Implementation of a rolling horizon / myopic optimization approach in the ER fundamental power market model

Johannes Kochems & Yannick Werner | Department of Energy and Resource Management at TU Berlin | 5 Decembre 2019
Motivation and background

Motivation

- General modelling approach: Using a **power market model** for
  - investment resp.
  - dispatch optimization
  for Germany implemented using oemof

- Need for measures for complexity reduction to fasten up solution times
  - dispatch model: Rolling horizon
  - investment model: Myopic optimization implemented as an option which can be chosen.

Basic concept

- Basic approach identical
  - dispatch model: shorter timeslices
    → a couple of days / weeks
  - investment model: longer timeslices
    → (multiple) entire years with limited foresight* and not necessarily an overlap

*drawback: no longer term prognosis on future earnings
Integrated (yet) which would influence investment decisions
Implementation: dispatch example (1) – initialization

1. Initialize by setting / calculating necessary parameters

```
timeslice_length_wo_overlap_in_hours = 168
overlap_in_hours = 48

# starttime and endtime must be manually set here
timeseries_start = pd.Timestamp(starttime, freq)
timeseries_end = pd.Timestamp(endtime, freq)

# Calculate timeslice length as well as overlap length
timeslice_length_wo_overlap_in_timesteps = {'60min': 1, '15min': 4}[freq] * timeslice_length_wo_overlap_in_hours
overlap_in_timesteps = {'60min': 1, '15min': 4}[freq] * overlap_in_hours
timeslice_length_with_overlap = timeslice_length_wo_overlap_in_timesteps + overlap_in_timesteps

# Calculate amount of timeslices needed
overall_timesteps = timessteps_between_timestamps(timeseries_start, timeseries_end, freq)
amount_of_timeslices = math.ceil(overall_timesteps / timeslice_length_wo_overlap_in_timesteps)
```

*min uptime / min downtime constraints may be violated in our current implementation state when crossing timeslices.
Implementation: dispatch example (1) – initialization

1. Initialize by setting / calculating necessary parameters

   ```python
   timeslice_length_wo_overlap_in_hours = 168
   overlap_in_hours = 48
   
   # starttime and endtime must be manually set here
   timeseries_start = pd.Timestamp(starttime, freq)
   timeseries_end = pd.Timestamp(endtime, freq)
   
   # Calculate timeslice length as well as overlap length
   timeslice_length_wo_overlap_in_timesteps = {'60min': 1, '15min': 4}[freq] * timeslice_length_wo_overlap_in_hours
   overlap_in_timesteps = {'60min': 1, '15min': 4}[freq] * overlap_in_hours
   timeslice_length_with_overlap = timeslice_length_wo_overlap_in_timesteps + overlap_in_timesteps
   
   # Calculate amount of timeslices needed
   overall_timesteps = timeseries_between_timestamps(timeseries_start, timeseries_end, freq)
   amount_of_timeslices = math.ceil(overall_timesteps / timeslice_length_with_overlap_in_timesteps)
   
   logging.info('Creating a LP optimization model for dispatch optimization 
   | 'using a ROLLING HORIZON approach for model solution.\n')
   
   # Initialization of RH model run
   counter = 0
   
   storages_init_df = pd.DataFrame()
   ...
   results = pd.DataFrame()
   power_prices = pd.DataFrame()
   ``

Empty DataFrames used to store initial states:
- LP dispatch model: storage level only (1)
- MILP dispatch model: (1) + transformer states*
- LP invest model: (1) + all investment variables

*min uptime / min downtime constraints may be violated in our current implementation state when crossing timeslices.
Implementation: dispatch example (2) – model run; basically a for-loop

2. Iteratively build-up and solve model using a (simple) for-loop

```python
# For loop controls rolling horizon model run
for counter in range(amount_of_timeslices):

    # (re)build optimization model in every iteration
    # Return the model, the energy system as well as the storage information
    om, es, timeseries_start, storage_labels, datetime_index \
    = build_RH_model(path_folder_input, filename_node_data, filename_cost_data, \
                      AggregateInput, Europe, \
                      fuel_cost_pathway, timeseries_start, \
                      timeslice_length_wo_overlap_in_timesteps, \
                      timeslice_length_with_overlap, \
                      counter,
                      storages_init_df, \
                      freq, year)
```

Build model using initial states
- Slice timeseries:
  - Use full length (incl. overlap) for reading in data
  - Set starting point of next iteration (excl. overlap)
- Set initial states:
  - Obtained from input sheets for first iteration
  - Stored in Dataframe elsewhere
Implementation: dispatch example (2) – model run; basically a for-loop

2. Iteratively build-up and solve model using a (simple) for-loop

```python
# For loop controls rolling horizon model run
for counter in range(amount_of_timeslices):
    # (re)build optimization model in every iteration
    # Return the model, the energy system as well as the storage information
    om, es, timeseries_start, storage_labels, datetime_index \n    = build_RH_model(path_folder_input, filename_node_data, filename_cost_data,
                      AggregateInput, Europe,
                      fuel_cost_pathway, timeseries_start,
                      timeslice_length_wo_overlap_in_timessteps,
                      timeslice_length_with_overlap,
                      counter,
                      storages_init_df,
                      freq, year)

# 14.05.2019, JK: Solve model and return results
om, model_results, results, overall_objective, overall_solution_time, \npower_prices = solve_RH_model(om, datetime_index, counter,
                             timeslice_length_wo_overlap_in_timessteps,
                             results,
                             power_prices,
                             overall_objective,
                             overall_solution_time,
                             solver = solver)
```

Build model using initial states
- Slice timeseries:
  - Use full length (incl. overlap) for reading in data
  - Set starting point of next iteration (excl. overlap)
- Set initial states:
  - Obtained from input sheets for first iteration
  - Stored in Dataframe elsewhise

Solve model
- optimize
- concat results (excl. overlap)
Implementation: dispatch example (2) – model run; basically a for-loop

2. Iteratively build-up and solve model using a (simple) for-loop

```python
# For loop controls rolling horizon model run
for counter in range(amount_of_timeslices):
    # (re)build optimization model in every iteration
    # Return the model, the energy system as well as the storage information
    om, es, timeseries_start, storage_labels, datetime_index =
        build_RH_model(path_folder_input, filename_node_data, filename_cost_data,
                        AggregateInput, Europe,
                        fuel_cost_pathway, timeseries_start,
                        timeslice_length_wo_overlap_in_timeseps,
                        timeslice_length_with_overlap,
                        counter,
                        storages_init_df, freq, year)

# 14.05.2019, JK: Solve model and return results
om, model_results, results, overall_objective, overall_solution_time, \
power_prices = solve_RH_model(om, datetime_index, counter,
                             timeslice_length_wo_overlap_in_timeseps,
                             results,
                             power_prices,
                             overall_objective,
                             overall_solution_time,
                             solver = solver)

# Set initial states for the next model run
# initial status for the first iteration is obtained from input data
storages_init_df = initial_states_RH(model_results, timeslice_length_wo_overlap_in_timeseps,
                                      storage_labels)
```

**Build model using initial states**
- Slice timeseries:
  - Use full length (incl. overlap) for reading in data
  - Set starting point of next iteration (excl. overlap)
- Set initial states:
  - Obtained from input sheets for first iteration
  - Stored in Dataframe elsewhise

**Solve model**
- optimize
- concat results (excl. overlap)

**Obtain initial states**
- get initial states (for next iteration)
Implementation: investment example – An overview on our myopic approach

- What differs compared to our dispatch implementation?
  
  - timeslices must be (multiple) entire years
  
  - a distinction between non leap years and leap years is included
    → leads to varying timeslice lengths
  
  - For this purpose, we introduced a `timeslice_length_dict`:

    ```python
    timeslice_length_dict = {i: (amount_of_years[i], timeslice_length_excl_overlap[i], timeslice_length_incl_overlap[i])}
    ```

  - …Most importantly: We call it myopic optimization, not rolling horizon optimization anymore. ;-)

- implementation of a rolling horizon / myopic optimization approach
Critical discussion and outlook

### Critical discussion

- **Drawbacks for our approach**
  - General: We lose a global optimum → decide on the basis of the modeling task and the hardware available
  - High computational overhead:
    - necessary data is read in „in chunks“
    - energy system is build up in every iteration
  - Lacking elements
    - Dumps are not properly included (yet)
    - Interemporal linking constraints are missing

- **Advantages for our approach**
  - Functional structure enables reusability
  - Computational advantage for MILP

### Outlook

- **What we will (probably) do**
  - Check whether we need this or whether other solutions can be found, like time series aggregation… or computing power
  - Reduce overhead
    - We prefer reading in the dataset upfront
    - Is there a workaround (planned) for not having to build the model everytime???
  - Close the gaps
    - We will add proper dumps and unfold these at the end
    - Introduction of linking constraints dependent on whether we will use MILP / need a rolling horizon approach at all…

- **Have a look on whether parts of the formulation can be generalized and made available in a small example**
Literature
