Das Python-Package pyam: Szenario-Analyse, Datenvisualisierung und Schnittstelle zwischen Modellen mit einheitlichem Dateiformat

Dieses Notebook entstand bei der IEWT 2023 in Wien und widmet sich der Frage

> Wie kann der Datenaustausch zwischen verschiedenen Modellen und der Prozess der Analyse- und Visualisierung vereinfacht werden?

Dazu werden folgende Punkte bearbeitet:

- Was ist pyam und das IAMC-Datenformat?
- Welche Schritte sind nötig, um pyam/IAMC-Datenformat in eigenen Modellen oder Projekten zu nutzen?
- Datenabfrage von IIASA-Szenarioexplorer (z.B. zu 1.5°-IPCC-Bericht)
- Visualisierungstools
- Analyse-Framework
- Sammlung von Vor- und Nachteilen von pyam

Table of contents

Vorstellungsrunde

- Aus welchen Fachgebieten stammen die Teilnehmenden?

    - --> überwiegend Energiesystemanalyse
- Warum seid ihr hier?
    - kennenlernen eines neuen Szenario-Analyse-Tools
    - pyam/IAMC-Format wird in Projekten genutzt, aber unklar, wo der Vorteil liegt
    - pyam wird genutzt und es bestehen Fragen, wie es optimaler genutzt werden kann
- Wer nutzt für Datenauswertung (v.a.) welche Tools?
    - überwiegend wird Python genutzt
    - teilweise sind Excel-Auswertungen vorhanden, die mittel-/langfristig ersetzt werden sollen
- Wer hat Vorkenntnisse in Python und pyam?
    - Großteil hat Vorkenntnisse in Python, teilweise wurde auch schon mit pyam gearbeitet

Was ist pyam und das IAMC-Datenformat?

pyam ist ein OpenSource-Python-Package, das Tools zur Datenanalyse und -visualisierung für Energiesystemmodelle und Integrated-Assessment-Modelle bereitstellt.

IAMC-Datenformat

Grundlage für pyam bildet das IAMC-Datenformat, das u.a. von IAMC, IPCC, openENTRANCE und Ariadne verwendet wird. Für jährliche Daten sieht ein Beispieldatensatz etwa so aus:

Model	Scenario	Region	Variable	Unit	2020	2030	2040	2050
REMod	Referenz	DEU	Final Energy	TWh/yr	2543	2329	1949	1720
REMod	Referenz	DEU	Final Energy|Electricity	TWh/yr	501.4	553.5	752.6	896.7
REMod	Referenz	DEU	Final Energy|Industry	TWh/yr	713.4	676.3	633.6	582.1
REMod	Referenz	DEU	Final Energy|Industry|Electricity	TWh/yr	284.1	261.2	334.1	417.7

Das Format setzt sich aus den Spalten `Model`, `Scenario`, `Region`, `Variable`, `Unit` und entweder dem jeweiligen Jahr (z.B. `2020`) oder einer Kombination aus `year` oder `time` und `value` zusammen. Es können dadurch auch Daten mit höherer zeitlicher Auflösung (z.B. repräsentative Zeitpunkte oder stündliche Zeitreihen) dargestellt werden.

Zentral ist die Definition der Variablen. Die Variablendefinitionen haben eine semi-hierarchische Struktur, die über den pipe-Charakter `|` unterteilt wird. Semi-hierarchisch bedeutet dabei, dass die Summe aller Sub-Variablen (z.B. `Final Energy|Electricity` und `Final Energy|Gas`) unterhalb eines pipe-Charakters die Hauptvariable (`Final Energy`) ergeben können, aber auch Abweichungen davon durch weitere Variablen (z.B. `Final Energy|Industry`) möglich sind. Die Variablen müssen/können projektspezifisch definiert werden, wodurch auch eine streng-hierarchische Struktur erzwungen werden kann.

Funktionsüberblick pyam

pyam dient als Framework, um Daten (In- und Output) verschiedener Szenarien oder Modelle analysieren, visualisieren und vergleichen zu können. Mit wenig Aufwand kann ein Workflow aus

- Datenabfrage oder Daten einlesen
- Daten validieren
- Daten aufbereiten
- Grafiken erstellen umgesetzt werden.

In einem kleinen Beispiel ist hier die Abfrage von einem öffentlichen Szenarioexplorer und die Erstellung einer simplen Grafik dargestellt.
Als erstes Fragen wir Daten für installierte Leistungen von Stromerzeugern aus dem Projekt Ariadne ab.

In [ ]:

```
import pyam
import matplotlib.pyplot as plt
%matplotlib inline

# Daten abfragen über read_iiasa
# Variablen aussuchen
variables = (
    ["Capacity|Electricity"]
    + [
        f"Capacity|Electricity|{i}"
        for i in [
            "Biomass",
            "Coal",
            "Gas",
            "Geothermal",
            "Hydro",
            "Hydrogen",
            "Nuclear",
            "Oil",
            "Solar",
            "Storage Converter",
            "Wind",
        ]
    ]
    + ["Final Energy"]
    + [
        f"Final Energy|{i}"
        for i in [
            "Electricity",
            "Gases",
            "Geothermal",
            "Heat",
            "Hydrogen",
            "Liquids",
            "Solar",
            "Solids",
            "Waste",
        ]
    ]
)

# Daten abfragen
df = pyam.read_iiasa('ariadne', variable=variables)
```

Wir führen eine kleine Datenvalidierung durch:

- Sind für alle Modelle und Szenarien die geforderten Daten vorhanden?
- Ist die Summe der Stromerzeuger (`Capacity|Electricity|*`) tatsächlich die angegebene gesamte installierte Leistung (`Capacity|Electricity`)?

In [ ]:
```
# Check for required variables
df.require_variable(variable=variables)

# Check, ob streng hierarich
np_isclose_args = {
    'equal_nan': True,
    'rtol': 1e-02, # relative tolerance
    'atol': 1, # absolute tolerance
}
df.check_aggregate("Capacity|Electricity", **np_isclose_args)
df.check_aggregate("Final Energy", **np_isclose_args)
```

Wir sehen, dass in diesem Datensatz die Modelle `REMod + FORECAST` und `TIMES PanEU` (kleinere) Inkonsistenzen haben. Für weitere Analysen könnten wir diesen Nachgehen oder die Modelierer:innen kontaktieren.

Wir erstellen nun zwei einfache Plots. Installierte Leistung von Stromerzeugern in dem Szenario `8Gt_Bal` des Modells `REMod` :

In [ ]:
```
df.filter(variable='Capacity|Electricity|*', level=0, model='REMod*', scenario='8Gt_Bal').plot.bar
```

und ein Boxplot für die Endenergien über alle Szenarien und Modelle hinweg für das Jahr 2045:

In [ ]:
```
df.filter(variable='Final Energy|*', level=0, year=2045).plot.box(x="variable")
plt.xticks(rotation=90);
```

Wir können dafür auch die Einheiten konvertieren, falls andere Einheiten gewünscht sind.

In [ ]:
```
df.filter(variable='Final Energy|*', level=0, year=2045).convert_unit('TWh/yr', to='PJ/a').plot.bo
plt.xticks(rotation=90);
```

Welche Schritte sind nötig, um pyam/IAMC-Datenformat in eigenen Modellen oder Projekten zu nutzen?

Brainstorming und Diskussion innerhalb des IEWT-Workshops.

- Im Projekt oder Team muss sich auf einheitliche Nomenklatur der Variablen geeinigt werden. Als Ausgangspunkt können dafür Nomenklaturen aus bestehenden Projekten übernommen werden, die dann für das Projekt/Modellteam angepasst werden. --> Einheitliche Nomenklatur ist ein großer Vorteil beim Modellvergleich, kann aber großen Aufwand für die Abstimmung zwischen Modellteams bedeuten
- Die Modelldaten müssen in das IAMC-Format gebracht werden. Je nach Zustand des Modelloutputs bedeutet das größeren Aufwand oder kann leicht umgesetzt werden.
    - besteht eine Datenbank-ähnliche Struktur, können Einleseroutinen von pyam genutzt werden, etwa

durch zusammenfassen mehrerer Spalten zu einer IAMC-Variablen
    - sind die output-Daten schlechter strukturiert und werden z.B. als Matrix ausgegeben, muss ein mapping erstellt werden, welche Zeilen/Spalten der entsprechenden IAMC-Variablen entsprechen --> kann großen Aufwand bedeuten, der Output wird dadurch aber verständlicher
    - einlesen aus csv, excel und json stellt kein Problem dar, siehe Beispiele weiter unten
- Detaillierterer Modelloutput kann in die Variablenstruktur integriert werden, indem weitere Hierarchieebenen durch | hinzugefügt werden.
- als Daten können alle von python einlesbaren Dateiformate genutzt werden. Für große Datensätze bieten sich alternativen zu csv/xlsx-Dateien an, die weniger Speicher benötigen und/oder schneller einzulesen sind. Auf towardsdatascience gibt es einen Vergleich zwischen csv, pickle, messagepack, hdf5, feather und parquet.

Daten einlesen

Daten können aus verschiedenen Import-Dateien eingelesen oder über einen IIASA-Szenarioexplorer abgefragt werden.

aus Import-Dateien

Für eigene Modelldaten bietet es sich an, die Daten im IAMC-Format als `csv`, `xlsx` oder ähnliches abzulegen. Von pyam können sie dann wie folgt eingelesen werden:

In [ ]:

```
import pandas as pd
import pyam

# Daten über Pandas einlesen
df = pd.read_csv('tutorial_data.csv')
# in pyam-Objekt `IamDataFrame` konvertieren
df = pyam.IamDataFrame(df)

# Daten wieder abspeichern über pyam-Funktion
#df.to_csv("tutorial_data.csv")
```

Es werden alle Datenformate unterstützt, die von pandas eingelesen werden können. Weitere Infos gibt es in der Dokumentation von pandas.

Über die Funktion `as_pandas` kann ein IamDataFrame wieder zurück in ein pandas DataFrame konvertiert werden. In pyam gibt es aber auch die Funktionen `to_csv` und `to_excel`, um Daten direkt abzuspeichern und die Möglichkeit, über `df = pyam.IamDataFrame(data='tutorial_data.csv')` Daten direkt aus csv-Dateien einzulesen

Datenabfrage von IIASA-Szenarioexplorer

In [ ]:

```
import pyam
# Verbindung aufbauen
conn = pyam.iiasa.Connection()
# Verbindung validieren und Liste an Szenarioexplorern anzeigen
# Teilweise werden für die Szenarioexplorer Zugangsdaten benötigt
# pyam.iiasa.set_config(<username>, <password>)
conn.valid_connections

# Verbindung zu einem Szenarioexplorer aufbauen
# kann auch direkt beim Aufbau der Verbindung angegeben werden
conn.connect("iamc15")

# Untersuchen, welche Daten verfügbar sind
conn.models().head()
conn.scenarios().head()
conn.variables().head()
conn.regions().head()

# Meta-Daten
conn.meta_columns.head()

# Daten über query Abfragen
df = conn.query(
    model='MESSAGEix*',
    variable=['Emissions|CO2', 'Primary Energy|Coal'],
    region='World'
)
```

In [ ]:

```
# Daten abfragen über read_iiasa
df = pyam.read_iiasa(
    'iamc15',
    model='MESSAGEix*',
    variable=['Emissions|CO2', 'Primary Energy|Coal'],
    region='World',
    meta=['category']
)

# oder bei großen Datensätzen über die Funktion `lazy_read_iiasa`
# dann werden die Daten als csv-Daten abgespeichert und können
# schneller wieder eingelesen werden
'''
lazy_df = pyam.lazy_read_iiasa(
    file="./tmp/messageix_co2_coal_data.csv",
    name="iamc15",
    model='MESSAGEix*',
    variable=['Emissions|CO2', 'Primary Energy|Coal'],
    region='World'
)
'''
```

Daten filtern und Anzeigen

Daten können mit der Funktion `filter` anhand der Attribute `model`, `scenario`, `region`, `variable`, `unit` und `year` gefiltert werden. Mit dem keyword `keep=False` kann die Auswahl umgekehrt werden.

Angezeigt werden können sie über `timeseries` oder ähnlich wie in pandas über `head()`. Die einzelnen Attribute können auch angezeigt werden.

In [ ]:
```
# nochmal Tutorial-Daten einlesen
df = pyam.IamDataFrame(data='tutorial_data.csv')

# Anhand der Jahre, nur alle 5 Jahre
df.filter(year=range(2020, 2051, 5))
# alle Variablen bis zu einem bestimmten Level
df.filter(variable='*', level='2-')

# Möglichkeiten zum Anzeigen
#df
#df.head()
#df.timeseries
#df.model
#df.scenario
#df.region
#df.unit_mapping
```

Visualisierungstools

Es stehen verschiedene Plot-Funktionen zur Verfügung.

In [ ]:
```
df.filter(variable='Capacity|Electricity*', level=1, scenario="01_Referenz").plot()

df.filter(variable='Capacity|Electricity*', level=1, scenario="01_Referenz").plot.bar(stacked=True
```

Berechnungen

In [ ]:
```
# Einheiten konvertieren
# df.convert_unit('GWh', to='TJ')

# Werte aggregieren
# ein paar Variablen rauswerfen
df.filter(variable=['Secondary Energy', 'Secondary Energy|Gases', 'Secondary Energy|Hydrogen'], ke
df.aggregate("Secondary Energy", recursive='skip-validate', append=True)

df.filter(variable='Secondary Energy*').variable

# Grundrechenarten
# df.subtract
# df.multiply
# df.divide
# df.add

# komplexe Berechnungen
# df.apply
```

Analyse-Framework

In [ ]:

```python
# Check for required variables
#df.require_variable(variable="Primary Energy", year=2049)
#df.require_variable(variable="Primary Energy", year=2050)

# validate numerical values
# check, dass Wert mindestens 'lo' ist
# mit keyword `excule_on_fail` können Datenpunkte entfernt werden
df.validate(criteria={'Primary Energy': {'lo': 540 * 0.9, 'year': 2020}})

# Check, ob streng hierarich
# df.check_internal_consistency()
```

Vor- und Nachteile von pyam / IAMC-Format

gemeinsame Sammlung aus dem IEWT-Workshop

Vorteile

- Standardisierte Variablenkonventionen erleichtern Modellvergleich in großen Projekten
- gemeinsame Weiterentwicklung von pyam führt zu Synergien, von Entwicklungen anderer kann profitiert werden
- automatisierte Auswertungsroutinen, v.a. im Vergleich zu Excel-basierten Tools
- schneller Einblick in neue Datensätze (eigene neue Rechnungen, Daten von anderen Modellen/Projekten)

Nachteile

- viel Arbeit, sich auf ein gemeinsames Variablentemplate zu einigen
- teilweise passen bestehende Datenbanken besser zu Modellen als das IAMC-Format
- Berechnungen direkt in pyam können recht langsam sein --> für Aufbereitung des roh-Outputs werden teilweise andere packages benötigt