_I6.0+',
 'PATH_DATA': 'C:\\...\\EUROMOD_RELEASES_I5.0+\\Input',
 'PATH_OUTPUT': '',
```

```
 'ID_DATASET': 'PL_2020_b2.txt',
 'COUNTRY': 'PL',
 'ID_SYSTEM': 'PL_2022',
 'EXTENSION_SWITCH0': 'LMA=on'}
```

License

We kindly ask you to acknowledge the use of EUROMOD in any publications or other outputs (such as conference presentations). A recommended wording for acknowledgement is provided below:

```
'The results presented here are based on EUROMOD version I5.0+. Originally
maintained, developed and managed by the Institute for Social and Economic
Research (ISER), since 2021 EUROMOD is maintained, developed and managed by
the Joint Research Centre (JRC) of the European Commission, in collaboration
with EUROSTAT and national teams from the EU countries. We are indebted to the
many people who have contributed to the development of EUROMOD. The results
and their interpretation are the author's(') responsibility'
```

This package includes one icon ('\XMLParam\AddOns\MTR\MTR.png'), adapted from LibreICONS, under :