Multi-period modeling in oemof.solph

An Overview

Motivation | Usage Example | Implementation
Motivation

*Depict long-term developments taking into account unit lifetimes*

- **Status quo** in oemof.solph
  - Only one timestep where investments may occur → \( t=0 \), i.e. the begin of the optimization
  - Investments are accounted for by their annuity
  - Timesteps are a one-dimensional set
  - Classical use case: Dimensioning of a system; short-term view (e.g. one typical year)

- **Idea of a multi-period optimization** model
  - Depict different periods in addition to different timesteps → there are two time-related indices
  - Investments may occur in every period
  - Lifetime tracking
  - Classical use case: long-term planning of a system
Example usage

Set `multi_period` attribute of `EnergySystem` to `True`.

```python
import pandas as pd
import oemof.solph as solph

my_index = pd.date_range('1/1/2013', periods=17520, freq='H')
my_energysystem = solph.EnergySystem(timeindex=my_index, multi_period=True)
```

By default consider one year to be equal to one period; Periods can also be defined explicitly.

set `multi_period` attribute to `True` (default=`False`)
Example usage

No need for changes to your „dispatch-related“ system

```
hydrogen_bus = solph.buses.Bus(label="hydrogen")
coal_bus = solph.buses.Bus(label="coal")
electricity_bus = solph.buses.Bus(label="electricity")

hydrogen_source = solph.components.Source(
    label="green_hydrogen",
    outputs={
        hydrogen_bus: solph.flows.Flow(
        )
    },
)

coal_source = solph.components.Source(
    label="hardcoal",
    outputs={
    },
)

electricity_sink = solph.components.Sink(
    label="electricity_demand",
    inputs={
        electricity_bus: solph.flows.Flow(
            nominal_value=1000, fix=[0.8] * len(my_index)
        )
    },
)
```

Variable costs defined in TIMESTEPS.

No changes to your system for units that is not invested into.
Example usage

Investments have new attributes

```python
hydrogen_power_plant = solph.components.Transformer(
    label="hydrogen_pp",
    Inputs={(hydrogen_bus: solph.flows.Flow(),),
    outputs={(electricity_bus: solph.flows.Flow(
        investment=solph.Investment(
            maximise=1000,
            ep_costs=100,
            lifetime=30,
            interest_rate=0.06,
            fixed_costs=100,
        ),
        variable_costs=3,
    ),
    conversion_factors={(electricity_bus: 0.6),
    })
```

- `ep_costs` used as specific investment expenses; nominal values on a period / annual basis.
- `lifetime` and `age` (optional; defaults to 0) for lifetime tracking and decommissioning.
Implementation – Main changes
Overview on the main changes

- Add `multi_period` attribute and `periods` attribute to `EnergySystem` class.
- Add `discount_rate` to `Model`.
- Create new timesets:
  - `PERIODS`: used for investment variables
  - `TIMEINDEX`: tuple of (period, timestep)
- Index `flow` variable in `TIMEINDEX`. Adjust every constraint that includes a flow; but do not change indexing of variables indexed in `TIMESTEPS`.
- Add additional attributes to `_options.Investment`, `GenericStorage` and `SinkDSM`: `lifetime`, `age`, `interest_rate`, `fixed_costs`.
- Add variables `old`, `old_end` and `old_exo` in investment blocks for lifetime tracking.
- Add adjusted objective value terms for multi-period model including discounting and annuities for investment.
- Adjust / restructure processing to properly retrieve results.
constructor of EnergySystem – multi_period and periods attributes

```python
if multi_period:
    msg = (  
        "CAUTION! You specified 'multi_period=True' for your "  
        "energy system.\nThis will lead to creating "  
        "a multi-period optimization modeling which can be "  
        "used e.g. for long-term investment modeling.\n"  
        "Please be aware that the feature is experimental as of "  
        "now. If you find anything suspicious or any bugs, "  
        "please report them."
    )
    warnings.warn(msg, debugging.SuspiciousUsageWarning)

self.multi_period = multi_period
self.periods = self._add_periods(periods)
self._extract_periods_years()
```

method EnergySystem._add_periods

```python
if not self.multi_period:
    periods = {0: 0}
elif periods is None:
    years = sorted(list(set(getattr(self.timeindex, "year"))))
    periods = {}
    filter_series = self.timeindex.to_series()
    for number, year in enumerate(years):
        start = filter_series.loc[
            filter_series.index.year == year
        ].min()
        end = filter_series.loc[filter_series.index.year == year].max()
        periods[number] = pd.date_range(start, end, freq="H")
```
Implementation – oemof.solph._models.py

**constructor of Model – discount_rate attribute**

```python
def __init__(self, energysystem, discount_rate=None, **kwargs):
    if discount_rate is not None:
        self.discount_rate = discount_rate
    elif energysystem.multi_period:
        self.discount_rate = 0.02
        msg = (
            f"By default, a discount_rate of {self.discount_rate} "
            f"is used for a multi-period model. "
            f"If you want to use another value, "
            f"you have to specify the `discount_rate` attribute."
        )
        warnings.warn(msg, debugging.SuspiciousUsageWarning)
```

Implementation – oemof.solph._models.py

method Model._add_parent_block_sets

```python
if not self.es.multi_period:
    self.TIMEINDEX = po.Set(
        initialize=list(
            zip(
                [0] * len(self.es.timeindex),
                range(len(self.es.timeindex)),
            ),
            ordered=True,
        )
    ).
else:
    nested_list = [
        [k] * len(self.es.periods[k]) for k in self.es.periods.keys()
    ]
    flattened_list = [
        item for sublist in nested_list for item in sublist
    ]
    self.TIMEINDEX = po.Set(
        initialize=list(
            zip(flattened_list, range(len(self.es.timeindex)))
        ),
        ordered=True,
    )
    self.PERIODS = po.Set(
        initialize=sorted(list(set(self.es.periods.keys())))
    )

# (Re-)Map timesteps to periods
for p, t in self.TIMEINDEX:
    timeslots_in_period[p].append(t)
self.TIMESTEPS_IN_PERIOD = timeslots_in_period
```

multi-period modeling in oemof.solph | J. Kochems & J. Giehl

18/05/2022
method Model._add_parent_block_variables

```python
self.flow = po.Var(self.FLOWS, self.TIMEINDEX, within=po.Reals)

for (o, i) in self.FLOWS:
    if self.flows[o, i].nominal_value is not None:
        if self.flows[o, i].fix[self.TIMESTEPS[i]] is not None:
            for p, t in self.TIMEINDEX:
                self.flow[o, i, p, t].value = (
                    self.flows[o, i].fix[t] * self.flows[o, i].nominal_value
                )
        self.flow[o, i, p, t].fix()
```
Implementation – oemof.solph._options.Investment

- **overall_maximum** (float, \( P_{\text{overall,max}} \) or \( E_{\text{overall,max}} \)) – Overall maximum capacity investment, i.e. the amount of capacity that can be totally installed at maximum in any period (taking into account decommissionings); only applicable for multi-period models.
- **overall_minimum** (float, \( P_{\text{overall,min}} \) or \( E_{\text{overall,min}} \)) – Overall minimum capacity investment that needs to be installed in the last period of the optimization (taking into account decommissionings); only applicable for multi-period models.
- **lifetime** (int, \( l \)) – Units lifetime, given in years; only applicable for multi-period models.
- **age** (int, \( a \)) – Units start age, given in years at the beginning of the simulation; only applicable for multi-period models.
- **interest_rate** (float, \( i_r \)) – Interest rate for calculating annuities when investing in a particular unit; only applicable for multi-period models. If nothing else is specified, the interest rate is the same as the model discount rate of the multi-period model.
- **fixed_costs** (float or list of float, \( c_{\text{fixed}}(p) \)) – Fixed costs in each period (given in nominal terms); only applicable for multi-period models.
Implementation – oemof.solph.flows._invest_flow

```python
self.invest = Var(
    self.INVESTFLOWS,
    m.PERIODS,
    within=NonNegativeReals,
    bounds=_investvar_bound_rule,
)

# Total capacity
self.total = Var(self.INVESTFLOWS, m.PERIODS, within=NonNegativeReals)

if m.es.multi_period:
    self.old = Var(
        self.INVESTFLOWS, m.PERIODS, within=NonNegativeReals
    )

    # Old endogenous capacity to be decommissioned (due to lifetime)
    self.old_end = Var(
        self.INVESTFLOWS, m.PERIODS, within=NonNegativeReals
    )

    # Old exogenous capacity to be decommissioned (due to lifetime)
    self.old_exo = Var(
        self.INVESTFLOWS, m.PERIODS, within=NonNegativeReals
    )
```
multi-period modeling in oemof.solph

Implementation – oemof.solph.flows._invest_flow

```python
for i, o in self.CONVEX_INVESTFLOWS:
    lifetime = m.flows[i, o].investment.lifetime
    interest = m.flows[i, o].investment.interest_rate
    if interest == 0:
        warn(
            msg.format(m.discount_rate),
            debugging.SuspiciousUsageWarning,
        )
    interest = m.discount_rate
    for p in m.PERIODS:
        annuity = economics.annuity(
            capex=m.flows[i, o].investment.ep_costs[p],
            n=lifetime,
            wacc=interest,
        )
        investment_costs_increment = (
            self.invest[i, o, p]
            * annuity
            * lifetime
            * ((1 + m.discount_rate) ** (-m.es.periods_years[p]))
        )
        investment_costs += investment_costs_increment
        period_investment_costs[p] += investment_costs_increment
```
Lifetime logic

- $P$: installed capacity; $p$: period; $n$: lifetime

$$P_{\text{total}}(p) = P_{\text{invest}}(p) + P_{\text{total}}(p-1) - P_{\text{old}}(p) \quad \forall p > 0$$

$$P_{\text{total}}(0) = P_{\text{invest}}(0) + P_{\text{existing}} \quad \forall p = 0$$

$$P_{\text{old, end}}(p) = P_{\text{invest}}(p-n) \quad \forall p \geq n$$

$$P_{\text{old, end}}(p) = 0 \quad \text{else}$$

$$P_{\text{old, exo}}(p) = P_{\text{existing}} \quad \forall p = n - \text{age}$$

$$P_{\text{old, exo}}(p) = 0 \quad \text{else}$$

$$P_{\text{old}}(p) = P_{\text{old, end}}(p) + P_{\text{old, exo}}(p)$$

Total (installed) cap: previous cap + installations - decommissionings

Decommissionings
- endogeneous plants: installations that happened in the period the plants lifetime ago
- exogeneous plants: decommissioning of existing capacity in period lifetime – (initial) age
- Total decommissioning: sum of endogeneous and exogeneous decommissioning
Handling cost values (1/2)

- In general: all cost values may vary on a **periodical basis**, but shall be fixed within a period.

- Cost values have to be provided in **nominal terms**.
  - Calculating real values and annuities takes place under the hood.

**Annuities and discounting**

- A **discount_rate** is given on a **model-wide basis**. It reflects inflation.
- An **interest rate may be given per component / flow** (asset) that can be invested in. It can deviate from the discount_rate, e.g. to take an investor’s view and demand for higher interest rates.
  
  If a social planner perspective is taken, the interest_rate should be equal to the model’s discount_rate, which is the default.
- Annuities are calculated under the hood (next slide).
Handling cost values (2/2)

• Cost terms for MultiPeriodInvestment objects (or other components that is invested in)

**CAPEX: investment annuities**

\[ P_{\text{invest}}(p) \cdot \text{annuity}(c_{\text{invest}}(p), n, i) \cdot n \cdot DF(p) \quad \forall p \in \text{PERIODS} \]

\[
\text{annuity}(c_{\text{invest}}(p), n, i) = \frac{(1 + i)^n \cdot i}{(1 + i)^n - 1} \cdot c_{\text{invest}}(p)
\]

**Fixed costs**

\[
\sum_{p=1}^{p+n} P_{\text{invest}}(p) \cdot c_{\text{fixed}}(pp) \cdot DF(pp) \cdot DF(p) \quad \forall p \in \text{PERIODS}
\]

with discount factor

\[ DF(p) = (1 + d)^{-p} \]

• \( P \): installed capacity
• \( p \): period
• \( n \): lifetime
• \( i \): interest rate
• \( DF \): discount factor
Outlook

- Pending PR: [https://github.com/oemof/oemof-solph/pull/810](https://github.com/oemof/oemof-solph/pull/810)
  - Functional and complete
  - Usable via pip install git+https://github.com/oemof/oemof-solph.git@features/multi-period
  - Currently some merge conflicts due to works on v0.5.0
  - Can't / won't be resolved until / unless there is a clear timeline for integration
Contact

Johannes Kochems

research associate at DLR, Institute of Networked Energy Systems, Stuttgart | PhD candidate at Technical University of Berlin

E-Mail: johannes.kochems@dlr.de kochems@campus.tu-berlin.de

GitHub: jokochems