Ribasim

Examples

Examples

1 Basic model with static forcing

```
from pathlib import Path

import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import ribasim
```

Setup the basins:

```
profile = pd.DataFrame(
    data={
        "node_id": [1, 1, 3, 3, 6, 6, 9, 9],
        "area": [0.01, 1000.0] * 4,
        "level": [0.0, 1.0] * 4,
    }
)

# Convert steady forcing to m/s
# 2 mm/d precipitation, 1 mm/d evaporation
seconds_in_day = 24 * 3600
precipitation = 0.002 / seconds_in_day
evaporation = 0.001 / seconds_in_day

static = pd.DataFrame(
    data={
        "node_id": [0],
        "potential_evaporation": [evaporation],
        "precipitation": [precipitation],
    }
)
static = static.iloc[[0, 0, 0, 0]]
static["node_id"] = [1, 3, 6, 9]

basin = ribasim.Basin(profile=profile, static=static)
```

Setup linear resistance:

```
linear_resistance = ribasim.LinearResistance(
    static=pd.DataFrame(
        data={"node_id": [10, 12], "resistance": [5e3, (3600.0 * 24) / 100.0]}
    )
```

```
)
```

Setup Manning resistance:

```
manning_resistance = ribasim.ManningResistance(
    static=pd.DataFrame(
        data={
            "node_id": [2],
            "length": [900.0],
            "manning_n": [0.04],
            "profile_width": [6.0],
            "profile_slope": [3.0],
        }
    )
)
```

Set up a rating curve node:

```
# Discharge: lose 1% of storage volume per day at storage = 1000.0.
q1000 = 1000.0 * 0.01 / seconds_in_day

rating_curve = ribasim.TabulatedRatingCurve(
    static=pd.DataFrame(
        data={
            "node_id": [4, 4],
            "level": [0.0, 1.0],
            "flow_rate": [0.0, q1000],
        }
    )
)
```

Setup fractional flows:

```
fractional_flow = ribasim.FractionalFlow(
    static=pd.DataFrame(
        data={
            "node_id": [5, 8, 13],
            "fraction": [0.3, 0.6, 0.1],
        }
    )
)
```

Setup pump:

```
pump = ribasim.Pump(
    static=pd.DataFrame(
        data={
            "node_id": [7],
```

```
                "flow_rate": [0.5 / 3600],
            }
        )
    )
```

Setup level boundary:

```
    level_boundary = ribasim.LevelBoundary(
        static=pd.DataFrame(
            data={
                "node_id": [11, 17],
                "level": [0.5, 1.5],
            }
        )
    )
```

Setup flow boundary:

```
    flow_boundary = ribasim.FlowBoundary(
        static=pd.DataFrame(
            data={
                "node_id": [15, 16],
                "flow_rate": [1e-4, 1e-4],
            }
        )
    )
```

Setup terminal:

```
    terminal = ribasim.Terminal(
        static=pd.DataFrame(
            data={
                "node_id": [14],
            }
        )
    )
```

Set up the nodes:

```
    xy = np.array(
        [
            (0.0, 0.0),  # 1: Basin,
            (1.0, 0.0),  # 2: ManningResistance
            (2.0, 0.0),  # 3: Basin
            (3.0, 0.0),  # 4: TabulatedRatingCurve
            (3.0, 1.0),  # 5: FractionalFlow
            (3.0, 2.0),  # 6: Basin
            (4.0, 1.0),  # 7: Pump
```

```
        (4.0, 0.0),  # 8: FractionalFlow
        (5.0, 0.0),  # 9: Basin
        (6.0, 0.0),  # 10: LinearResistance
        (2.0, 2.0),  # 11: LevelBoundary
        (2.0, 1.0),  # 12: LinearResistance
        (3.0, -1.0),  # 13: FractionalFlow
        (3.0, -2.0),  # 14: Terminal
        (3.0, 3.0),  # 15: FlowBoundary
        (0.0, 1.0),  # 16: FlowBoundary
        (6.0, 1.0),  # 17: LevelBoundary
    ]
)
node_xy = gpd.points_from_xy(x=xy[:, 0], y=xy[:, 1])

node_id, node_type = ribasim.Node.node_ids_and_types(
    basin,
    manning_resistance,
    rating_curve,
    pump,
    fractional_flow,
    linear_resistance,
    level_boundary,
    flow_boundary,
    terminal,
)

# Make sure the feature id starts at 1: explicitly give an index.
node = ribasim.Node(
    df=gpd.GeoDataFrame(
        data={"node_type": node_type},
        index=pd.Index(node_id, name="fid"),
        geometry=node_xy,
        crs="EPSG:28992",
    )
)
```

Setup the edges:

```
from_id = np.array(
    [1, 2, 3, 4, 4, 5, 6, 8, 7, 9, 11, 12, 4, 13, 15, 16, 10], dtype=np.int64
)
to_id = np.array(
    [2, 3, 4, 5, 8, 6, 7, 9, 9, 10, 12, 3, 13, 14, 6, 1, 17], dtype=np.int64
)
lines = node.geometry_from_connectivity(from_id, to_id)
edge = ribasim.Edge(
    df=gpd.GeoDataFrame(
        data={
            "from_node_id": from_id,
            "to node id": to id,
```

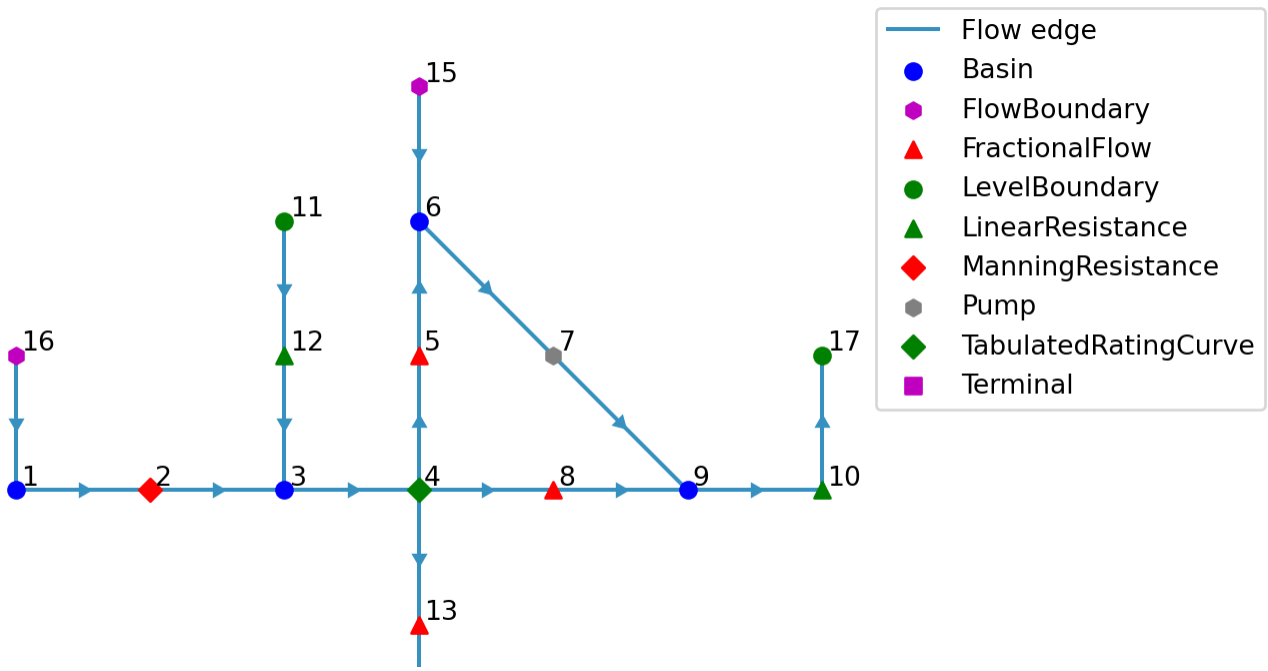
```
                "edge_type": len(from_id) * ["flow"],
            },
            geometry=lines,
            crs="EPSG:28992",
        )
    )
```

Setup a model:

```
    model = ribasim.Model(
        network=ribasim.Network(
            node=node,
            edge=edge,
        ),
        basin=basin,
        level_boundary=level_boundary,
        flow_boundary=flow_boundary,
        pump=pump,
        linear_resistance=linear_resistance,
        manning_resistance=manning_resistance,
        tabulated_rating_curve=rating_curve,
        fractional_flow=fractional_flow,
        terminal=terminal,
        starttime="2020-01-01 00:00:00",
        endtime="2021-01-01 00:00:00",
    )
```

Let's take a look at the model:

```
    model.plot()
```



Write the model to a TOML and GeoPackage:

```
datadir = Path("data")
model.write(datadir / "basic/ribasim.toml")
```

```
PosixPath('data/basic/ribasim.toml')
```

2 Update the basic model with transient forcing

This assumes you have already created the basic model with static forcing.

```
import numpy as np
import pandas as pd
import ribasim
import xarray as xr
```

```
model = ribasim.Model(filepath=datadir / "basic/ribasim.toml")
```

```
Can't read from data/basic/database.gpkg:Basin / area
```

```
time = pd.date_range(model.starttime, model.endtime)
day_of_year = time.day_of_year.to_numpy()
seconds_per_day = 24 * 60 * 60
evaporation = (
    (-1.0 * np.cos(day_of_year / 365.0 * 2 * np.pi) + 1.0) * 0.0025 / seconds_per_day
)
rng = np.random.default_rng(seed=0)
precipitation = (
    rng.lognormal(mean=-1.0, sigma=1.7, size=time.size) * 0.001 / seconds_per_day
)
```

We'll use xarray to easily broadcast the values.

```
timeseries = (
    pd.DataFrame(
        data={
            "node_id": 1,
            "time": pd.date_range(model.starttime, model.endtime),
            "drainage": 0.0,
            "potential_evaporation": evaporation,
            "infiltration": 0.0,
```

```
                "precipitation": precipitation,
                "urban_runoff": 0.0,
            }
        )
        .set_index("time")
        .to_xarray()
    )

    basin_ids = model.basin.static.df["node_id"].to_numpy()
    basin_nodes = xr.DataArray(
        np.ones(len(basin_ids)), coords={"node_id": basin_ids}, dims=["node_id"]
    )
    forcing = (timeseries * basin_nodes).to_dataframe().reset_index()
```

```
    state = pd.DataFrame(
        data={
            "node_id": basin_ids,
            "level": 1.4,
        }
    )
```

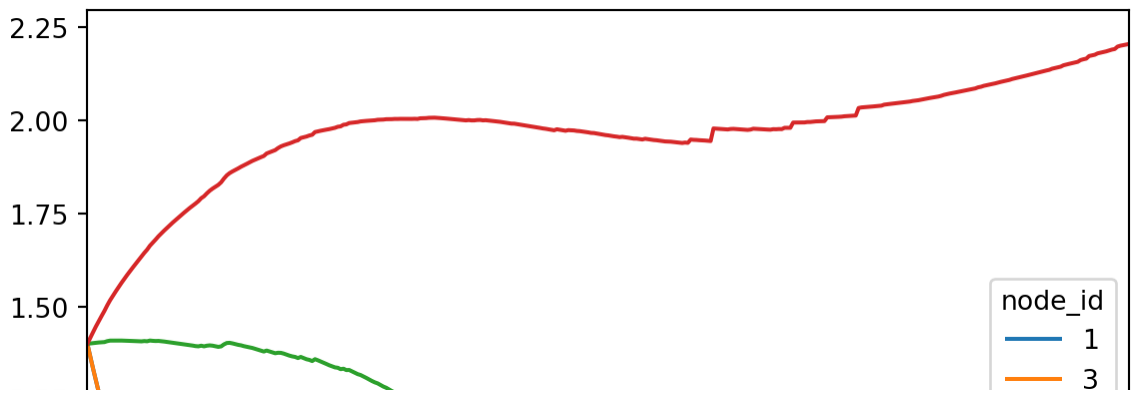
```
    model.basin.time.df = forcing
    model.basin.state.df = state
```

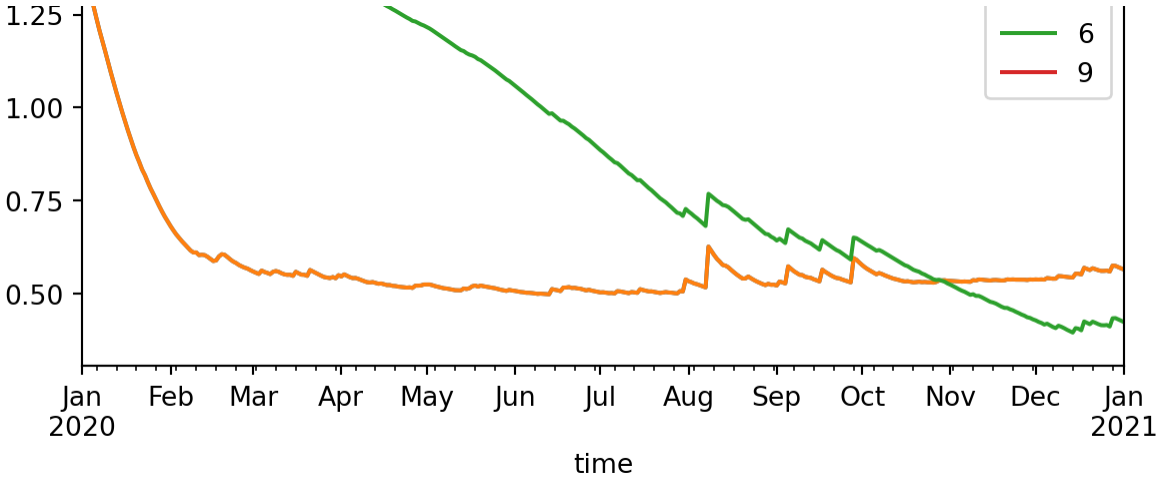
```
    model.write(datadir / "basic_transient/ribasim.toml")
```

```
PosixPath('data/basic_transient/ribasim.toml')
```

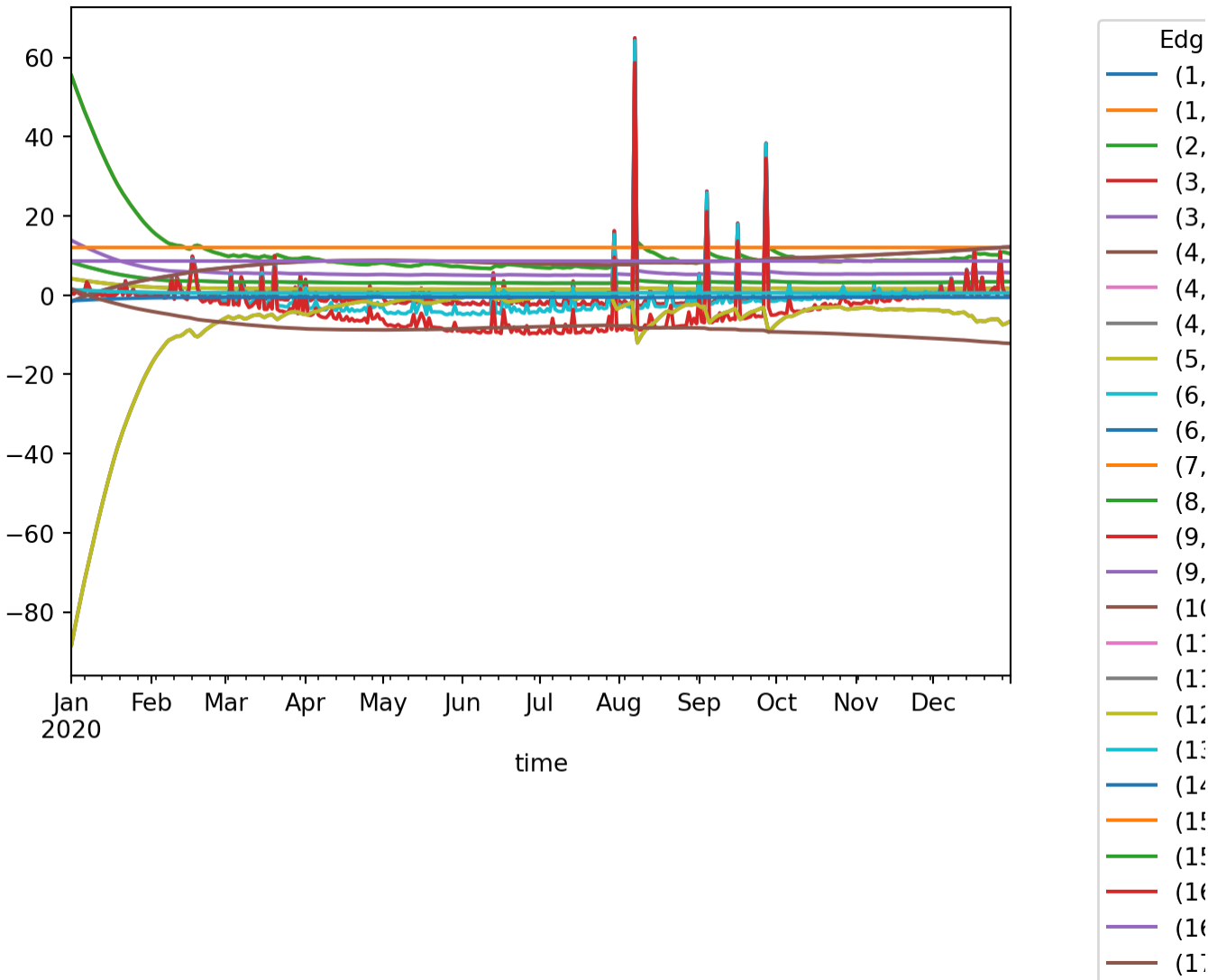
Now run the model with `ribasim basic_transient/ribasim.toml`. After running the model, read back the results:

```
    df_basin = pd.read_feather(datadir / "basic_transient/results/basin.arrow")
    df_basin_wide = df_basin.pivot_table(
        index="time", columns="node_id", values=["storage", "level"]
    )
    df_basin_wide["level"].plot()
```






```
df_flow = pd.read_feather(datadir / "basic_transient/results/flow.arrow")
df_flow["edge"] = list(zip(df_flow.from_node_id, df_flow.to_node_id))
df_flow["flow_m3d"] = df_flow.flow_rate * 86400
ax = df_flow.pivot_table(index="time", columns="edge", values="flow_m3d").plot()
ax.legend(bbox_to_anchor=(1.3, 1), title="Edge")
```



```
type(df_flow)
```

```
pandas.core.frame.DataFrame
```

3 Model with discrete control

The model constructed below consists of a single basin which slowly drains trough a `TabulatedRatingCurve`, but is held within a range around a target level (setpoint) by two connected pumps. These two pumps behave like a reversible pump. When pumping can be done in only one direction, and the other direction is only possible under gravity, use an Outlet for that direction.

Set up the nodes:

```
xy = np.array(
    [
        (0.0, 0.0),  # 1: Basin
        (1.0, 1.0),  # 2: Pump
        (1.0, -1.0),  # 3: Pump
        (2.0, 0.0),  # 4: LevelBoundary
        (-1.0, 0.0),  # 5: TabulatedRatingCurve
        (-2.0, 0.0),  # 6: Terminal
        (1.0, 0.0),  # 7: DiscreteControl
    ]
)

node_xy = gpd.points_from_xy(x=xy[:, 0], y=xy[:, 1])

node_type = [
    "Basin",
    "Pump",
    "Pump",
    "LevelBoundary",
    "TabulatedRatingCurve",
    "Terminal",
    "DiscreteControl",
]

# Make sure the feature id starts at 1: explicitly give an index.
node = ribasim.Node(
    df=gpd.GeoDataFrame(
        data={"node_type": node_type},
        index=pd.Index(np.arange(len(xy)) + 1, name="fid"),
        geometry=node_xy,
        crs="EPSG:28992",
    )
```

```
    )
)
```

Setup the edges:

```
from_id = np.array([1, 3, 4, 2, 1, 5, 7, 7], dtype=np.int64)
to_id = np.array([3, 4, 2, 1, 5, 6, 2, 3], dtype=np.int64)

edge_type = 6 * ["flow"] + 2 * ["control"]

lines = node.geometry_from_connectivity(from_id, to_id)
edge = ribasim.Edge(
    df=gpd.GeoDataFrame(
        data={"from_node_id": from_id, "to_node_id": to_id, "edge_type": edge_type},
        geometry=lines,
        crs="EPSG:28992",
    )
)
```

Setup the basins:

```
profile = pd.DataFrame(
    data={
        "node_id": [1, 1],
        "area": [1000.0, 1000.0],
        "level": [0.0, 1.0],
    }
)

state = pd.DataFrame(data={"node_id": [1], "level": [20.0]})

basin = ribasim.Basin(profile=profile, state=state)
```

Setup the discrete control:

```
condition = pd.DataFrame(
    data={
        "node_id": 3 * [7],
        "listen_feature_id": 3 * [1],
        "variable": 3 * ["level"],
        "greater_than": [5.0, 10.0, 15.0],  # min, setpoint, max
    }
)

logic = pd.DataFrame(
    data={
        "node_id": 5 * [7],
        "truth_state": ["FFF", "U**", "T*F", "**D", "TTT"],
        "control_state": ["in", "in", "none", "out", "out"],
```

```
    }
)

discrete_control = ribasim.DiscreteControl(condition=condition, logic=logic)
```

The above control logic can be summarized as follows: - If the level gets above the maximum, activate the control state “out” until the setpoint is reached; - If the level gets below the minimum, active the control state “in” until the setpoint is reached; - Otherwise activate the control state “none”.

Setup the pump:

```
pump = ribasim.Pump(
    static=pd.DataFrame(
        data={
            "node_id": 3 * [2] + 3 * [3],
            "control_state": 2 * ["none", "in", "out"],
            "flow_rate": [0.0, 2e-3, 0.0, 0.0, 0.0, 2e-3],
        }
    )
)
```

The pump data defines the following:

Control state	Pump #2 flow rate (m/s)	Pump #3 flow rate (m/s)
“none”	0.0	0.0
“in”	2e-3	0.0
“out”	0.0	2e-3

Setup the level boundary:

```
level_boundary = ribasim.LevelBoundary(
    static=pd.DataFrame(data={"node_id": [4], "level": [10.0]})
)
```

Setup the rating curve:

```
rating_curve = ribasim.TabulatedRatingCurve(
    static=pd.DataFrame(
        data={"node_id": 2 * [5], "level": [2.0, 15.0], "flow_rate": [0.0, 1e-3]}
    )
)
```

Setup the terminal:

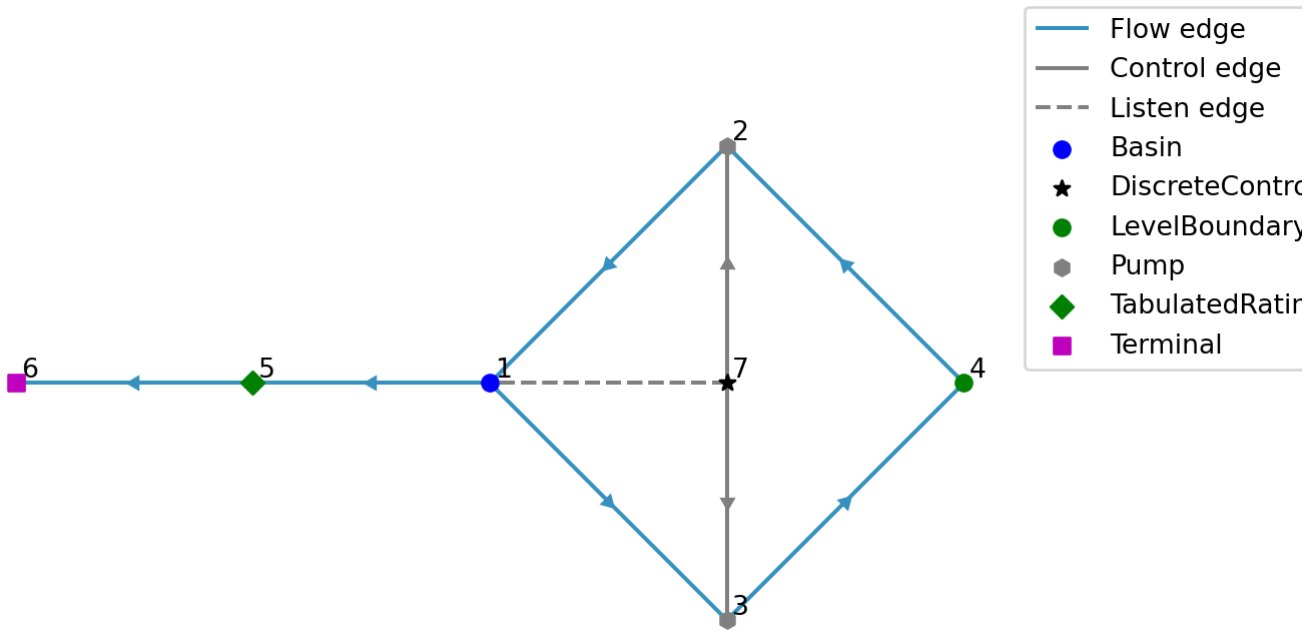
```
terminal = ribasim.Terminal(static=pd.DataFrame(data={"node_id": [6]}))
```

Setup a model:

```
model = ribasim.Model(
    network=ribasim.Network(
        node=node,
        edge=edge,
    ),
    basin=basin,
    pump=pump,
    level_boundary=level_boundary,
    tabulated_rating_curve=rating_curve,
    terminal=terminal,
    discrete_control=discrete_control,
    starttime="2020-01-01 00:00:00",
    endtime="2021-01-01 00:00:00",
)
```

Let's take a look at the model:

```
model.plot()
```



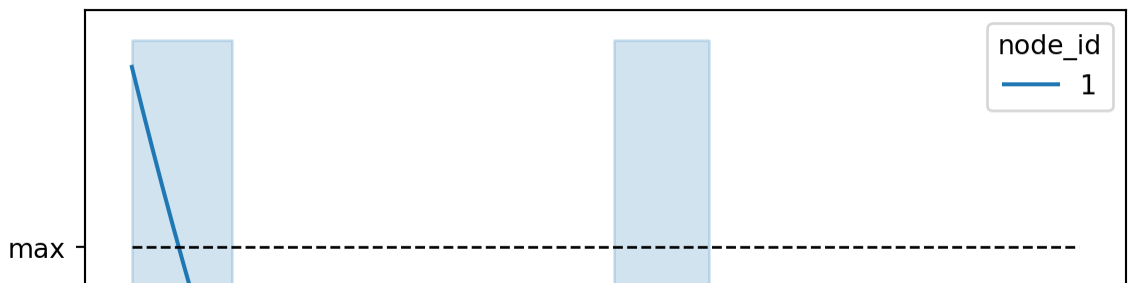
Listen edges are plotted with a dashed line since they are not present in the "Edge / static" schema but only in the "Control / condition" schema.

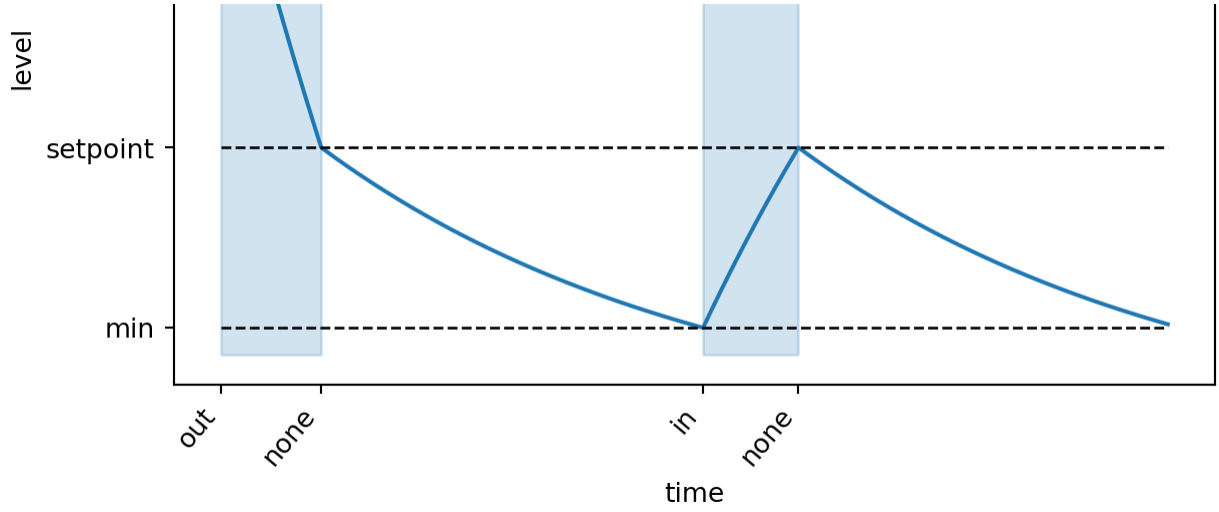
```
datadir = Path("data")
model.write(datadir / "level_setpoint_with_minmax/ribasim.toml")
```

```
PosixPath('data/level_setpoint_with_minmax/ribasim.toml')
```

Now run the model with `level_setpoint_with_minmax/ribasim.toml`. After running the model, read back the results:

```
from matplotlib.dates import date2num

df_basin = pd.read_feather(datadir / "level_setpoint_with_minmax/results/basin.arrow")
df_basin_wide = df_basin.pivot_table(
    index="time", columns="node_id", values=["storage", "level"]
)

ax = df_basin_wide["level"].plot()

greater_than = model.discrete_control.condition.df.greater_than

ax.hlines(
    greater_than,
    df_basin.time[0],
    df_basin.time.max(),
    lw=1,
    ls="--",
    color="k",
)

df_control = pd.read_feather(
    datadir / "level_setpoint_with_minmax/results/control.arrow"
)

y_min, y_max = ax.get_ybound()
ax.fill_between(df_control.time[:2], 2 * [y_min], 2 * [y_max], alpha=0.2, color="C0")
ax.fill_between(df_control.time[2:4], 2 * [y_min], 2 * [y_max], alpha=0.2, color="C0")

ax.set_xticks(
    date2num(df_control.time).tolist(),
    df_control.control_state.tolist(),
    rotation=50,
)

ax.set_yticks(greater_than, ["min", "setpoint", "max"])
ax.set_ylabel("level")
plt.show()
```





The highlighted regions show where a pump is active.

Let's print an overview of what happened with control:

```
model.print_discrete_control_record(
    datadir / "level_setpoint_with_minmax/results/control.arrow"
)
```

```
0. At 2020-01-01 00:00:00 the control node with ID 7 reached truth state TTT:
    For node ID 1 (Basin): level > 5.0
    For node ID 1 (Basin): level > 10.0
    For node ID 1 (Basin): level > 15.0

   This yielded control state "out":
    For node ID 2 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan
    For node ID 3 (Pump): active = <NA>, flow_rate = 0.002, min_flow_rate = nan, max_flow_rate
= nan

1. At 2020-02-08 19:07:33.550000 the control node with ID 7 reached truth state TFF:
    For node ID 1 (Basin): level > 5.0
    For node ID 1 (Basin): level < 10.0
    For node ID 1 (Basin): level < 15.0

   This yielded control state "none":
    For node ID 2 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan
    For node ID 3 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan

2. At 2020-07-05 09:06:07.383000 the control node with ID 7 reached truth state FFF:
    For node ID 1 (Basin): level < 5.0
    For node ID 1 (Basin): level < 10.0
    For node ID 1 (Basin): level < 15.0
```

```
   This yielded control state "in":
    For node ID 2 (Pump): active = <NA>, flow_rate = 0.002, min_flow_rate = nan, max_flow_rate
= nan
    For node ID 3 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan

3. At 2020-08-11 06:17:03.985000 the control node with ID 7 reached truth state TTF:
    For node ID 1 (Basin): level > 5.0
    For node ID 1 (Basin): level > 10.0
    For node ID 1 (Basin): level < 15.0

   This yielded control state "none":
    For node ID 2 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan
    For node ID 3 (Pump): active = <NA>, flow_rate = 0.0, min_flow_rate = nan, max_flow_rate =
nan
```

Note that crossing direction specific truth states (containing “U”, “D”) are not present in this overview even though they are part of the control logic. This is because in the control logic for this model these truth states are only used to sustain control states, while the overview only shows changes in control states.

4 Model with PID control

Set up the nodes:

```
xy = np.array(
    [
        (0.0, 0.0),  # 1: FlowBoundary
        (1.0, 0.0),  # 2: Basin
        (2.0, 0.5),  # 3: Pump
        (3.0, 0.0),  # 4: LevelBoundary
        (1.5, 1.0),  # 5: PidControl
        (2.0, -0.5),  # 6: outlet
        (1.5, -1.0),  # 7: PidControl
    ]
)

node_xy = gpd.points_from_xy(x=xy[:, 0], y=xy[:, 1])

node_type = [
    "FlowBoundary",
    "Basin",
    "Pump",
    "LevelBoundary",
    "PidControl",
    "Outlet",
    "PidControl",
]
```

```
# Make sure the feature id starts at 1: explicitly give an index.
node = ribasim.Node(
    df=gpd.GeoDataFrame(
        data={"node_type": node_type},
        index=pd.Index(np.arange(len(xy)) + 1, name="fid"),
        geometry=node_xy,
        crs="EPSG:28992",
    )
)
```

Setup the edges:

```
from_id = np.array([1, 2, 3, 4, 6, 5, 7], dtype=np.int64)
to_id = np.array([2, 3, 4, 6, 2, 3, 6], dtype=np.int64)

lines = node.geometry_from_connectivity(from_id, to_id)
edge = ribasim.Edge(
    df=gpd.GeoDataFrame(
        data={
            "from_node_id": from_id,
            "to_node_id": to_id,
            "edge_type": 5 * ["flow"] + 2 * ["control"],
        },
        geometry=lines,
        crs="EPSG:28992",
    )
)
```

Setup the basins:

```
profile = pd.DataFrame(
    data={"node_id": [2, 2], "level": [0.0, 1.0], "area": [1000.0, 1000.0]}
)

state = pd.DataFrame(
    data={
        "node_id": [2],
        "level": [6.0],
    }
)

basin = ribasim.Basin(profile=profile, state=state)
```

Setup the pump:

```
pump = ribasim.Pump(
    static=pd.DataFrame(
        data={
```

```
            data={
                "node_id": [3],
                "flow_rate": [0.0],  # Will be overwritten by PID controller
            }
        )
    )
```

Setup the outlet:

```
    outlet = ribasim.Outlet(
        static=pd.DataFrame(
            data={
                "node_id": [6],
                "flow_rate": [0.0],  # Will be overwritten by PID controller
            }
        )
    )
```

Setup flow boundary:

```
    flow_boundary = ribasim.FlowBoundary(
        static=pd.DataFrame(data={"node_id": [1], "flow_rate": [1e-3]})
    )
```

Setup flow boundary:

```
    level_boundary = ribasim.LevelBoundary(
        static=pd.DataFrame(
            data={
                "node_id": [4],
                "level": [1.0],  # Not relevant
            }
        )
    )
```

Setup PID control:

```
    pid_control = ribasim.PidControl(
        time=pd.DataFrame(
            data={
                "node_id": 4 * [5, 7],
                "time": [
                    "2020-01-01 00:00:00",
                    "2020-01-01 00:00:00",
                    "2020-05-01 00:00:00",
                    "2020-05-01 00:00:00",
                    "2020-07-01 00:00:00",
                    "2020-07-01 00:00:00",
                    "2020-12-01 00:00:00"
```

```
                    "2020-12-01 00:00:00",
                ],
                "listen_node_id": 4 * [2, 2],
                "target": [5.0, 5.0, 5.0, 5.0, 7.5, 7.5, 7.5, 7.5],
                "proportional": 4 * [-1e-3, 1e-3],
                "integral": 4 * [-1e-7, 1e-7],
                "derivative": 4 * [0.0, 0.0],
            }
        )
    )
```

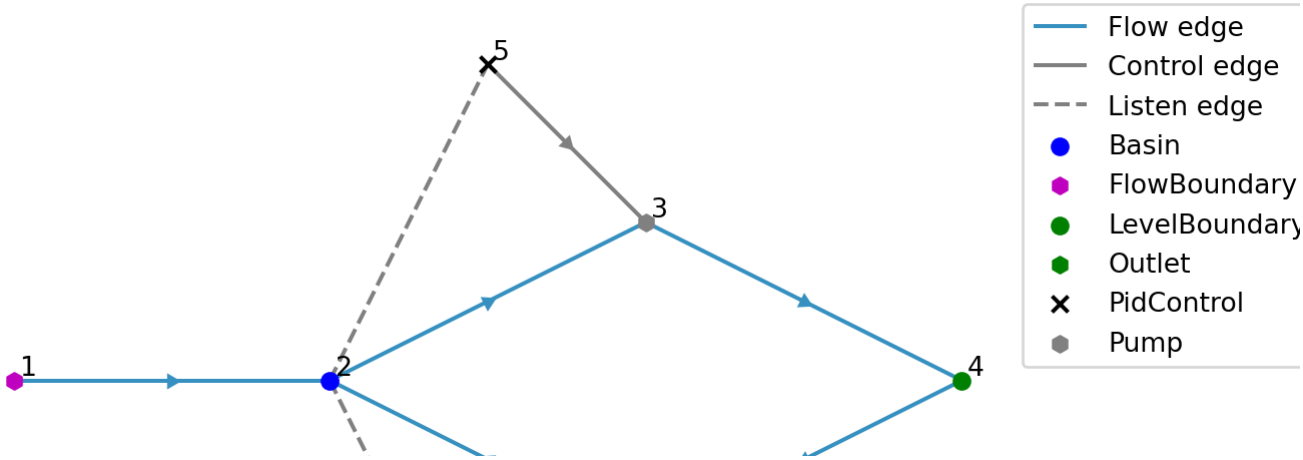
Note that the coefficients for the pump and the outlet are equal in magnitude but opposite in sign. This way the pump and the outlet equally work towards the same goal, while having opposite effects on the controlled basin due to their connectivity to this basin.

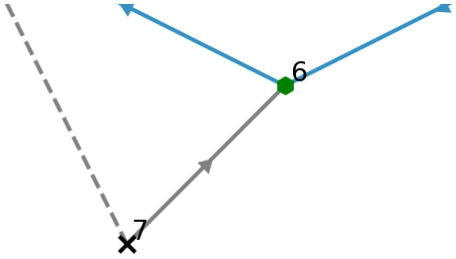
Setup a model:

```
model = ribasim.Model(
    network=ribasim.Network(
        node=node,
        edge=edge,
    ),
    basin=basin,
    flow_boundary=flow_boundary,
    level_boundary=level_boundary,
    pump=pump,
    outlet=outlet,
    pid_control=pid_control,
    starttime="2020-01-01 00:00:00",
    endtime="2020-12-01 00:00:00",
)
```

Let's take a look at the model:

```
model.plot()
```





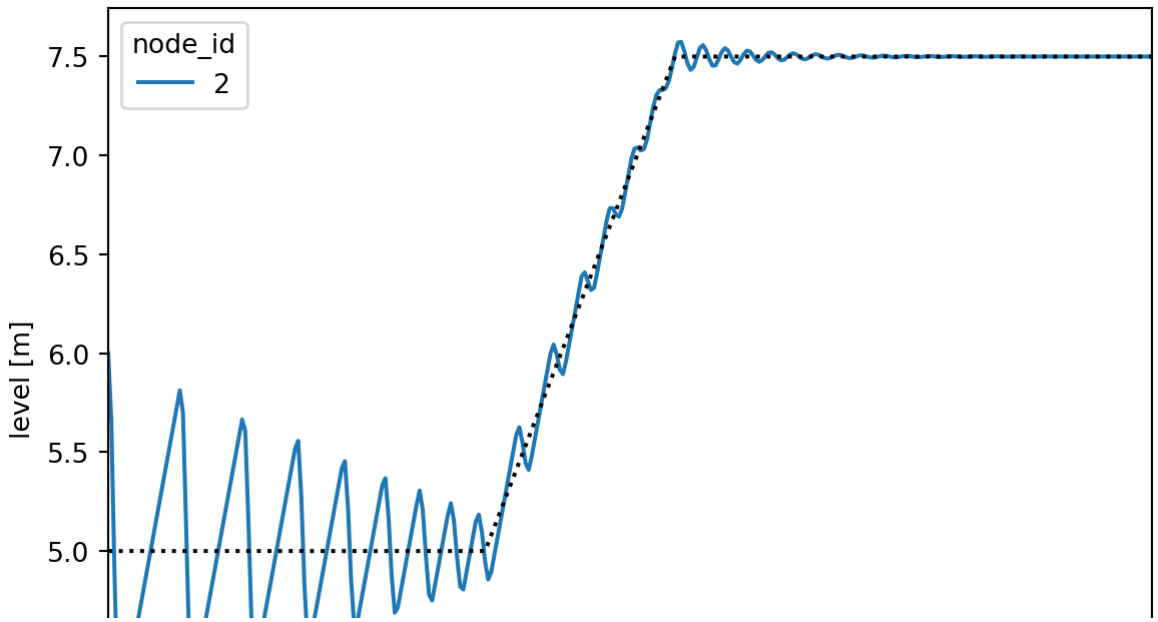
Write the model to a TOML and GeoPackage:

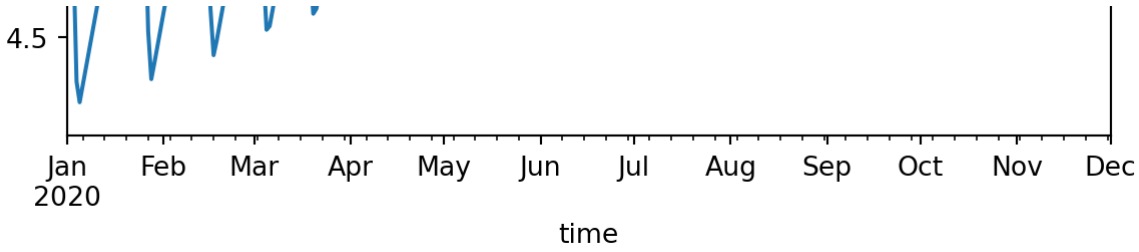
```
datadir = Path("data")
model.write(datadir / "pid_control/ribasim.toml")
```

```
PosixPath('data/pid_control/ribasim.toml')
```

Now run the model with `ribasim pid_control/ribasim.toml`. After running the model, read back the results:

```
from matplotlib.dates import date2num

df_basin = pd.read_feather(datadir / "pid_control/results/basin.arrow")
df_basin_wide = df_basin.pivot_table(
    index="time", columns="node_id", values=["storage", "level"]
)
ax = df_basin_wide["level"].plot()
ax.set_ylabel("level [m]")

# Plot target level
level_demands = model.pid_control.time.df.target.to_numpy()[:4]
times = date2num(model.pid_control.time.df.time)[:4]
ax.plot(times, level_demands, color="k", ls=":", label="target level")
pass
```





5 Model with allocation (user demand)

Setup the nodes:

```
xy = np.array(
    [
        (0.0, 0.0),  # 1: FlowBoundary
        (1.0, 0.0),  # 2: Basin
        (1.0, 1.0),  # 3: UserDemand
        (2.0, 0.0),  # 4: LinearResistance
        (3.0, 0.0),  # 5: Basin
        (3.0, 1.0),  # 6: UserDemand
        (4.0, 0.0),  # 7: TabulatedRatingCurve
        (4.5, 0.0),  # 8: FractionalFlow
        (4.5, 0.5),  # 9: FractionalFlow
        (5.0, 0.0),  # 10: Terminal
        (4.5, 0.25),  # 11: DiscreteControl
        (4.5, 1.0),  # 12: Basin
        (5.0, 1.0),  # 13: UserDemand
    ]
)
node_xy = gpd.points_from_xy(x=xy[:, 0], y=xy[:, 1])

node_type = [
    "FlowBoundary",
    "Basin",
    "UserDemand",
    "LinearResistance",
    "Basin",
    "UserDemand",
    "TabulatedRatingCurve",
    "FractionalFlow",
    "FractionalFlow",
    "Terminal",
    "DiscreteControl",
    "Basin",
    "UserDemand",
]

# All nodes belong to allocation network id 1
```

```
node = ribasim.Node(
    df=gpd.GeoDataFrame(
        data={"node_type": node_type, "subnetwork_id": 1},
        index=pd.Index(np.arange(len(xy)) + 1, name="fid"),
        geometry=node_xy,
        crs="EPSG:28992",
    )
)
```

Setup the edges:

```
from_id = np.array(
    [1, 2, 2, 4, 5, 5, 7, 3, 6, 7, 8, 9, 12, 13, 11, 11],
    dtype=np.int64,
)
to_id = np.array(
    [2, 3, 4, 5, 6, 7, 8, 2, 5, 9, 10, 12, 13, 10, 8, 9],
    dtype=np.int64,
)
# Denote the first edge, 1 => 2, as a source edge for
# allocation network 1
subnetwork_id = len(from_id) * [None]
subnetwork_id[0] = 1
lines = node.geometry_from_connectivity(from_id, to_id)
edge = ribasim.Edge(
    df=gpd.GeoDataFrame(
        data={
            "from_node_id": from_id,
            "to_node_id": to_id,
            "edge_type": (len(from_id) - 2) * ["flow"] + 2 * ["control"],
            "subnetwork_id": subnetwork_id,
        },
        geometry=lines,
        crs="EPSG:28992",
    )
)
```

Setup the basins:

```
profile = pd.DataFrame(
    data={
        "node_id": [2, 2, 5, 5, 12, 12],
        "area": 300_000.0,
        "level": 3 * [0.0, 1.0],
    }
)

state = pd.DataFrame(data={"node_id": [2, 5, 12], "level": 1.0})
```

```
basin = ribasim.Basin(profile=profile, state=state)
```

Setup the flow boundary:

```
flow_boundary = ribasim.FlowBoundary(
    static=pd.DataFrame(
        data={
            "node_id": [1],
            "flow_rate": 2.0,
        }
    )
)
```

Setup the linear resistance:

```
linear_resistance = ribasim.LinearResistance(
    static=pd.DataFrame(
        data={
            "node_id": [4],
            "resistance": 0.06,
        }
    )
)
```

Setup the tabulated rating curve:

```
tabulated_rating_curve = ribasim.TabulatedRatingCurve(
    static=pd.DataFrame(
        data={
            "node_id": 7,
            "level": [0.0, 0.5, 1.0],
            "flow_rate": [0.0, 0.0, 2.0],
        }
    )
)
```

Setup the fractional flow:

```
fractional_flow = ribasim.FractionalFlow(
    static=pd.DataFrame(
        data={
            "node_id": [8, 8, 9, 9],
            "fraction": [0.6, 0.9, 0.4, 0.1],
            "control_state": ["divert", "close", "divert", "close"],
        }
    )
)
```

Setup the terminal:

```
terminal = ribasim.Terminal(
    static=pd.DataFrame(
        data={
            "node_id": [10],
        }
    )
)
```

Setup the discrete control:

```
condition = pd.DataFrame(
    data={
        "node_id": [11],
        "listen_feature_id": 5,
        "variable": "level",
        "greater_than": 0.52,
    }
)

logic = pd.DataFrame(
    data={
        "node_id": 11,
        "truth_state": ["T", "F"],
        "control_state": ["divert", "close"],
    }
)

discrete_control = ribasim.DiscreteControl(condition=condition, logic=logic)
```

Setup the users:

```
user_demand = ribasim.UserDemand(
    static=pd.DataFrame(
        data={
            "node_id": [6, 13],
            "demand": [1.5, 1.0],
            "return_factor": 0.0,
            "min_level": -1.0,
            "priority": [1, 3],
        }
    ),
    time=pd.DataFrame(
        data={
            "node_id": [3, 3, 3, 3],
            "demand": [0.0, 1.0, 1.2, 1.2],
            "priority": [1, 1, 2, 2],
```

```
                "return_factor": 0.0,
                "min_level": -1.0,
                "time": 2 * ["2020-01-01 00:00:00", "2020-01-20 00:00:00"],
            }
        ),
    )
```

Setup the allocation:

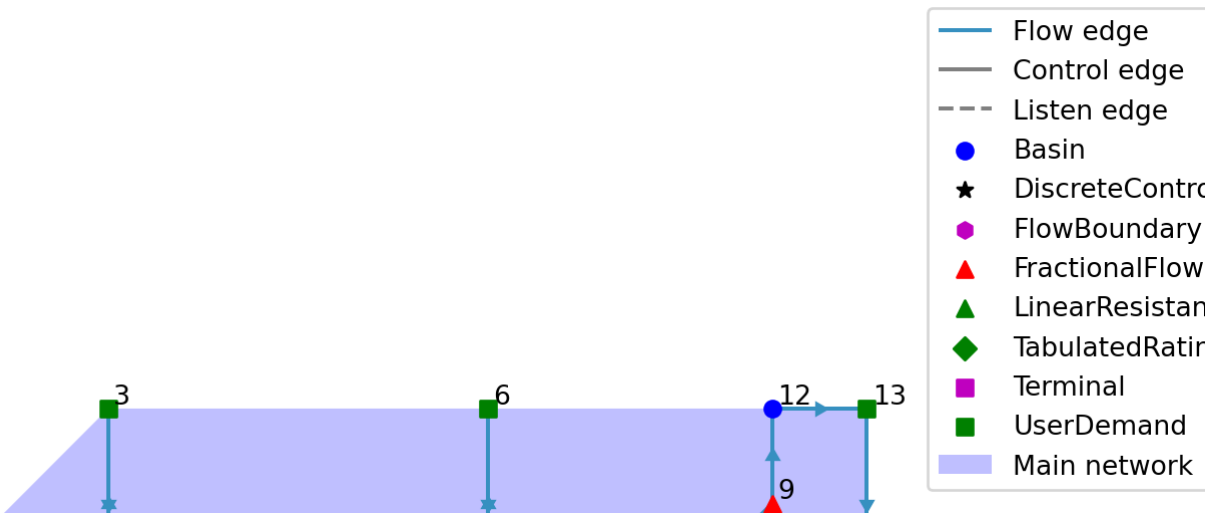
```
    allocation = ribasim.Allocation(use_allocation=True, timestep=86400)
```

Setup a model:

```
    model = ribasim.Model(
        network=ribasim.Network(
            node=node,
            edge=edge,
        ),
        basin=basin,
        flow_boundary=flow_boundary,
        linear_resistance=linear_resistance,
        tabulated_rating_curve=tabulated_rating_curve,
        terminal=terminal,
        user_demand=user_demand,
        discrete_control=discrete_control,
        fractional_flow=fractional_flow,
        allocation=allocation,
        starttime="2020-01-01 00:00:00",
        endtime="2020-01-20 00:00:00",
    )
```

Let's take a look at the model:

```
    model.plot()
```



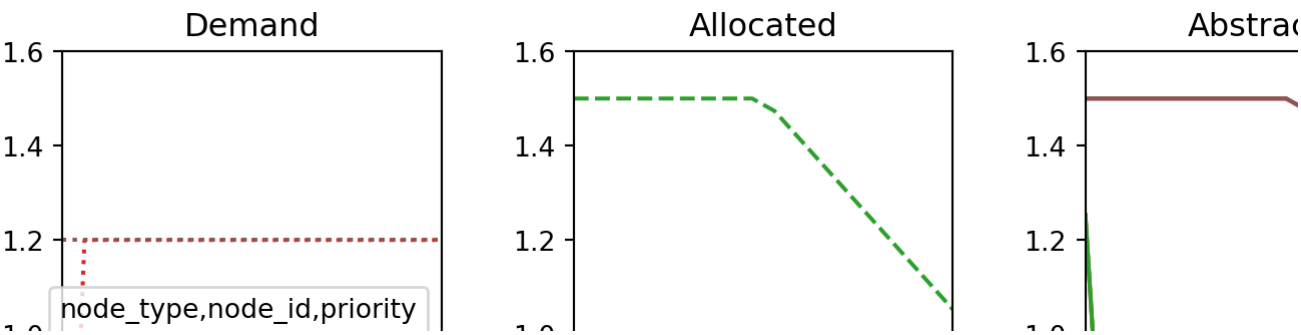
Write the model to a TOML and GeoPackage:

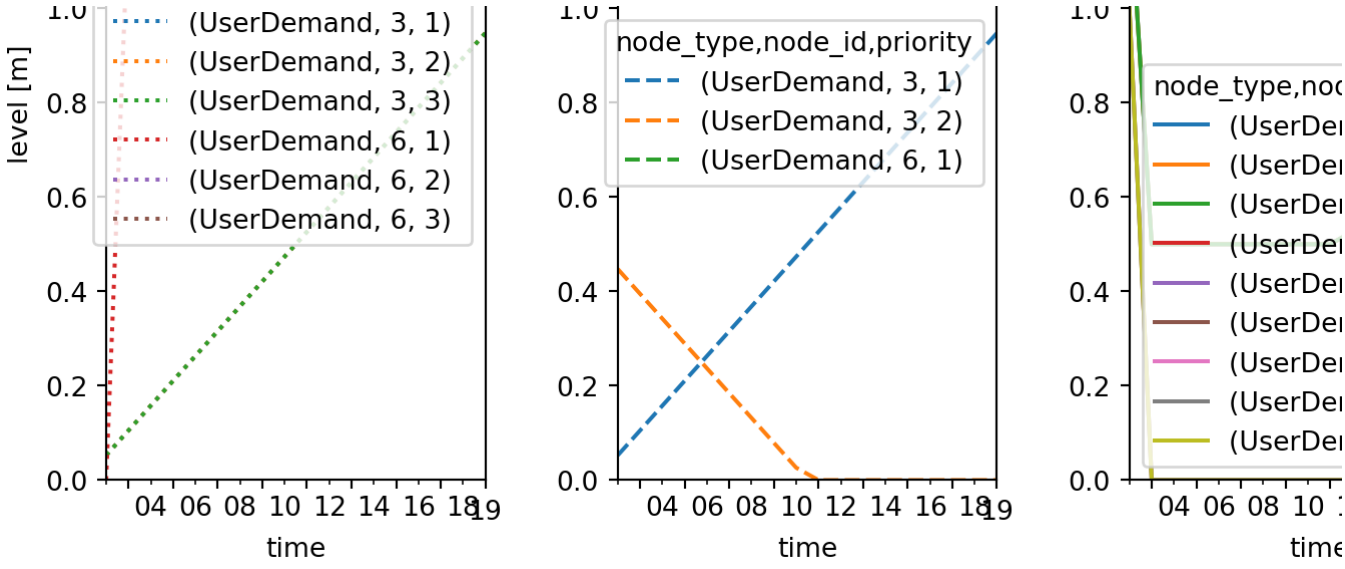
```
datadir = Path("data")
model.write(datadir / "allocation_example/ribasim.toml")
```

```
PosixPath('data/allocation_example/ribasim.toml')
```

Now run the model with `ribasim allocation_example/ribasim.toml`. After running the model, read back the results:

```
import matplotlib.ticker as plticker

df_allocation = pd.read_feather(datadir / "allocation_example/results/allocation.arrow
df_allocation_wide = df_allocation.pivot_table(
    index="time",
    columns=["node_type", "node_id", "priority"],
    values=["demand", "allocated", "realized"],
)
df_allocation_wide = df_allocation_wide.loc[:, (df_allocation_wide != 0).any(axis=0)]

fig, axs = plt.subplots(1, 3, figsize=(8, 5))

df_allocation_wide["demand"].plot(ax=axs[0], ls=":")
df_allocation_wide["allocated"].plot(ax=axs[1], ls="--")
df_allocation_wide["realized"].plot(ax=axs[2])

fig.tight_layout()
loc = plticker.MultipleLocator(2)

axs[0].set_ylabel("level [m]")

for ax, title in zip(axs, ["Demand", "Allocated", "Abstracted"]):
    ax.set_title(title)
    ax.set_ylim(0.0, 1.6)
    ax.xaxis.set_major_locator(loc)
```



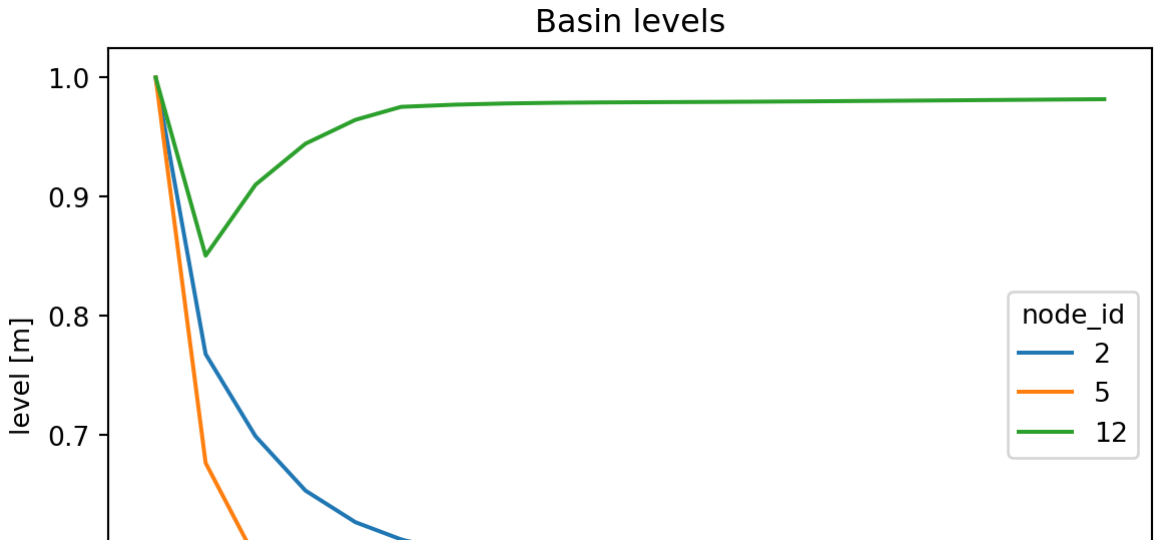


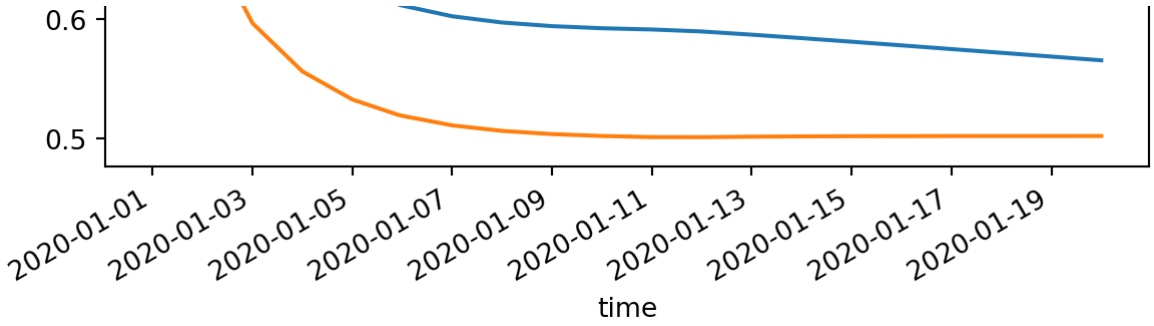
Some things to note about this plot:

- Abstraction behaves somewhat erratically at the start of the simulation. This is because allocation is based on flows computed in the physical layer, and at the start of the simulation these are not known yet.
- Although there is a plotted line for abstraction per priority, abstraction is actually accumulated over all priorities per user.

```
df_basin = pd.read_feather(datadir / "allocation_example/results/basin.arrow")
df_basin_wide = df_basin.pivot_table(
    index="time", columns="node_id", values=["storage", "level"]
)

ax = df_basin_wide["level"].plot()
ax.set_title("Basin levels")
ax.set_ylabel("level [m]")
```

```
Text(0, 0.5, 'level [m]')
```





6 Model with allocation (basin supply/demand)

Setup the nodes:

```
xy = np.array(
    [
        (0.0, 0.0),  # 1: FlowBoundary
        (1.0, 0.0),  # 2: Basin
        (2.0, 0.0),  # 3: UserDemand
        (1.0, -1.0),  # 4: LevelDemand
        (2.0, -1.0),  # 5: Basin
    ]
)
node_xy = gpd.points_from_xy(x=xy[:, 0], y=xy[:, 1])

node_type = ["FlowBoundary", "Basin", "UserDemand", "LevelDemand", "Basin"]

# Make sure the feature id starts at 1: explicitly give an index.
node = ribasim.Node(
    df=gpd.GeoDataFrame(
        data={
            "node_type": node_type,
            "subnetwork_id": 5 * [2],
        },
        index=pd.Index(np.arange(len(xy)) + 1, name="fid"),
        geometry=node_xy,
        crs="EPSG:28992",
    )
)
```

Setup the edges:

```
from_id = np.array([1, 2, 4, 3, 4])
to_id = np.array([2, 3, 2, 5, 5])
edge_type = ["flow", "flow", "control", "flow", "control"]
subnetwork_id = [2, None, None, None, None]
```

```
lines = node.geometry_from_connectivity(from_id.tolist(), to_id.tolist())
edge = ribasim.Edge(
    df=gpd.GeoDataFrame(
        data={
            "from_node_id": from_id,
            "to_node_id": to_id,
            "edge_type": edge_type,
            "subnetwork_id": subnetwork_id,
        },
        geometry=lines,
        crs="EPSG:28992",
    )
)
```

Setup the basins:

```
profile = pd.DataFrame(
    data={"node_id": [2, 2, 5, 5], "area": 1e3, "level": [0.0, 1.0, 0.0, 1.0]}
)
static = pd.DataFrame(
    data={
        "node_id": [5],
        "drainage": 0.0,
        "potential_evaporation": 0.0,
        "infiltration": 0.0,
        "precipitation": 0.0,
        "urban_runoff": 0.0,
    }
)
time = pd.DataFrame(
    data={
        "node_id": 2,
        "time": ["2020-01-01 00:00:00", "2020-01-16 00:00:00"],
        "drainage": 0.0,
        "potential_evaporation": 0.0,
        "infiltration": 0.0,
        "precipitation": [1e-6, 0.0],
        "urban_runoff": 0.0,
    },
)

state = pd.DataFrame(data={"node_id": [2, 5], "level": 0.5})
basin = ribasim.Basin(profile=profile, static=static, time=time, state=state)
```

Setup the flow boundary:

```
flow_boundary = ribasim.FlowBoundary(
    static=pd.DataFrame(data={"node_id": [1], "flow_rate": 1e-3})
)
```

Setup allocation level control:

```
level_demand = ribasim.LevelDemand(
    static=pd.DataFrame(
        data={"node_id": [4], "priority": 1, "min_level": 1.0, "max_level": 1.5}
    )
)
```

Setup the users:

```
user_demand = ribasim.UserDemand(
    static=pd.DataFrame(
        data={
            "node_id": [3],
            "priority": [2],
            "demand": [1.5e-3],
            "return_factor": [0.2],
            "min_level": [0.2],
        }
    )
)
```

Setup the allocation:

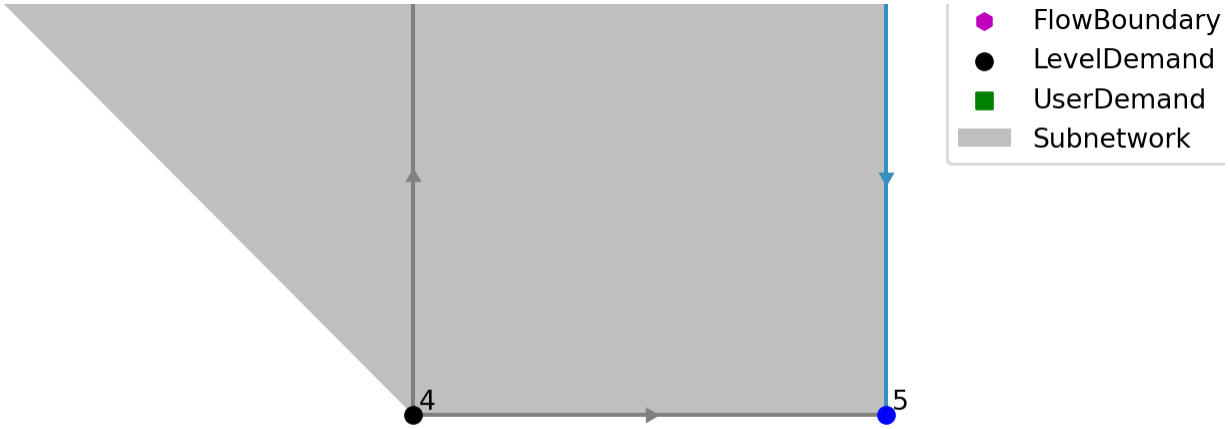
```
allocation = ribasim.Allocation(use_allocation=True, timestep=1e5)
```

Setup a model:

```
model = ribasim.Model(
    network=ribasim.Network(node=node, edge=edge),
    basin=basin,
    flow_boundary=flow_boundary,
    level_demand=level_demand,
    user_demand=user_demand,
    allocation=allocation,
    starttime="2020-01-01 00:00:00",
    endtime="2020-02-01 00:00:00",
)
```

Let's take a look at the model:

```
model.plot()
```



Write the model to a TOML and GeoPackage:

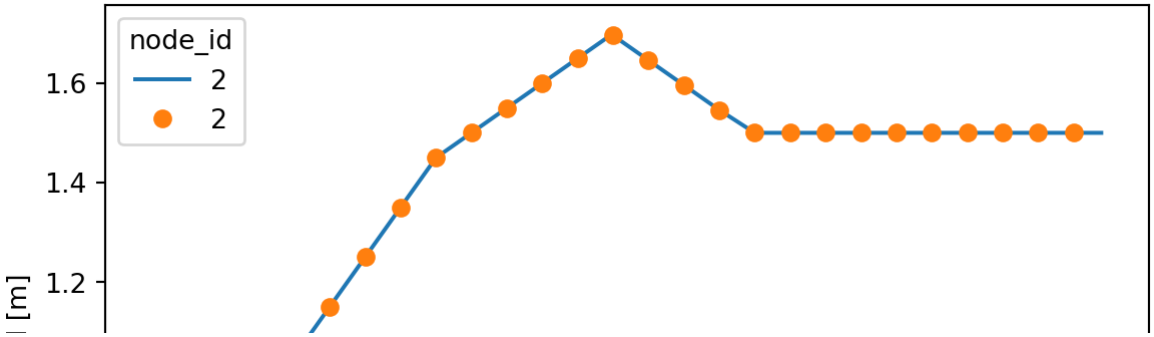
```
model.write(datadir / "level_demand/ribasim.toml")
```

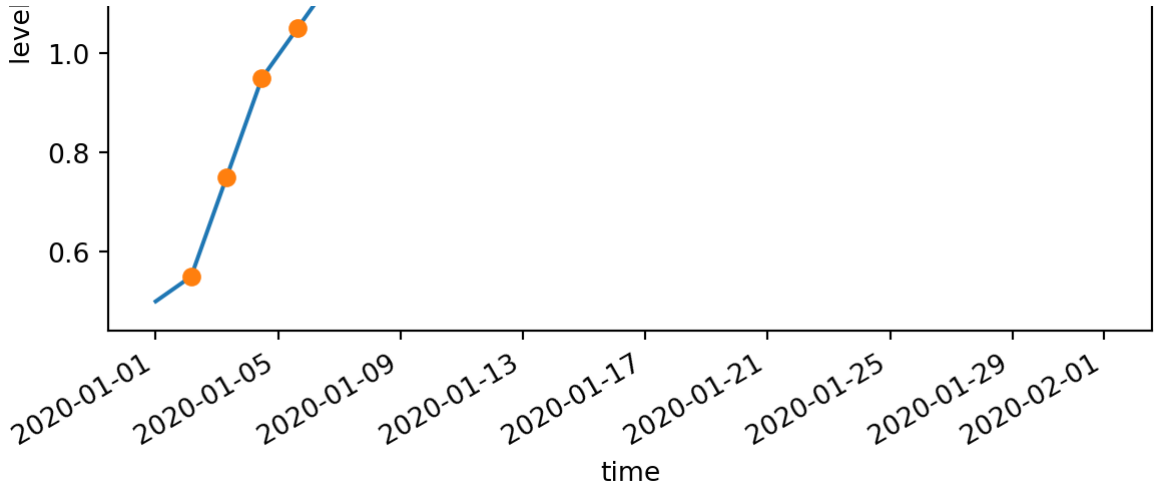
```
PosixPath('data/level_demand/ribasim.toml')
```

Now run the model with `ribasim level_demand/ribasim.toml`. After running the model, read back the results:

```
df_basin = pd.read_feather(datadir / "level_demand/results/basin.arrow")
df_basin = df_basin[df_basin.node_id == 2]
df_basin_wide = df_basin.pivot_table(
    index="time", columns="node_id", values=["storage", "level"]
)
ax = df_basin_wide["level"].plot()
where_allocation = (
    df_basin_wide.index - df_basin_wide.index[0]
).total_seconds() % model.allocation.timestep == 0
where_allocation[0] = False
df_basin_wide[where_allocation]["level"].plot(
    style="o",
    ax=ax,
)
ax.set_ylabel("level [m]")
```

```
Text(0, 0.5, 'level [m]')
```





In the plot above, the line denotes the level of Basin #2 over time and the dots denote the times at which allocation optimization was run, with intervals of Δt_{alloc}. The Basin level is a piecewise linear function of time, with several stages explained below.

Constants: - d: UserDemand #3 demand, - ϕ: Basin #2 precipitation rate, - q: LevelBoundary flow.

Stages: - In the first stage the UserDemand abstracts fully, so the net change of Basin #2 is $q + \phi - d$; - In the second stage the Basin takes precedence so the UserDemand doesn't abstract, hence the net change of Basin #2 is $q + \phi$; - In the third stage (and following stages) the Basin no longer has a positive demand, since precipitation provides enough water to get the Basin to its target level. The FlowBoundary flow gets fully allocated to the UserDemand, hence the net change of Basin #2 is ϕ; - In the fourth stage the Basin enters its surplus stage, even though initially the level is below the maximum level. This is because the simulation anticipates that the current precipitation is going to bring the Basin level over its maximum level. The net change of Basin #2 is now $q + \phi - d$; - At the start of the fifth stage the precipitation stops, and so the UserDemand partly uses surplus water from the Basin to fulfill its demand. The net change of Basin #2 becomes $q - d$. - In the final stage the Basin is in a dynamical equilibrium, since the Basin has no supply so the user abstracts precisely the flow from the LevelBoundary.