Euromod Connector

build unknown

Syntax

class `core.Model`(**model_path**)

Parameters:

model_path: **string**
Path to the EUROMOD root directory.

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

Index:

Installation

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

```
pip install euromod
```

Requirements

The Euromod Connector requires two EUROMOD 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.

The Dependencies section below lists other required dependencies .

Working with the Euromod Connector

Import the Euromod Connector as follows:

```
from euromod import Model
```

Creating an object of the `core.Model` class by passing a `str` path to the EUROMOD root directory:

```
In [1]: mod=Model(r"C:\...\EUROMOD_RELEASES_I6.0+")
In [2]: mod
```

```
Out[2]: <core.Model at 0x28e67633cd0>
```

Object `mod` has two attributes: `model_path`, and `countries` which stores the instantiated `core.Country` classes for the EUROMOD default countries.

> ***Note:*** *Objects can be accessed using a single integer or a label. For the `core.Country` object the label is a two-letter country name, for the `core.System` object it is the system's name, for the `core.Simulation` object it is the name of the simulation output dataset (Examples: `Model['PL']['PL_2020']`, `Model[3][10]`).*

> ***Note:*** *Countries in EUROMOD use the two-letter country codes convention. Please, see the Eurostat Glossary: Country codes.*

Use `core.Model.countries` to access the country object(s):

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

In [4]: # The following commands are equivalent:
        # mod.countries['PL']
        # mod.countries[21]
        mod[21]
Out[4]:
Country PL
```

Displaying country's name using the `core.Country.name` attribute:

```
In [5]: # The following commands are equivalent:
        # mod[21].name
        # mod['PL'].name
        # mod.countries[21].name
        mod.countries['PL'].name
Out[5]: 'PL'
```

Method `core.Country.load()` instantiates new `core.System` class objects for each tax-benefit system policy:

```
In [6]: mod[21].load()
```

Accessing the system object(s) calling the attribute `core.System.systems`:

```
In [7]: mod['PL'].systems
Out[7]:
PL_2005
PL_2006
PL_2007
...

In [7]: # The following commands are equivalent:
        # mod[21][17]
```

```
        # mod[21].systems[17]
        # mod.countries[21].systems[17]
        mod.countries['PL'].systems['PL_2022']
Out[7]:
System PL_2022
```

Display some system information using attributes `name`, `bestMatchDatasets` and `currencyParam` of the `core.System` object:

```
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_2021', ['PL_2020_b2'], 'national']
['PL_2022', ['PL_2020_b2'], 'national']
```

The method run()

Syntax

`core.System.run`(**data**,**ID_DATASET**,**constantsToOverwrite**=[],**verbose**=True,**outputpath**="",**addons**=[],**switches**=[])

Run simulations of a EUROMOD tax-benefit system.

Return a `core.Simulation` class object with simulation results and other configuration information.

Parameters:

data: **pandas.DataFrame**
Path to the EUROMOD root directory.

ID_DATASET: **str**
Name of the dataset. ***Note:** It determines the year of the uprating factros to use in the simulation.

constantsToOverwrite: **dict of {tuple(str,str): str}, default []**
A list of constants to overwrite. Note that the key is a tuple for which the first item is the name of the constant and the second is the group number.

verbose: **bool, default True**
If True then information on the output will be printed.

outputpath: **str, default ""**
When an output path is provided, there will be an output file generated.

addons: **list of [tuple(str,str)], default []**
List of addons to be integrated in the spine. The first item of the tuple is the name of the Addon, the second item 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). Available Addons are: LMA, MTR, NRR, TCA.

switches: **list of [tuple(str,bool)], default []**
List of Extensions to be switched on or off. The first item of the tuple

```
        is the short name of the Extension, the second item is a boolean.
        Available Extensions are: BTA, TCA, FYA, UAA, EPS, PBE,MWA, HHoT_un, WEB,
        HHoT_ext, HHoT_ncp.
```

1. Example: Simulating two systems with default optional parameters:

```
In [8]: data=pd.read_csv("PL_2020_b2.txt",sep="\t")
        out=[]
        for sysnam in ['PL_2021','PL_2022']:
            out.append(mod['PL'][sysnam].run(data,"PL_2020_b2.txt"))

Out[8]:
Simulation: Sim1, System: PL_2021, Data: PL_2020_b2.txt  .. done!   Time to
simulate16.469298839569092s
Simulation: Sim2, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate13.683719396591187s

In [9]: out
Out[9]: [
 name:                 Sim1
 output:               pl_2021_std.txt ,

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

Accessing the simulation results by indexing `core.Simulation.outputs` with the name of the dataset provided in the attribute `core.Simulation.output`:

```
In [10]: out1 = out[1]
         out1.outputs['pl_2022_std.txt']
Out[10]:
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]
```

`core.Simulation.configSettings` shows the simulation configuration settings:

```
In [11]: out1.configSettings
Out[11]:
{'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'}
```

Attribute `core.Simulation.configSettings` is a `struct` collecting the information about the system, dataset, addons, extensions, and other configuration settings used in the simulation.

2) Example: Simulating changing the values of constants by passing parameter `constantsToOverwrite` to `run()`:

```
In [12]: out=mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",constantsToOverwrite=
{("$f_h_cpi","2022"):'10000'})
Out[12]:
Simulation: Sim3, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate15.7604472637176 51s

In [13]: out.constantsToOverwrite
Out[13]: {('$f_h_cpi', '2022'): '10000'}
```

The optional parameter `constantsToOverwrite` specifies which constants to overwrite. `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.

3) Example: Simulating including the EUROMOD Addons by passing parameter `addons` to `run()`:

```
In [14]: out =mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",addons=
[("LMA","LMA_PL")])
Out[14]:
Simulation: Sim4, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate18.564006567001343s

In [15]: out
Out[15]:
name:                 Sim4
output:               pl_2022_lma.txt

In [16]: out.configSettings
Out[16]:
{'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'}
```

The optional parameter `addons` 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). Available Addons are: LMA, MTR, NRR, TCA. The default is [].

4) Example: Simulating switching on/off the Extensions by passing parameter `switches` to `run()`:

```
In [17]: out =mod['PL']['PL_2022'].run(data,"PL_2020_b2.txt",switches=[("BTA",True)])
Out[17]:
Simulation: Sim5, System: PL_2022, Data: PL_2020_b2.txt  .. done!   Time to
simulate18.564006567001343s

In [18]: out
Out[18]:
name:                 Sim5
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',
 'EXTENSION_SWITCH0': 'BTA=on'}
```

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`. Available Extensions are: BTA, TCA, FYA, UAA, EPS, PBE, MWA, HHoT_un, WEB, HHoT_ext, HHoT_ncp. The default is [].

List of Attributes:

Model class attributes:

model_path	Return a string with the path to the EUROMOD root directory.
countries	Access `Country` class objects.

Country class attributes:

model	Access `Model` class object.
name	Return a string with the name of the country.
systems	Access the `System` class objects. ***Note**: Available after the `load()`.

System class attributes:

bestMatchDatasets	Return a list of dataset names with best match for the system.
comment	String comment related to the country-system.
country	Access the `Country` class objects.

currencyOutput	Return a string with the currency of the simulation output.
currencyParam	Return a string with currency of the system parameters.
datasets	Return a list of dataset names that match the system.
headDefInc	Return a string with the main income definition for the tax base.
iD	Return a string with the system identifier.
name	Return a string with the name of the system.
order	Return a string defining the system order.
private	Return a string with the system access.
year	Return a string with the system year.

Simulation class attributes:

configSettings	Dictionary of configuration settings used in the simulation (including addons and extensions).
constantsToOverwrite	Dictionary with constants that are overwritten in the simulation.
errors	String Error/warning messages produced by **EUROMOD** during the simulation.
currencyOutput	Return a string with the currency of the simulation results.
name	Return a string with the name of the output dataset.
outputs	Return a list of datasets with simulation results.

List of Methods:

Country class methods:

load()	Load the **EUROMOD** tax-benefit systems in the `Country` object.
load_data()	Load data from a .csv file as a `pandas.DataFrame`.

System class methods:

run([data,ID_DATASET,...])	Run simulations of the **EUROMOD** tax-benefit systems.

Dependencies

The Euromod Connector requires the following dependencies:

Package	Minimum supported version

pandas	2.0.3
pythonnet	3.0.2

Managing Errors

1) ModuleNotFoundError or **AttributeError**: If the `import` of the Euromod Connector libreries fails with one of the messages below:

```
ModuleNotFoundError: No module named 'System'
```

```
AttributeError: module 'clr' has no attribute 'AddReference'
```

uninstall the Python *clr* package and re-install the *pythonnet* package:

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

This error is caused by a conflict between the Python *clr* package and the *clr* library of the *pythonnet* package.

2) RuntimeError: If you encounter a `RuntimeError` as below, either 1) restart the kernel, or 2) open a new console window, or 3) deselect the option **User Module Reloader (UMR)** in the `Tools -> Preferences -> Python Interpreter` (or `Tools -> Console -> Advanced setting`, depending on the Python editor version) then press `Apply` and `Ok` and restart the consol windows.

Note: Re-enabling the UMR option has no effect on the console windows that are already open.

This error is produced when Python reloads the libraries of the *pythonnet* package.

```
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)
```

License

```
'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 :