investment_optimization

May 12, 2017

1 Capabilities of the investment object

1.1 Different kinds of investment optimization - 4 variations

1.2 Import tools and time series data

```
In [1]: # Outputlib
        from oemof import outputlib

        # Default logger of oemof
        from oemof.tools import logger
        from oemof.tools import helpers

        # import oemof base classes to create energy system objects
        import logging
        import os
        import pandas as pd
        import matplotlib.pyplot as plt
        from oemof.tools import economics
        import oemof.solph as solph

        # read time series data
        filename = 'storage_invest.csv'
        solvername='cbc'
        debug=True
        number_timesteps=8760
        tee_switch=True

        logging.info('Initialize the energy system')
        date_time_index = pd.date_range('1/1/2012', periods=number_timesteps,
                                        freq='H')

        energysystem = solph.EnergySystem(timeindex=date_time_index)

        logging.info('Read data file')
        full_filename = os.path.join('../../User-Shares/Git/oemof/examples/solph/storage_optimiz
        data = pd.read_csv(full_filename, sep=",")
```

```
consumption_total = 2255*1e6
```

1.3 1) Set cost parameter - invest optimize all components

In [2]:
```
# epc: equivalent periodical cost
# If the period is one year the equivalent periodical costs (epc) of an investment are e

epc_wind = economics.annuity(capex=1000, n=20, wacc=0.05)

epc_pv = economics.annuity(capex=1000, n=20, wacc=0.05)

epc_storage = economics.annuity(capex=1000, n=20, wacc=0.05)

price_gas = 0.04
```

1.4 1) Create oemof objects - invest optimize all components

1.4.1 Wind, PV and Storage get an investment attribute each; the gas resource gets variable costs.

In [3]:
```
# create natural gas bus
bgas = solph.Bus(label="natural_gas")

# create electricity bus
bel = solph.Bus(label="electricity")

# create excess component for the electricity bus to allow overproduction
solph.Sink(label='excess_bel', inputs={bel: solph.Flow()})

# create simple sink object representing the electrical demand
solph.Sink(label='demand', inputs={bel: solph.Flow(
    actual_value=data['demand_el'], fixed=True, nominal_value=1)})

# create fixed source object representing wind power plants
solph.Source(label='wind', outputs={bel: solph.Flow(
    actual_value=data['wind'],
    fixed=True,
    fixed_costs=20,
    investment=solph.Investment(ep_costs=epc_wind))})

# create fixed source object representing pv power plants
solph.Source(label='pv', outputs={bel: solph.Flow(
    actual_value=data['pv'],
    fixed=True,
    fixed_costs=15,
    investment=solph.Investment(ep_costs=epc_pv))})
```

```
        # create source object representing the natural gas commodity (annual limit)
        solph.Source(label='rgas', outputs={bgas: solph.Flow(
            variable_costs=price_gas)})

        # create simple transformer object representing a gas power plant
        solph.LinearTransformer(
            label="pp_gas",
            inputs={bgas: solph.Flow()},
            outputs={bel: solph.Flow(nominal_value=10e10, variable_costs=0)},
            conversion_factors={bel: 0.58})

        # create storage object representing a battery
        solph.Storage(
            label='storage',
            inputs={bel: solph.Flow(variable_costs=1)},
            outputs={bel: solph.Flow(variable_costs=1)},
            capacity_loss=0.00, initial_capacity=0,
            nominal_input_capacity_ratio=1/6,
            nominal_output_capacity_ratio=1/6,
            inflow_conversion_factor=1, outflow_conversion_factor=0.8,
            fixed_costs=35,
            investment=solph.Investment(ep_costs=epc_storage),
        )
```

```
Out[3]: <oemof.solph.network.Storage at 0x7fcbb482c8b8>
```

1.5 1) Optimize the energy system - invest optimize all components

1.5.1 This takes around one minute...

```
In [4]: logging.info('Optimize the energy system')
        om = solph.OperationalModel(energysystem)

        if debug:
            filename = os.path.join(
                helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
            logging.info('Store lp-file in {0}.'.format(filename))
            om.write(filename, io_options={'symbolic_solver_labels': True})

        logging.info('Solve the optimization problem')
        om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
```

```
Welcome to the CBC MILP Solver
Version: 2.8.12
Build Date: Sep  8 2014

command line - /usr/bin/cbc -printingOptions all -import /tmp/tmpe1cuiy8y.pyomo.lp -import -stat
Option for printingOptions changed from normal to all
Current default (if $ as parameter) for import is /tmp/tmpe1cuiy8y.pyomo.lp
```

```
Presolve 43795 (-52569) rows, 35038 (-43808) columns and 126685 (-113902) elements
Statistics for presolved model


Problem has 43795 rows, 35038 columns (26281 with objective) and 126685 elements
There are 8760 singletons with objective
Column breakdown:
26278 of type 0.0->inf, 8760 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
8758 of type E 0.0, 0 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
8760 of type G other, 26277 of type L 0.0, 0 of type L 1.0,
0 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 43795 (-52569) rows, 35038 (-43808) columns and 126685 (-113902) elements
Perturbing problem by 0.001 %% of 1297.3921 - largest nonzero change 9.9999582e-05 (%% 0.0751187
0  Obj 0 Primal inf 1.4829433e+10 (8760)
443  Obj 8127740.4 Primal inf 1.1303449e+10 (8317)
886  Obj 15963290 Primal inf 9.2675644e+09 (7874)
1329  Obj 23819455 Primal inf 7.7821313e+09 (7431)
1772  Obj 31617449 Primal inf 6.6163868e+09 (6988)
2215  Obj 39314103 Primal inf 5.6618609e+09 (6545)
2658  Obj 47309308 Primal inf 4.8600585e+09 (6102)
3101  Obj 55215174 Primal inf 4.1873339e+09 (5659)
3544  Obj 63112427 Primal inf 3.6166405e+09 (5216)
3987  Obj 71121208 Primal inf 3.118613e+09 (4773)
4430  Obj 78972236 Primal inf 2.6787394e+09 (4330)
4873  Obj 86827859 Primal inf 2.2870851e+09 (3887)
5316  Obj 94904247 Primal inf 1.9300632e+09 (3444)
5759  Obj 1.0290737e+08 Primal inf 1.6020429e+09 (3001)
6202  Obj 1.1107629e+08 Primal inf 1.3002798e+09 (2558)
6645  Obj 1.17588e+08 Primal inf 2.0012478e+08 (1602)
7039  Obj 1.2019553e+08 Primal inf 70179515 (928)
7351  Obj 1.2094914e+08 Primal inf 12731989 (350)
7587  Obj 1.211205e+08
Optimal - objective value 1.2111743e+08
After Postsolve, objective 1.2111743e+08, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 121117432.2 - 7587 iterations time 3.252, Presolve 2.22
Total time (CPU seconds):       5.45   (Wallclock seconds):       5.60
```

```
Out[4]: {'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed',
```

1.6 1) Check the results - invest optimize all components

1.6.1 Only wind power is installed with almost 400 MW. The rest of the energy supply comes from the gas resource. A renewable energy share of nearly 50% is achieved.

```
In [5]: storage = energysystem.groups['storage']
        wind_inst = energysystem.groups['wind']
        pv_inst = energysystem.groups['pv']
        myresults = outputlib.DataFramePlot(energy_system=energysystem)

        # electrical output of natural gas power plant
        pp_gas = myresults.slice_by(obj_label='pp_gas', type='to_bus',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical demand
        demand = myresults.slice_by(obj_label='demand',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical output of wind power plant
        wind = myresults.slice_by(obj_label='wind',
                                  date_from='2012-01-01 00:00:00',
                                  date_to='2012-12-31 23:00:00')

        # electrical output of pv power plant
        pv = myresults.slice_by(obj_label='pv',
                                date_from='2012-01-01 00:00:00',
                                date_to='2012-12-31 23:00:00')

        import pprint as pp
        pp.pprint({
                   'wind_inst_MW': energysystem.results[wind_inst][bel].invest/1000,
                   'pv_inst_MW': energysystem.results[pv_inst][bel].invest/1000,
                   'storage_cap_GWh': energysystem.results[storage][storage].invest/1e6,
                   'res_share': 1 - pp_gas.sum()/demand.sum(),
                'objective': energysystem.results.objective})
```

```
{'objective': 121117432.50284117,
 'pv_inst_MW': 0.0,
 'res_share': val    0.474236
dtype: float64,
 'storage_cap_GWh': 0.0,
 'wind_inst_MW': 392.56882}
```

1.7 1) Plot one week - invest optimize all components

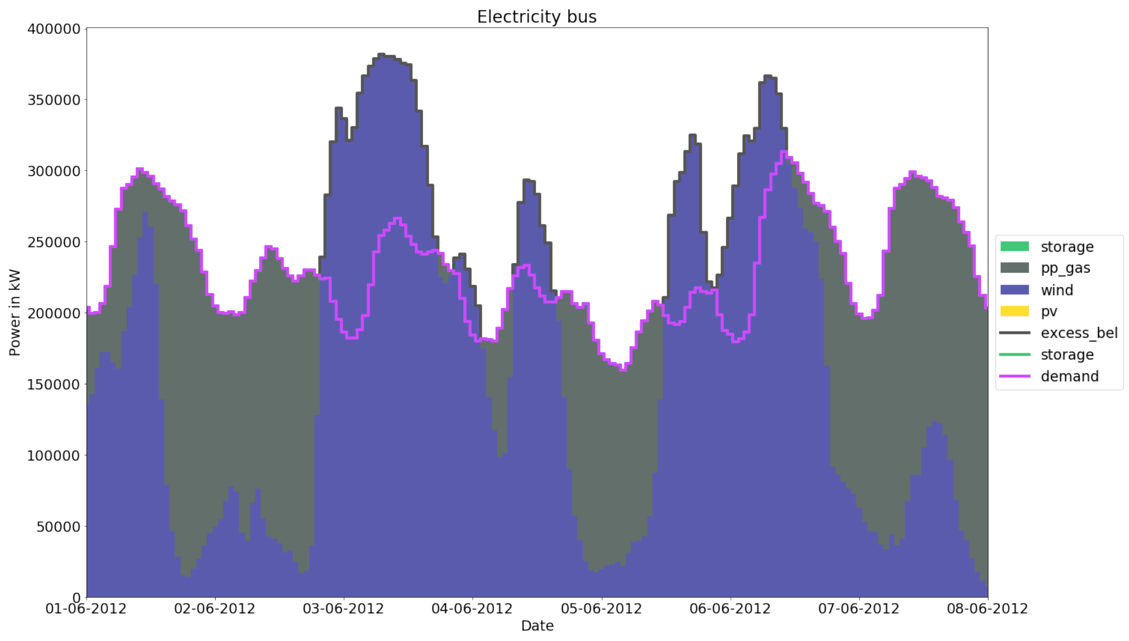
```
In [5]: cdict = {'wind': '#5b5bae',
                 'pv': '#ffde32',
                 'storage': '#42c77a',
                 'pp_gas': '#636f6b',
                 'demand': '#ce4aff',
                 'excess_bel': '#555555'}

        myplot = outputlib.DataFramePlot(energy_system=energysystem)

        # Plotting the balance around the electricity plot for one week using a
        # combined stacked plot
        fig = plt.figure(figsize=(24, 14))
        plt.rc('legend', **{'fontsize': 19})
        plt.rcParams.update({'font.size': 19})
        plt.style.use('grayscale')

        handles, labels = myplot.io_plot(
            bus_label='electricity', cdict=cdict,
            barorder=['pv', 'wind', 'pp_gas', 'storage'],
            lineorder=['demand', 'storage', 'excess_bel'],
            line_kwa={'linewidth': 4},
            ax=fig.add_subplot(1, 1, 1),
            date_from="2012-06-01 00:00:00",
            date_to="2012-06-08 00:00:00",
            )
        myplot.ax.set_ylabel('Power in kW')
        myplot.ax.set_xlabel('Date')
        myplot.ax.set_title("Electricity bus")
        myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
        myplot.outside_legend(handles=handles, labels=labels)

        plt.show()
```



1.8 2) Set cost parameter - fix PV and wind installed capacity --> optimize storage capacity vs. gas resource

```
In [2]: # epc: equivalent periodical cost
        # If the period is one year the equivalent periodical costs (epc) of an investment are e

        epc_storage = economics.annuity(capex=1000, n=20, wacc=0.05)

        price_gas = 0.04
```

1.9 2) Create oemof objects - fix PV and wind installed capacity --> optimize storage capacity vs. gas resource

1.9.1 Wind and PV nominal capacities are set to 1000 MW and 600 MW.

```
In [3]: # create natural gas bus
        bgas = solph.Bus(label="natural_gas")

        # create electricity bus
        bel = solph.Bus(label="electricity")

        # create excess component for the electricity bus to allow overproduction
        solph.Sink(label='excess_bel', inputs={bel: solph.Flow()})

        # create simple sink object representing the electrical demand
        solph.Sink(label='demand', inputs={bel: solph.Flow(
            actual_value=data['demand_el'], fixed=True, nominal_value=1)})
```

```
# create fixed source object representing wind power plants
solph.Source(label='wind', outputs={bel: solph.Flow(
    actual_value=data['wind'],
    fixed=True,
    fixed_costs=20,
    nominal_value=1000000)})

# create fixed source object representing pv power plants
solph.Source(label='pv', outputs={bel: solph.Flow(
    actual_value=data['pv'],
    fixed=True,
    fixed_costs=15,
    nominal_value=600000)})

# create source object representing the natural gas commodity (annual limit)
solph.Source(label='rgas', outputs={bgas: solph.Flow(
    variable_costs=price_gas)})

# create simple transformer object representing a gas power plant
solph.LinearTransformer(
    label="pp_gas",
    inputs={bgas: solph.Flow()},
    outputs={bel: solph.Flow(nominal_value=10e10, variable_costs=0)},
    conversion_factors={bel: 0.58})

# create storage object representing a battery
solph.Storage(
    label='storage',
    inputs={bel: solph.Flow(variable_costs=1)},
    outputs={bel: solph.Flow(variable_costs=1)},
    capacity_loss=0.00, initial_capacity=0,
    nominal_input_capacity_ratio=1/6,
    nominal_output_capacity_ratio=1/6,
    inflow_conversion_factor=1, outflow_conversion_factor=0.8,
    fixed_costs=35,
    investment=solph.Investment(ep_costs=epc_storage),
)
```

```
Out[3]: <oemof.solph.network.Storage at 0x7f23afe01900>
```

1.10 2) Optimize the energy system - fix PV and wind installed capacity --> optimize storage capacity vs. gas resource

1.10.1 This takes around one minute...

```
In [4]: logging.info('Optimize the energy system')
        om = solph.OperationalModel(energysystem)
        if debug:
            filename = os.path.join(
                helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
            logging.info('Store lp-file in {0}.'.format(filename))
            om.write(filename, io_options={'symbolic_solver_labels': True})

        logging.info('Solve the optimization problem')
        om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
```

```
Welcome to the CBC MILP Solver
Version: 2.8.12
Build Date: Sep  8 2014

command line - /usr/bin/cbc -printingOptions all -import /tmp/tmpbo84bkgv.pyomo.lp -import -stat
Option for printingOptions changed from normal to all
Current default (if $ as parameter) for import is /tmp/tmpbo84bkgv.pyomo.lp
Presolve 43794 (-17530) rows, 35035 (-26289) columns and 113862 (-43824) elements
Statistics for presolved model


Problem has 43794 rows, 35035 columns (26278 with objective) and 113862 elements
There are 8759 singletons with objective
Column breakdown:
26276 of type 0.0->inf, 8759 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
8758 of type E 0.0, 0 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
8759 of type G other, 26277 of type L 0.0, 0 of type L 1.0,
0 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 43794 (-17530) rows, 35035 (-26289) columns and 113862 (-43824) elements
Perturbing problem by 0.001 %% of 118.50229 - largest nonzero change 9.9997854e-05 (%% 0.0251872
0  Obj 29007009 Primal inf 7.4482875e+08 (3544)
443  Obj 37090079 Primal inf 5.7206156e+08 (3101)
886  Obj 43560951 Primal inf 4.3375352e+08 (2658)
1329  Obj 49214105 Primal inf 3.1292333e+08 (2215)
1772  Obj 54127033 Primal inf 2.0791467e+08 (1772)
2215  Obj 58187996 Primal inf 1.2111587e+08 (1329)
2658  Obj 61215401 Primal inf 56408260 (886)
3101  Obj 63161598 Primal inf 14810353 (443)
3544  Obj 63854518
Optimal - objective value 63853802
After Postsolve, objective 63853802, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 63853801.71 - 3544 iterations time 0.562, Presolve 0.39
Total time (CPU seconds):       1.50   (Wallclock seconds):       1.55
```

```
Out[4]: {'Solver': [{'Error rc': 0, 'User time': -1.0, 'Status': 'ok', 'Termination condition':
```

1.11 2) Check the results - fix PV and wind installed capacity --> optimize storage capacity vs. gas resource

1.11.1 Now the renewable share is higher because of the fixed installed wind and PV capacities. Storage is not installed; gas is cheaper in this case.

```
In [5]: storage = energysystem.groups['storage']
        wind_inst = energysystem.groups['wind']
        pv_inst = energysystem.groups['pv']
        myresults = outputlib.DataFramePlot(energy_system=energysystem)

        # electrical output of natural gas power plant
        pp_gas = myresults.slice_by(obj_label='pp_gas', type='to_bus',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical demand
        demand = myresults.slice_by(obj_label='demand',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical output of wind power plant
        wind = myresults.slice_by(obj_label='wind',
                                  date_from='2012-01-01 00:00:00',
                                  date_to='2012-12-31 23:00:00')

        # electrical output of pv power plant
        pv = myresults.slice_by(obj_label='pv',
                                date_from='2012-01-01 00:00:00',
                                date_to='2012-12-31 23:00:00')

        import pprint as pp
        pp.pprint({
                   'storage_cap_GWh': energysystem.results[storage][storage].invest/1e6,
                   'res_share': 1 - pp_gas.sum()/demand.sum(),
                'objective': energysystem.results.objective})
```

```
{'objective': 63853801.70328918,
 'res_share': val    0.775885
dtype: float64,
 'storage_cap_GWh': 0.0}
```

1.12 2) Plot one week - fix PV and wind installed capacity --> optimize storage capacity vs. gas resource

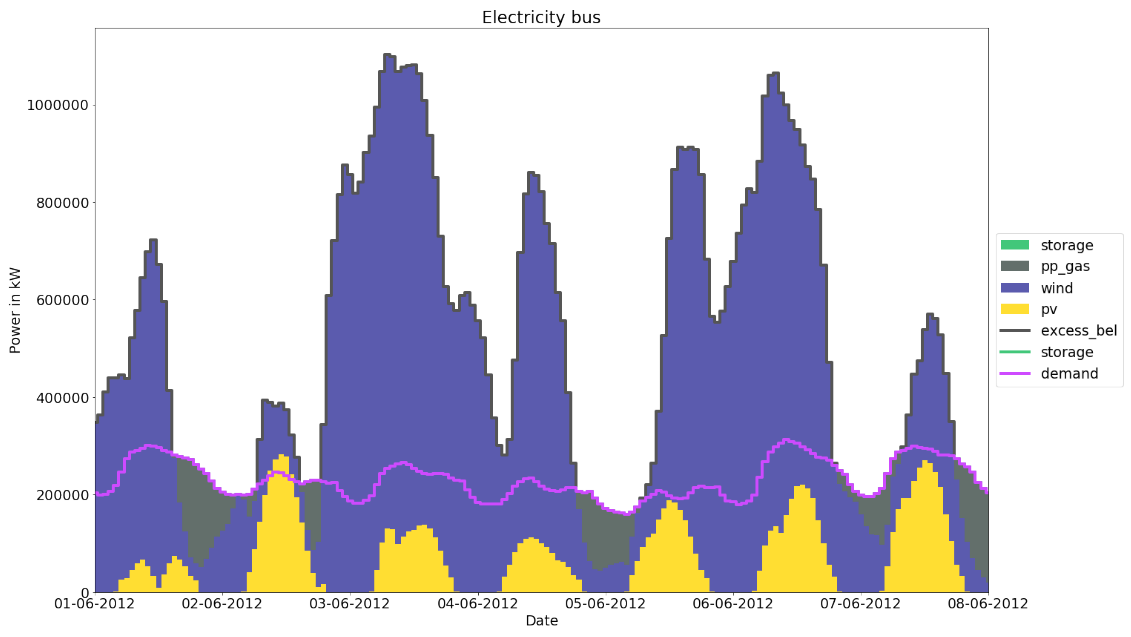
```
In [6]: cdict = {'wind': '#5b5bae',
                 'pv': '#ffde32',
                 'storage': '#42c77a',
                 'pp_gas': '#636f6b',
                 'demand': '#ce4aff',
                 'excess_bel': '#555555'}

        myplot = outputlib.DataFramePlot(energy_system=energysystem)

        # Plotting the balance around the electricity plot for one week using a
        # combined stacked plot
        fig = plt.figure(figsize=(24, 14))
        plt.rc('legend', **{'fontsize': 19})
        plt.rcParams.update({'font.size': 19})
        plt.style.use('grayscale')

        handles, labels = myplot.io_plot(
            bus_label='electricity', cdict=cdict,
            barorder=['pv', 'wind', 'pp_gas', 'storage'],
            lineorder=['demand', 'storage', 'excess_bel'],
            line_kwa={'linewidth': 4},
            ax=fig.add_subplot(1, 1, 1),
            date_from="2012-06-01 00:00:00",
            date_to="2012-06-08 00:00:00",
            )
        myplot.ax.set_ylabel('Power in kW')
        myplot.ax.set_xlabel('Date')
        myplot.ax.set_title("Electricity bus")
        myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
        myplot.outside_legend(handles=handles, labels=labels)

        plt.show()
```



1.13 3) Set cost parameter - additionally set a fossil share (special case: calculation of required storage capacity)

```
In [2]: epc_storage = economics.annuity(capex=1000, n=20, wacc=0.05)

        fossil_share = 0.2
```

1.14 3) Create oemof objects - additionally set a fossil share (special case: calculation of required storage capacity)

1.14.1 The variable cost is removed from the gas resource. Therefore a fossil share is set.

```
In [3]: # create natural gas bus
        bgas = solph.Bus(label="natural_gas")

        # create electricity bus
        bel = solph.Bus(label="electricity")

        # create excess component for the electricity bus to allow overproduction
        solph.Sink(label='excess_bel', inputs={bel: solph.Flow()})

        # create simple sink object representing the electrical demand
        solph.Sink(label='demand', inputs={bel: solph.Flow(
            actual_value=data['demand_el'], fixed=True, nominal_value=1)})

        # create fixed source object representing wind power plants
        solph.Source(label='wind', outputs={bel: solph.Flow(
    actual_value=data['wind'],
    fixed=True,
    fixed_costs=20,
    nominal_value=1000000)})

# create fixed source object representing pv power plants
solph.Source(label='pv', outputs={bel: solph.Flow(
    actual_value=data['pv'],
    fixed=True,
    fixed_costs=15,
    nominal_value=600000)})

# create source object representing the natural gas commodity (annual limit)
solph.Source(label='rgas', outputs={bgas: solph.Flow(
    nominal_value=fossil_share * consumption_total/0.58 * number_timesteps / 8760, summe

# create simple transformer object representing a gas power plant
solph.LinearTransformer(
    label="pp_gas",
    inputs={bgas: solph.Flow()},
    outputs={bel: solph.Flow(nominal_value=10e10, variable_costs=0)},
    conversion_factors={bel: 0.58})

# create storage object representing a battery
solph.Storage(
    label='storage',
    inputs={bel: solph.Flow(variable_costs=1)},
    outputs={bel: solph.Flow(variable_costs=1)},
    capacity_loss=0.00, initial_capacity=0,
    nominal_input_capacity_ratio=1/6,
    nominal_output_capacity_ratio=1/6,
    inflow_conversion_factor=1, outflow_conversion_factor=0.8,
    fixed_costs=35,
    investment=solph.Investment(ep_costs=epc_storage),
)
```

```
Out[3]: <oemof.solph.network.Storage at 0x7f0b8b9d7828>
```

1.15 3) Optimize the energy system - additionally set a fossil share (special case: calculation of required storage capacity)

1.15.1 This takes around one minute...

```
In [4]: logging.info('Optimize the energy system')
        om = solph.OperationalModel(energysystem)

        if debug:
            filename = os.path.join(
                helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
            logging.info('Store lp-file in {0}.'.format(filename))
            om.write(filename, io_options={'symbolic_solver_labels': True})

        logging.info('Solve the optimization problem')
        om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
```

```
Welcome to the CBC MILP Solver
Version: 2.8.12
Build Date: Sep  8 2014

command line - /usr/bin/cbc -printingOptions all -import /tmp/tmpp6zkd5km.pyomo.lp -import -stat
Option for printingOptions changed from normal to all
Current default (if $ as parameter) for import is /tmp/tmpp6zkd5km.pyomo.lp
Presolve 43794 (-17531) rows, 35034 (-26290) columns and 122619 (-43827) elements
Statistics for presolved model


Problem has 43794 rows, 35034 columns (17519 with objective) and 122619 elements
Column breakdown:
26276 of type 0.0->inf, 8758 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
8758 of type E 0.0, 0 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
8758 of type G other, 26277 of type L 0.0, 0 of type L 1.0,
1 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 43794 (-17531) rows, 35034 (-26290) columns and 122619 (-43827) elements
Perturbing problem by 0.001 %% of 118.50229 - largest nonzero change 9.9996095e-05 (%% 0.0251848
0  Obj 29000000 Primal inf 7.4482736e+08 (3543)
443  Obj 29008315 Primal inf 5.7206042e+08 (3100)
886  Obj 29015003 Primal inf 4.3375213e+08 (2657)
1329  Obj 29020829 Primal inf 3.1292194e+08 (2214)
1772  Obj 29025857 Primal inf 2.0791327e+08 (1771)
2215  Obj 29030036 Primal inf 1.2111447e+08 (1328)
2658  Obj 33749249 Primal inf 98712367 (1020)
3101  Obj 33840585 Primal inf 99205824 (1019)
3544  Obj 33931877 Primal inf 99577094 (1018)
3987  Obj 34296371 Primal inf 1.0075254e+08 (1018)
4430  Obj 34568941 Primal inf 1.0150877e+08 (1019)
4873  Obj 35022624 Primal inf 1.0288338e+08 (1022)
5316  Obj 51346989 Primal inf 1.1482192e+08 (3697)
5759  Obj 67915890 Primal inf 1.1460665e+08 (3681)
6202  Obj 76748449 Primal inf 1.199712e+08 (3692)
6645  Obj 81987091 Primal inf 1.26899e+08 (3677)
7088  Obj 84757864 Primal inf 1.5306656e+08 (3591)
7531  Obj 85762152 Primal inf 1.5545446e+08 (3530)
```

```
7974  Obj 87077954 Primal inf 1.6179947e+08 (3413)
8417  Obj 88155648 Primal inf 1.9167883e+08 (3348)
8860  Obj 89342970 Primal inf 1.9363795e+08 (3235)
9303  Obj 90558937 Primal inf 1.8579625e+08 (3110)
9746  Obj 92162740 Primal inf 1.7526327e+08 (2923)
10189  Obj 93563966 Primal inf 1.9465536e+08 (2868)
10632  Obj 95147239 Primal inf 2.0517448e+08 (2880)
11075  Obj 97247502 Primal inf 2.1905771e+08 (2745)
11518  Obj 99296819 Primal inf 2.3515744e+08 (2753)
11961  Obj 1.0226234e+08 Primal inf 2.5341051e+08 (2652)
12404  Obj 1.0713184e+08 Primal inf 3.0608995e+08 (2542)
12847  Obj 1.1250468e+08 Primal inf 3.5253572e+08 (2465)
13290  Obj 1.1955192e+08 Primal inf 4.0944611e+08 (2402)
13733  Obj 1.2803415e+08 Primal inf 4.6085013e+08 (2427)
14176  Obj 1.3297544e+08 Primal inf 7.3783445e+08 (1654)
14619  Obj 1.3299216e+08 Primal inf 9.5411971e+08 (2127)
15062  Obj 1.3302517e+08 Primal inf 1.2889097e+09 (2463)
15505  Obj 1.3549269e+08 Primal inf 1.8390031e+08 (2109)
15948  Obj 1.4322282e+08 Primal inf 1.8598162e+08 (1994)
16391  Obj 1.5830532e+08 Primal inf 2.0656152e+08 (1725)
16834  Obj 1.6218712e+08 Primal inf 2.114518e+08 (1870)
17277  Obj 1.6812712e+08 Primal inf 1.9309491e+08 (1794)
17720  Obj 1.7484512e+08 Primal inf 1.6409961e+08 (1759)
18163  Obj 1.801134e+08 Primal inf 1.8584777e+08 (1752)
18606  Obj 1.8572904e+08 Primal inf 1.4250599e+08 (1721)
19049  Obj 1.900892e+08 Primal inf 1.8145903e+08 (1761)
19492  Obj 1.9476139e+08 Primal inf 44025264 (1092)
19935  Obj 1.9719098e+08 Primal inf 4100352.9 (404)
20056  Obj 1.9726903e+08
Optimal - objective value 1.9719817e+08
After Postsolve, objective 1.9719817e+08, infeasibilities - dual 0 (0), primal 3.595488e-06 (1)
Presolved model was optimal, full model needs cleaning up
0  Obj 1.9719817e+08 Primal inf 3.595488e-06 (1) Dual inf 1e+08 (1)
End of values pass after 1 iterations
1  Obj 1.9719817e+08
Perturbing problem by 0.0001 %% of 0.1 - largest nonzero change 3.3716206e-07 (%% 3.3716206e-05)
1  Obj 1.9719817e+08
Optimal - objective value 1.9719817e+08
Optimal objective 197198173.9 - 20057 iterations time 29.002, Presolve 0.45
Total time (CPU seconds):       30.00   (Wallclock seconds):       30.65
```

```
Out[4]: {'Problem': [{'Upper bound': 197198170.0, 'Name': 'tmpp6zkd5km.pyomo', 'Number of variab
```

1.16 3) Check the results - additionally set a fossil share (special case: calculation of required storage capacity)

1.16.1 Through the fixed fossil share the renewable share is now at exactly 80%. To achieve this storage with around 400 MWh is required.

```
In [5]: storage = energysystem.groups['storage']
        wind_inst = energysystem.groups['wind']
        pv_inst = energysystem.groups['pv']
        myresults = outputlib.DataFramePlot(energy_system=energysystem)

        # electrical output of natural gas power plant
        pp_gas = myresults.slice_by(obj_label='pp_gas', type='to_bus',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical demand
        demand = myresults.slice_by(obj_label='demand',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical output of wind power plant
        wind = myresults.slice_by(obj_label='wind',
                                  date_from='2012-01-01 00:00:00',
                                  date_to='2012-12-31 23:00:00')

        # electrical output of pv power plant
        pv = myresults.slice_by(obj_label='pv',
                                date_from='2012-01-01 00:00:00',
                                date_to='2012-12-31 23:00:00')

        import pprint as pp
        pp.pprint({
                   'storage_cap_GWh': energysystem.results[storage][storage].invest/1e6,
                   'res_share': 1 - pp_gas.sum()/demand.sum(),
                'objective': energysystem.results.objective})
```

```
{'objective': 197198172.9032895,
 'res_share': val    0.8
dtype: float64,
 'storage_cap_GWh': 0.39779472}
```

1.17 3) Plot one week - additionally set a fossil share (special case: calculation of required storage capacity)

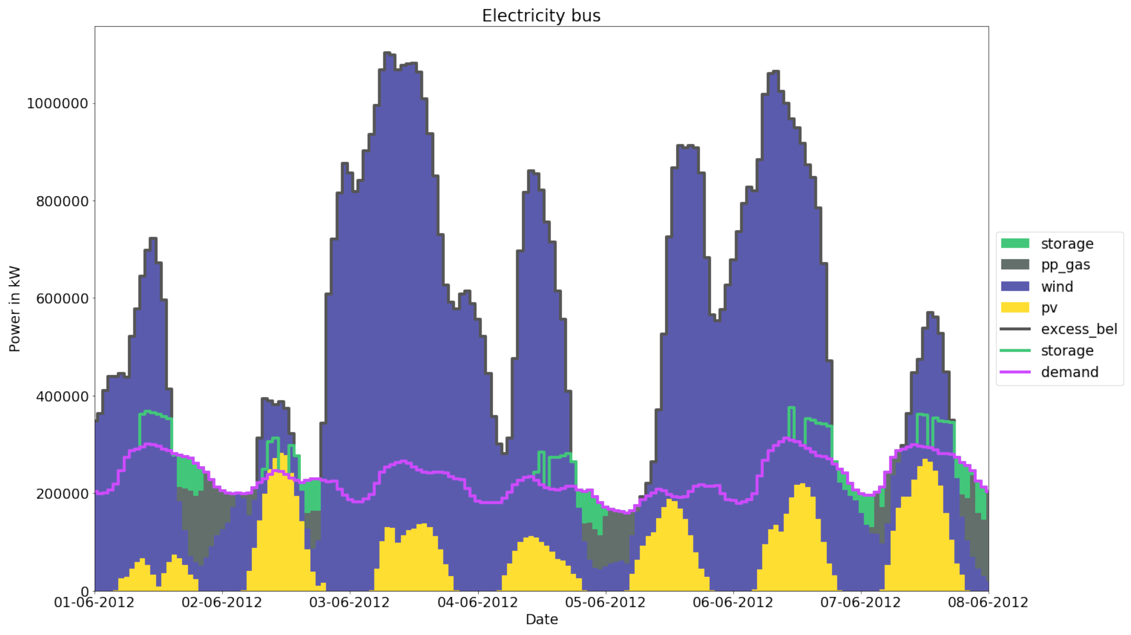
```
In [6]: cdict = {'wind': '#5b5bae',
                 'pv': '#ffde32',
                 'storage': '#42c77a',
         'pp_gas': '#636f6b',
         'demand': '#ce4aff',
         'excess_bel': '#555555'}

myplot = outputlib.DataFramePlot(energy_system=energysystem)

# Plotting the balance around the electricity plot for one week using a
# combined stacked plot
fig = plt.figure(figsize=(24, 14))
plt.rc('legend', **{'fontsize': 19})
plt.rcParams.update({'font.size': 19})
plt.style.use('grayscale')

handles, labels = myplot.io_plot(
    bus_label='electricity', cdict=cdict,
    barorder=['pv', 'wind', 'pp_gas', 'storage'],
    lineorder=['demand', 'storage', 'excess_bel'],
    line_kwa={'linewidth': 4},
    ax=fig.add_subplot(1, 1, 1),
    date_from="2012-06-01 00:00:00",
    date_to="2012-06-08 00:00:00",
    )
myplot.ax.set_ylabel('Power in kW')
myplot.ax.set_xlabel('Date')
myplot.ax.set_title("Electricity bus")
myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
myplot.outside_legend(handles=handles, labels=labels)

plt.show()
```



1.18 4) Set cost parameter - set fossil share but release PV and wind installed capacity

```
In [2]: epc_wind = economics.annuity(capex=1000, n=20, wacc=0.05)

        epc_pv = economics.annuity(capex=1000, n=20, wacc=0.05)

        epc_storage = economics.annuity(capex=1000, n=20, wacc=0.05)

        fossil_share = 0.2
```

1.19 4) Create oemof objects - set fossil share but release PV and wind installed capacity

1.19.1 The fix values of wind and PV installed capacity are removed. Therefore the investment attribute is set.

```
In [3]: # create natural gas bus
        bgas = solph.Bus(label="natural_gas")

        # create electricity bus
        bel = solph.Bus(label="electricity")

        # create excess component for the electricity bus to allow overproduction
        solph.Sink(label='excess_bel', inputs={bel: solph.Flow()})

        # create simple sink object representing the electrical demand
        solph.Sink(label='demand', inputs={bel: solph.Flow(
            actual_value=data['demand_el'], fixed=True, nominal_value=1)})

        # create fixed source object representing wind power plants
        solph.Source(label='wind', outputs={bel: solph.Flow(
            actual_value=data['wind'],
            fixed=True,
            fixed_costs=20,
            investment=solph.Investment(ep_costs=epc_wind))})

        # create fixed source object representing pv power plants
        solph.Source(label='pv', outputs={bel: solph.Flow(
            actual_value=data['pv'],
            fixed=True,
            fixed_costs=15,
            investment=solph.Investment(ep_costs=epc_pv))})

        # create source object representing the natural gas commodity (annual limit)
        solph.Source(label='rgas', outputs={bgas: solph.Flow(
            nominal_value=fossil_share * consumption_total/0.58 * number_timesteps / 8760, summe

        # create simple transformer object representing a gas power plant
        solph.LinearTransformer(
            label="pp_gas",
            inputs={bgas: solph.Flow()},
            outputs={bel: solph.Flow(nominal_value=10e10, variable_costs=0)},
            conversion_factors={bel: 0.58})

        # create storage object representing a battery
        solph.Storage(
            label='storage',
            inputs={bel: solph.Flow(variable_costs=1)},
            outputs={bel: solph.Flow(variable_costs=1)},
            capacity_loss=0.00, initial_capacity=0,
            nominal_input_capacity_ratio=1/6,
            nominal_output_capacity_ratio=1/6,
            inflow_conversion_factor=1, outflow_conversion_factor=0.8,
            fixed_costs=35,
            investment=solph.Investment(ep_costs=epc_storage),
        )
```

```
Out[3]: <oemof.solph.network.Storage at 0x7ff3f9796a68>
```

1.20 4) Optimize the energy system - set fossil share but release PV and wind installed capacity

1.20.1 This takes around one minute...

```
In [4]: logging.info('Optimize the energy system')
        om = solph.OperationalModel(energysystem)

        if debug:
            filename = os.path.join(
                helpers.extend_basic_path('lp_files'), 'storage_invest.lp')
            logging.info('Store lp-file in {0}.'.format(filename))
            om.write(filename, io_options={'symbolic_solver_labels': True})

        logging.info('Solve the optimization problem')
        om.solve(solver=solvername, solve_kwargs={'tee': tee_switch})
```

```
Welcome to the CBC MILP Solver
Version: 2.8.12
Build Date: Sep  8 2014

command line - /usr/bin/cbc -printingOptions all -import /tmp/tmpmiyfew1e.pyomo.lp -import -stat
Option for printingOptions changed from normal to all
Current default (if $ as parameter) for import is /tmp/tmpmiyfew1e.pyomo.lp
Presolve 43796 (-52569) rows, 35038 (-43808) columns and 135445 (-113902) elements
```

```
Statistics for presolved model


Problem has 43796 rows, 35038 columns (17521 with objective) and 135445 elements
Column breakdown:
26278 of type 0.0->inf, 8760 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
8758 of type E 0.0, 0 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
8760 of type G other, 26277 of type L 0.0, 0 of type L 1.0,
1 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 43796 (-52569) rows, 35038 (-43808) columns and 135445 (-113902) elements
Perturbing problem by 0.001 %% of 3532.8076 - largest nonzero change 7.2720075e-05 (%% 0.0146452
0  Obj 0 Primal inf 1.4875083e+10 (8760)
443  Obj 2263.5021 Primal inf 1.13491e+10 (8317)
886  Obj 3764.7789 Primal inf 9.3132145e+09 (7874)
1329  Obj 4850.8289 Primal inf 7.8277814e+09 (7431)
1772  Obj 5560.3583 Primal inf 6.6990211e+09 (6989)
2215  Obj 69651520 Primal inf 7.0647918e+09 (2018)
2658  Obj 1.1153288e+08 Primal inf 1.528498e+09 (3724)
3101  Obj 1.4627516e+08 Primal inf 2.6974334e+08 (1089)
3544  Obj 1.8016889e+08 Primal inf 34962027 (252)
3718  Obj 1.8164757e+08
Optimal - objective value 1.8164317e+08
After Postsolve, objective 1.8164317e+08, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 181643166.1 - 3718 iterations time 6.892, Presolve 2.30
Total time (CPU seconds):       9.18   (Wallclock seconds):       9.34
```

```
Out[4]: {'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed',
```

1.21 4) Check the results - set fossil share but release PV and wind installed capacity

1.21.1 80% renewable share can be also achieved with slightly more installed capacity of wind and PV than the fixed values from before and no storage.

```
In [5]: storage = energysystem.groups['storage']
        wind_inst = energysystem.groups['wind']
        pv_inst = energysystem.groups['pv']
        myresults = outputlib.DataFramePlot(energy_system=energysystem)

        # electrical output of natural gas power plant
        pp_gas = myresults.slice_by(obj_label='pp_gas', type='to_bus',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')
        # electrical demand
        demand = myresults.slice_by(obj_label='demand',
                                    date_from='2012-01-01 00:00:00',
                                    date_to='2012-12-31 23:00:00')

        # electrical output of wind power plant
        wind = myresults.slice_by(obj_label='wind',
                                  date_from='2012-01-01 00:00:00',
                                  date_to='2012-12-31 23:00:00')

        # electrical output of pv power plant
        pv = myresults.slice_by(obj_label='pv',
                                date_from='2012-01-01 00:00:00',
                                date_to='2012-12-31 23:00:00')

        import pprint as pp
        pp.pprint({
                   'wind_inst_MW': energysystem.results[wind_inst][bel].invest/1000,
                   'pv_inst_MW': energysystem.results[pv_inst][bel].invest/1000,
                   'storage_cap_GWh': energysystem.results[storage][storage].invest/1e6,
                   'res_share': 1 - pp_gas.sum()/demand.sum(),
                'objective': energysystem.results.objective})
```

```
{'objective': 181643168.87313932,
 'pv_inst_MW': 675.04257,
 'res_share': val    0.8
dtype: float64,
 'storage_cap_GWh': 0.0,
 'wind_inst_MW': 1170.6638}
```

1.22 4) Plot one week - set fossil share but release PV and wind installed capacity

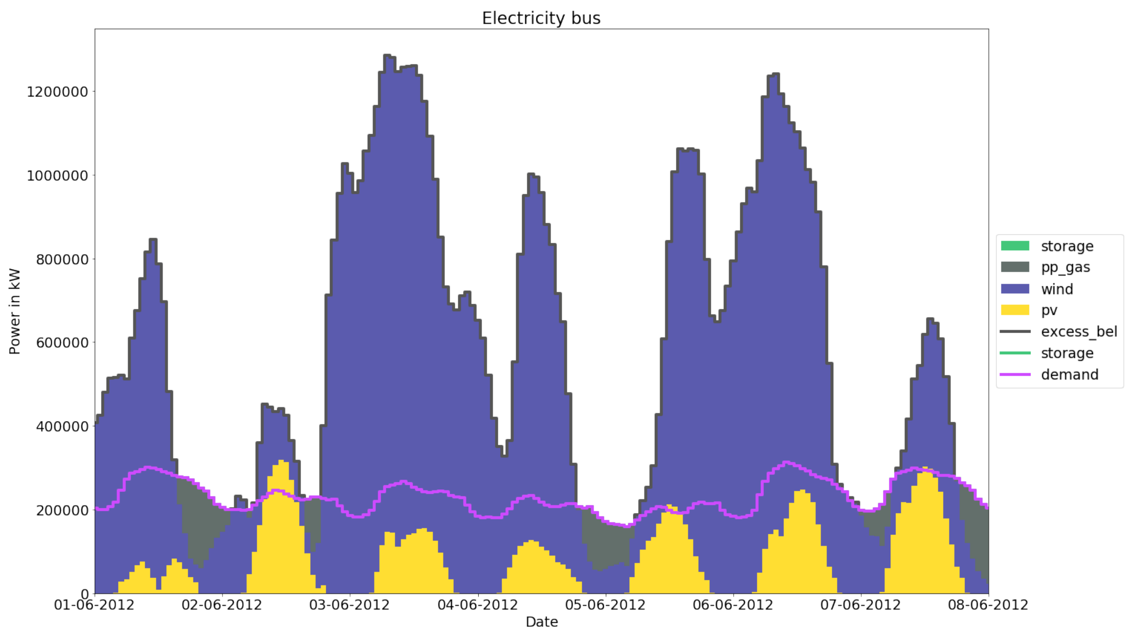
```
In [6]: cdict = {'wind': '#5b5bae',
                 'pv': '#ffde32',
                 'storage': '#42c77a',
                 'pp_gas': '#636f6b',
                 'demand': '#ce4aff',
                 'excess_bel': '#555555'}

        myplot = outputlib.DataFramePlot(energy_system=energysystem)

        # Plotting the balance around the electricity plot for one week using a
        # combined stacked plot
        fig = plt.figure(figsize=(24, 14))
        plt.rc('legend', **{'fontsize': 19})
        plt.rcParams.update({'font.size': 19})
plt.style.use('grayscale')

handles, labels = myplot.io_plot(
    bus_label='electricity', cdict=cdict,
    barorder=['pv', 'wind', 'pp_gas', 'storage'],
    lineorder=['demand', 'storage', 'excess_bel'],
    line_kwa={'linewidth': 4},
    ax=fig.add_subplot(1, 1, 1),
    date_from="2012-06-01 00:00:00",
    date_to="2012-06-08 00:00:00",
    )
myplot.ax.set_ylabel('Power in kW')
myplot.ax.set_xlabel('Date')
myplot.ax.set_title("Electricity bus")
myplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m-%Y')
myplot.outside_legend(handles=handles, labels=labels)

plt.show()
```


Electricity bus
Power in kW
Date
storage
pp_gas
wind
pv
excess_bel
storage
demand