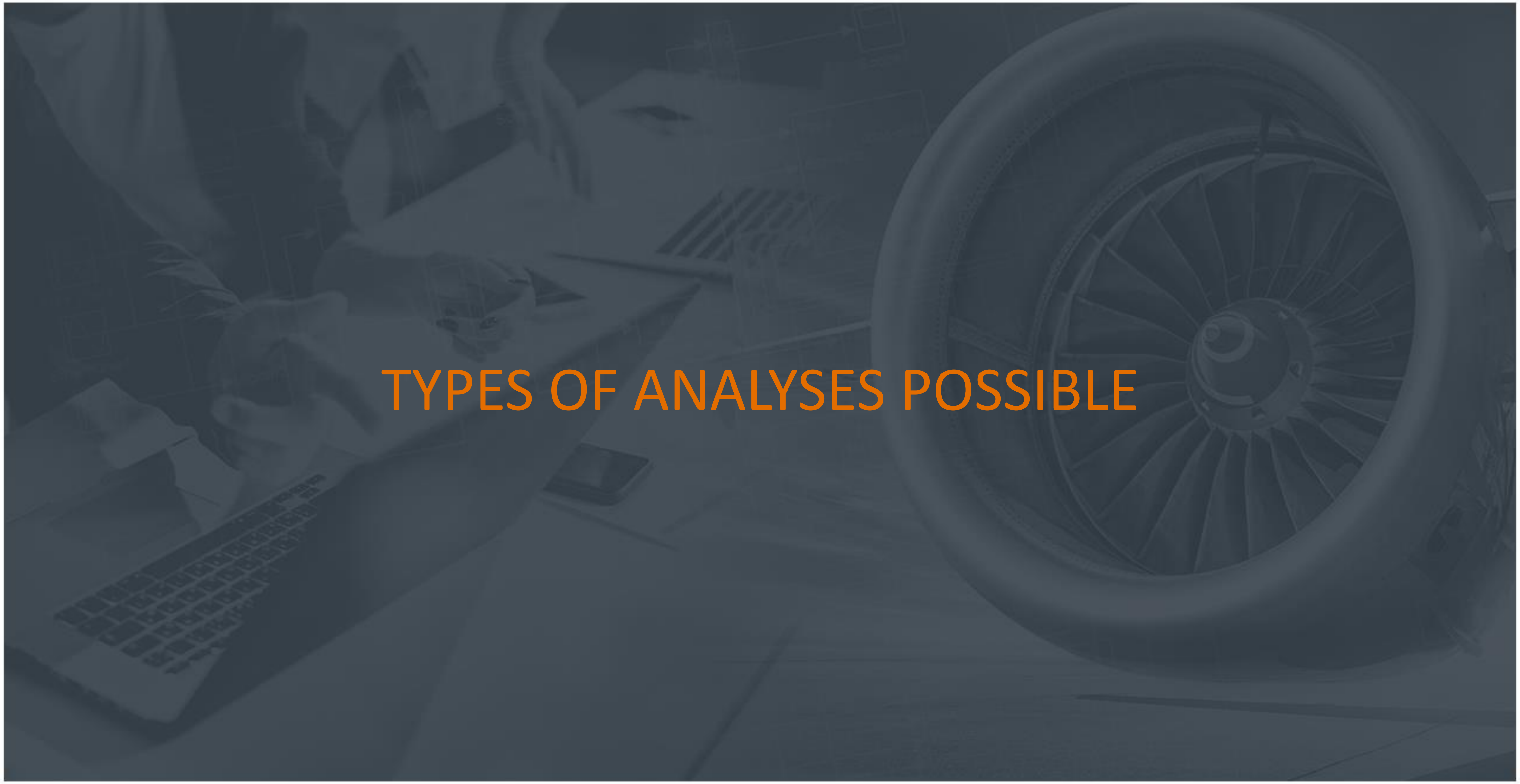
MODEL UTILIZATION

Lecture 2.3

OVERVIEW

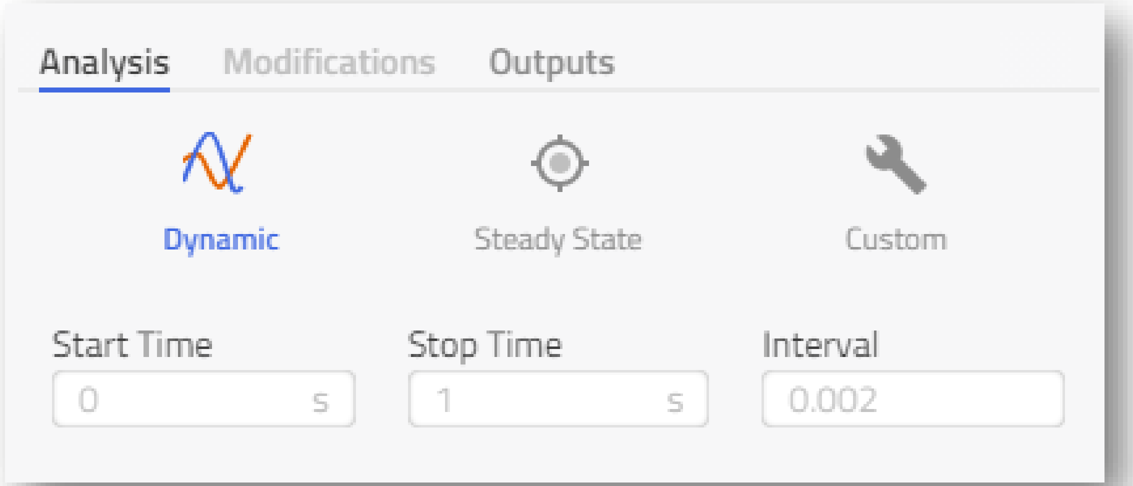
- ✓ Types of analyses available
- ✓ Design of Experiments using Impact
- ✓ Custom views in Impact



TYPES OF ANALYSES POSSIBLE

DIFFERENT SIMULATION MODES IN IMPACT

- Modelon Impact improves model utilization by allowing the same model/code to be used for different types of analyses
- Modelon Impact has 3 simulation modes
  - **Dynamic**: simulation that marches through time
  - **Steady state**: simulation of a single equilibrium point
  - **Custom**: simulation that executes a custom function (Python script)



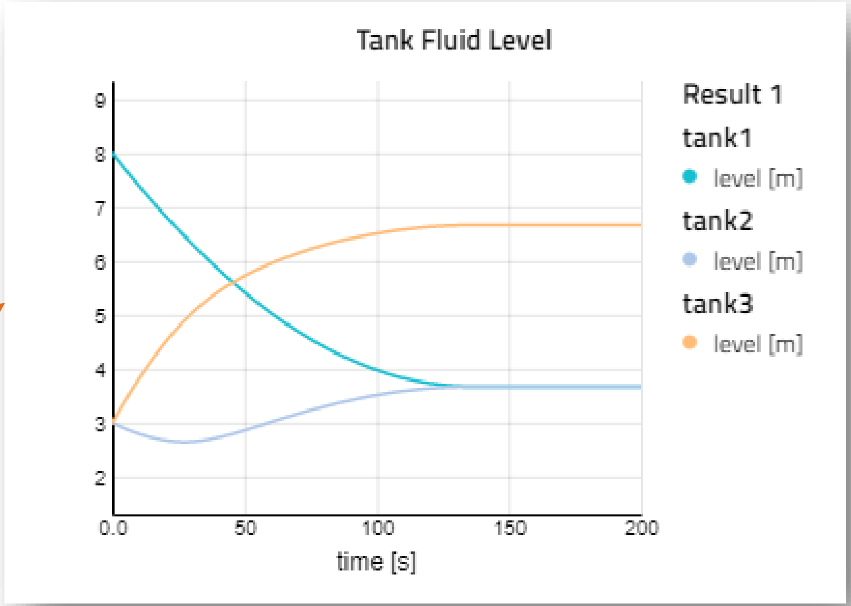
Start Time: 0 s
Stop Time: 1 s
Interval: 0.002

DYNAMIC SIMULATION

- Of the form:

$$\dot{x}(t) = f(x(t), y(t), t),$$
$$0 = g(x(t), y(t), t)$$

- Initial values for the states must be provided (through initialization)
- The state variables are known from the start and evolve along with the governing equations, whether it's an ODE or a continuous DAE.
- Outputs of the simulation are time trajectories



STEADY STATE SIMULATION

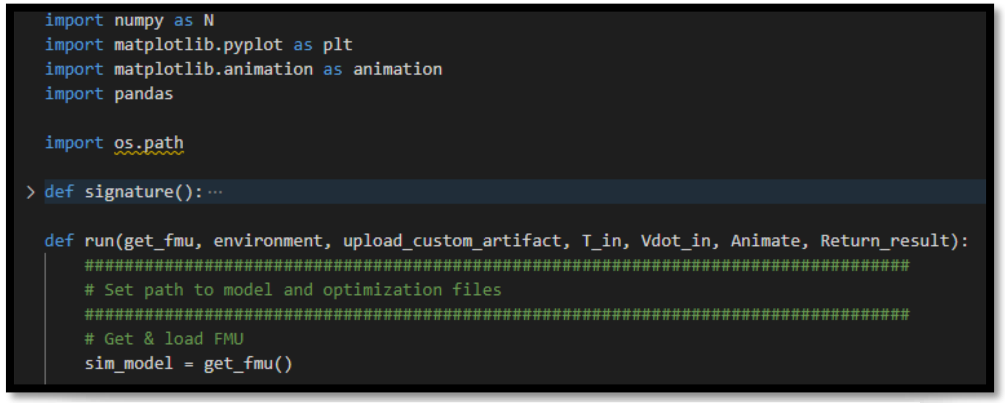
- Of the form:

$$\begin{gathered} f(x, sw) = 0 \\ g(x, sw) \ldots \end{gathered}$$

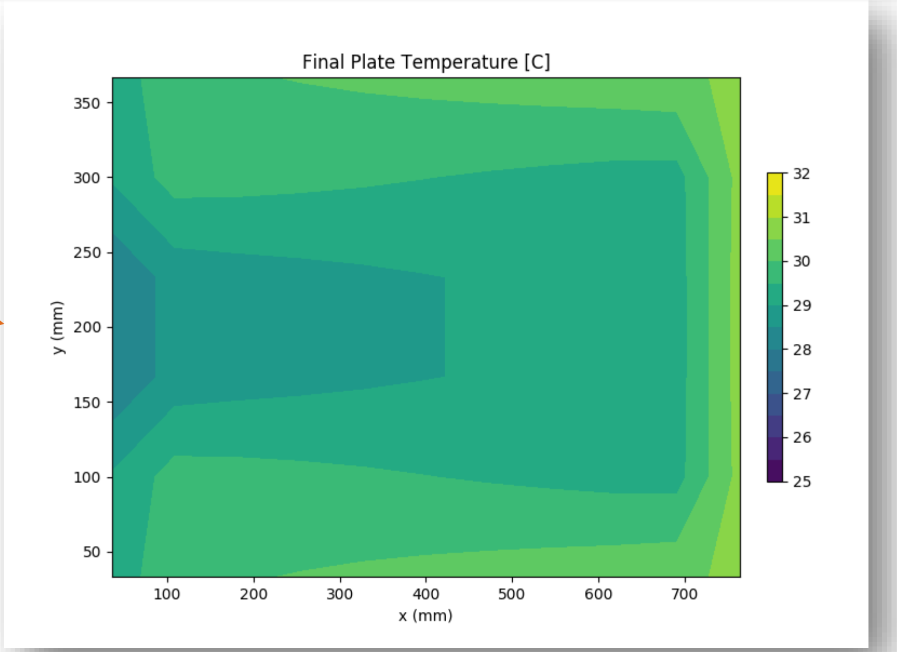
- Require initial guess values
- Algorithm iteratively improves the vector of unknowns
    - Modelon Impact compiler can use “hints” provided in Modelica annotations to identify better iteration variable-residual pairs (Physics based Solving)
- Outputs of the simulation are equilibrium points and not trajectories

CUSTOM SIMULATION

- Executed through a custom function or Python script
- Can be used for *dynamic* or *steady state* analysis
- Outputs of the simulation can be customized:
  - Selected variables
  - Custom plots or animations
  - Reports
  - …
- Covered in advanced scripting module



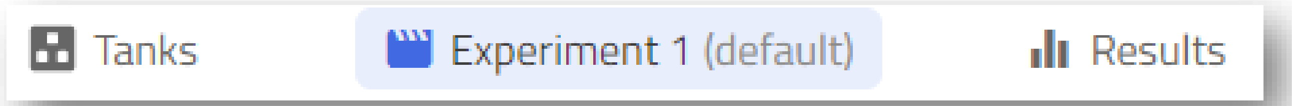
```
import numpy as N
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import pandas

import os.path

def signature():⋯

def run(get_fmu, environment, upload_custom_artifact, T_in, Vdot_in, Animate, Return_result):
    ##################################################################################
    # Set path to model and optimization files
    ##################################################################################
    # Get & load FMU
    sim_model = get_fmu()
```



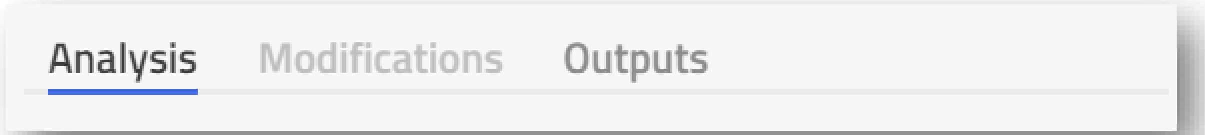
DESIGN OF EXPERIMENTS USING IMPACT

EXPERIMENT MODE

- Second mode available through the mode-selector toolbar

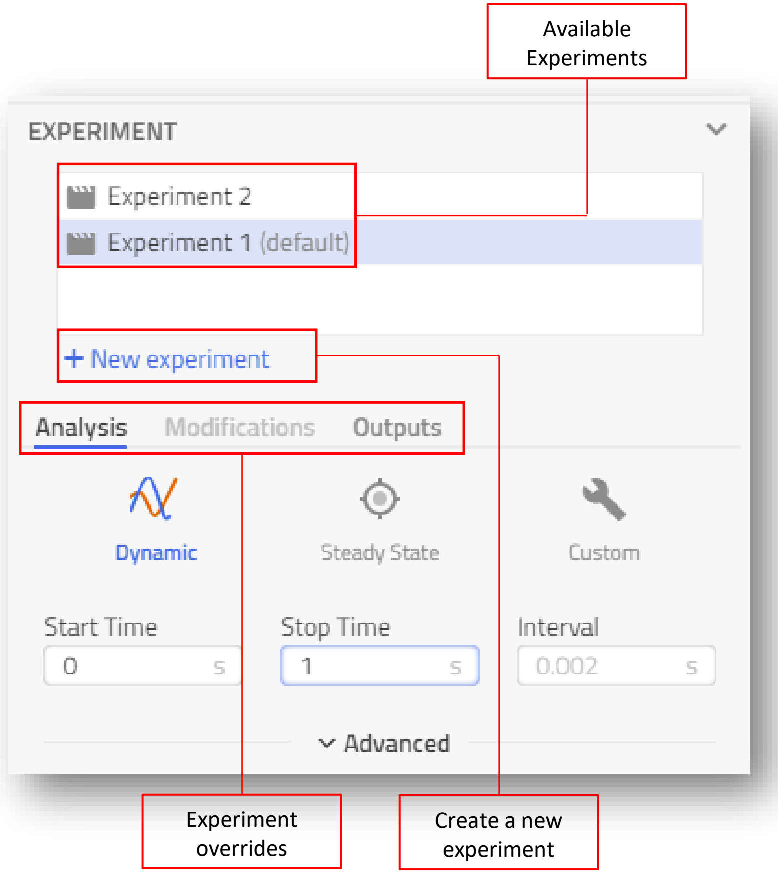


- Three sub tabs to provide overrides to model-defaults:
  - ***Analysis*** : Define/view simulation settings
  - ***Modifications*** : View parameters/inputs with value overrides/changes
  - ***Outputs*** : Create/view filters for variables to write to result file



- **Note**: modifications to parameters made in **Experiment** mode in Impact do not appear as experimental modifiers in the Modelica code

EXPERIMENT MODE



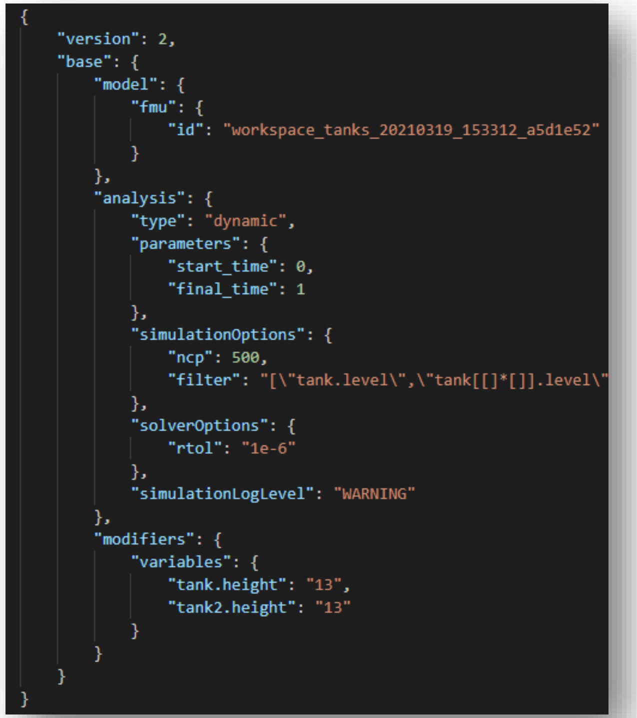
Start Time 0 s

Stop Time 1 s

Interval 0.002 s

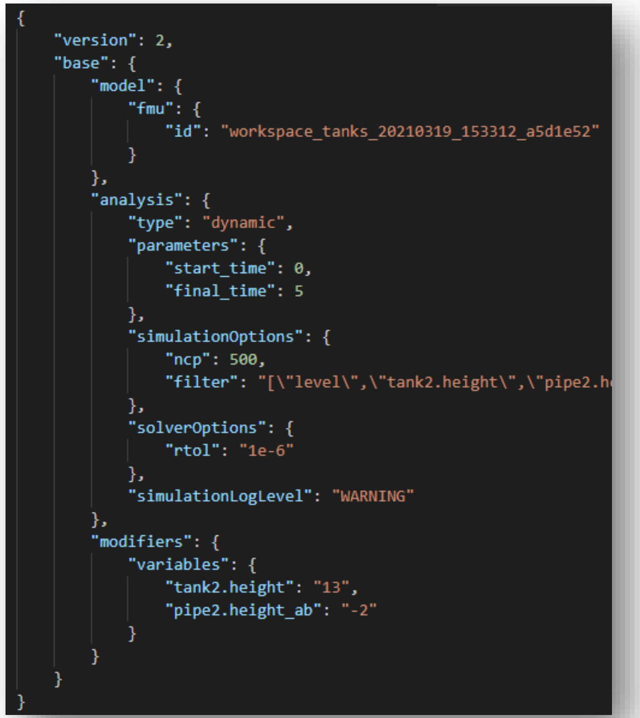
- Experiment settings available in the **Experiment** tab of the *Details Panel*
- Multiple experiments can be created, each with its own overrides for simulation settings, parameters, inputs and outputs. For example:

Experiment 1



```
{
    "version": 2,
    "base": {
        "model": {
            "fmu": {
                "id": "workspace_tanks_20210319_153312_a5d1e52"
            }
        },
        "analysis": {
            "type": "dynamic",
            "parameters": {
                "start_time": 0,
                "final_time": 1
            },
            "simulationOptions": {
                "ncp": 500,
                "filter": "[\"tank.level\",\"tank[[]*[]].level\"
            },
            "solverOptions": {
                "rtol": "1e-6"
            },
            "simulationLogLevel": "WARNING"
        },
        "modifiers": {
            "variables": {
                "tank.height": "13",
                "tank2.height": "13"
            }
        }
    }
}
```

Experiment 2

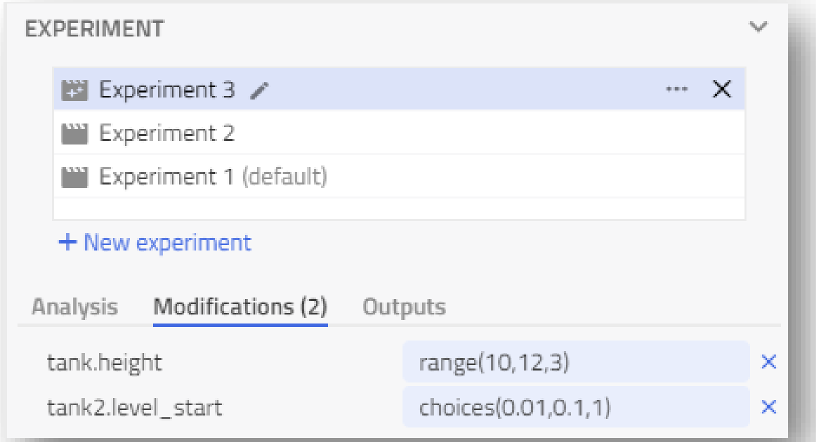


```
{
    "version": 2,
    "base": {
        "model": {
            "fmu": {
                "id": "workspace_tanks_20210319_153312_a5d1e52"
            }
        },
        "analysis": {
            "type": "dynamic",
            "parameters": {
                "start_time": 0,
                "final_time": 5
            },
            "simulationOptions": {
                "ncp": 500,
                "filter": "[\"level\",\"tank2.height\",\"pipe2.h
            },
            "solverOptions": {
                "rtol": "1e-6"
            },
            "simulationLogLevel": "WARNING"
        },
        "modifiers": {
            "variables": {
                "tank2.height": "13",
                "pipe2.height_ab": "-2"
            }
        }
    }
}
```

DESIGN OF EXPERIMENTS

- We already saw how experiments in Impact can contain parameter/input modifications
- Impact also provides two operators for parameter sweeps: *range()* and *choices()*
- Usage:
  - *range(start_value, end_value, no_of_steps)*
  - *choices(choice1, choice2, …, choiceN)*
- Currently only support Full Factorial DoE from within the gui.

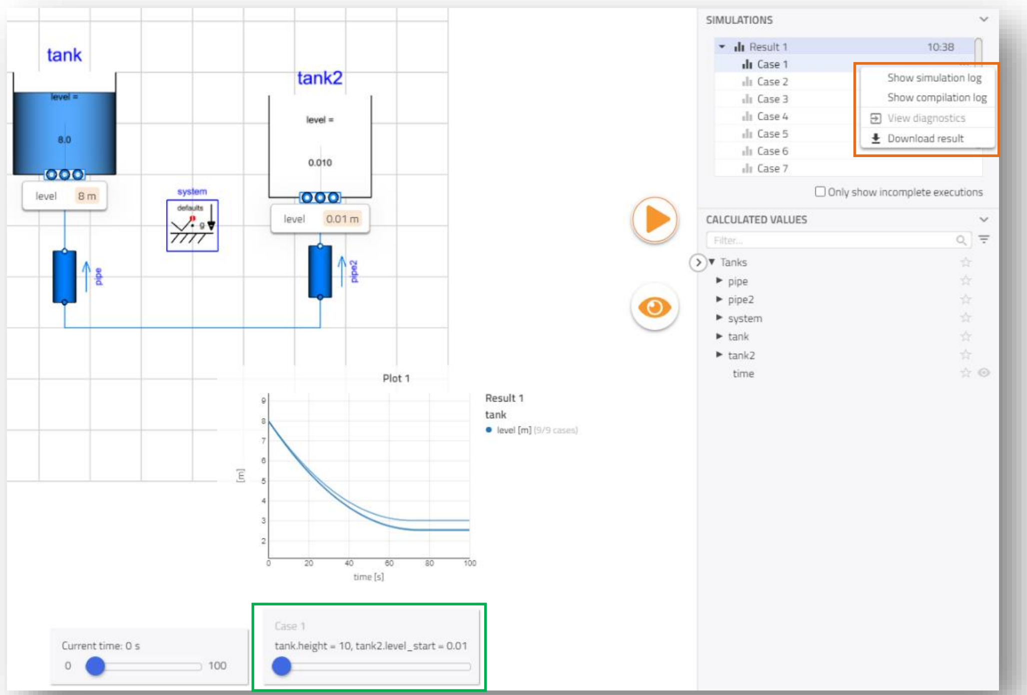
Beyond a simple DoE, Impact provides a rest API framework for **multi-execution**. This is covered later in the course.



tank.height	range(10,12,3)
tank2.level_start	choices(0.01,0.1,1)

DESIGN OF EXPERIMENTS

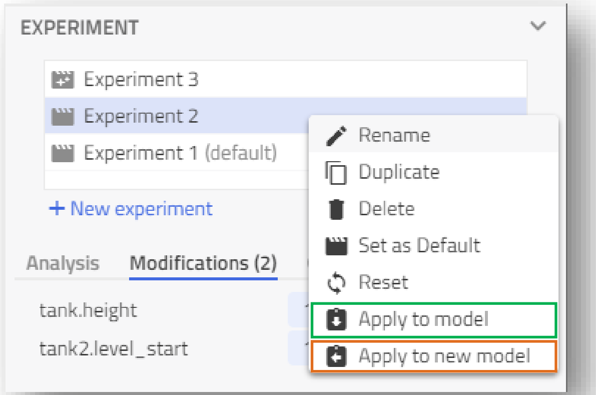
- Results mode:
  - An extra slider for selecting the case is available
  - Result files and variables (ones meeting the output filter criteria) for each case is available to view and download



tank.height = 10, tank2.level_start = 0.01

USING RESULTS FROM EXPERIMENTS

- Several experiments can be executed to find an optimal combination of parameters
- Once an optimal set of values has been found, they can:
  - Be applied to the model to replace the defaults
  - Be applied to a duplicate of the model
- This writes the modifications defined in the Experiment, back into the modelica code.



Note: This does not work for DoE experiment results using *range()* or *choice()* today

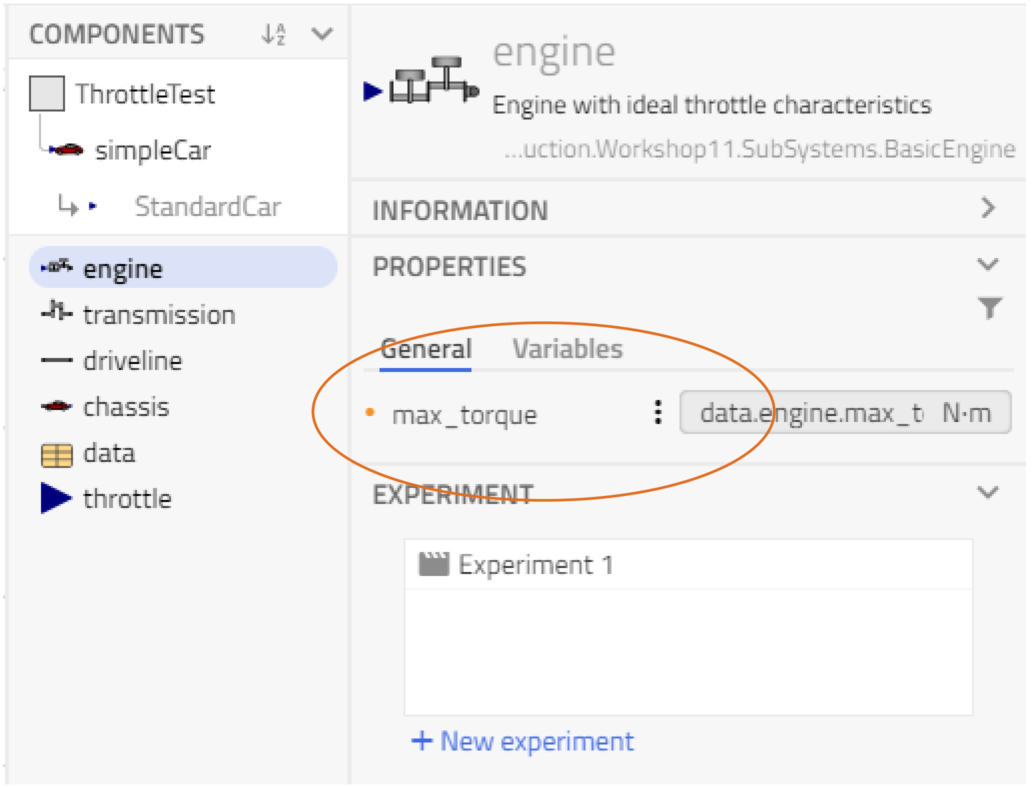
STRUCTURAL PARAMETERS

- Some parameters cannot be changed after compilation
- These are denoted by yellow dot

- Sometimes this can be remedied by:
  - Use the annotation Evaluate = false

```
parameter Real ratio=1.0
  "Transmission ratio (flange_a.phi/flange_b.phi)" annotation( Evaluate = false );
```

  - This is only possible if the parameter is evaluated, and is not structural


COMPONENTS

ThrottleTest
simpleCar
StandardCar
engine
transmission
driveline
chassis
data
throttle

engine
Engine with ideal throttle characteristics
...uction.Workshop11.SubSystems.BasicEngine

INFORMATION
PROPERTIES
General Variables

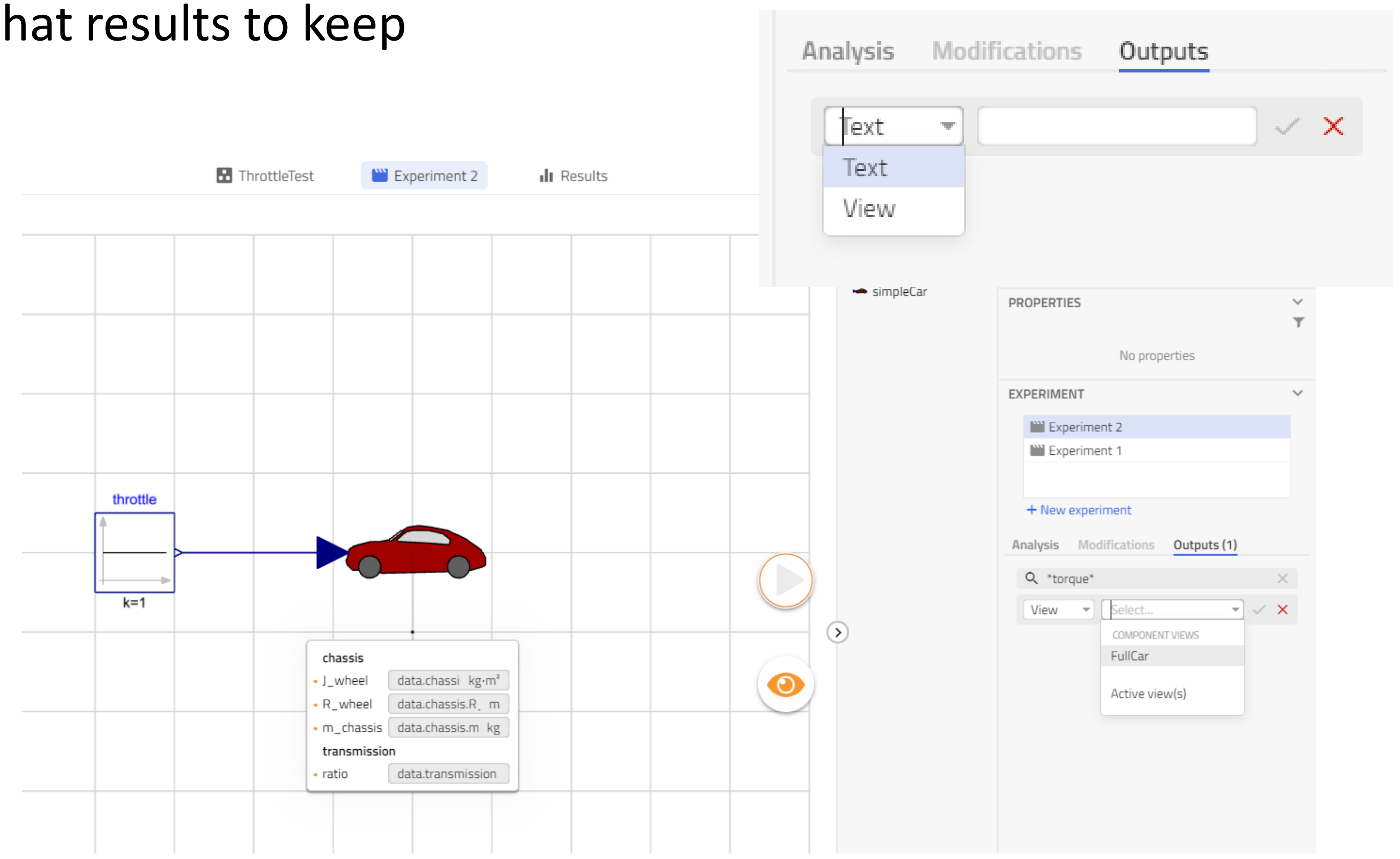
max_torque	data.engine.max_t	N·m

EXPERIMENT
Experiment 1
+ New experiment

Modelon

RELEVANT OUTPUT

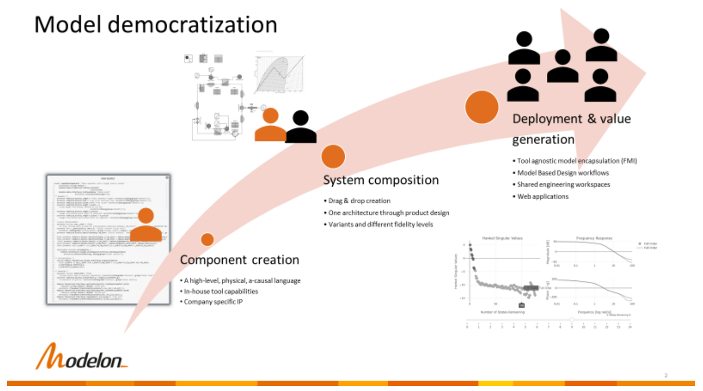
- If you are running DoE experiments, you can quickly generate a lot of output data
- Use “Outputs” filters, to define what results to keep
- Possible to add several
- Text: filter by string
    - Supports glob patterns
- View: choose available views

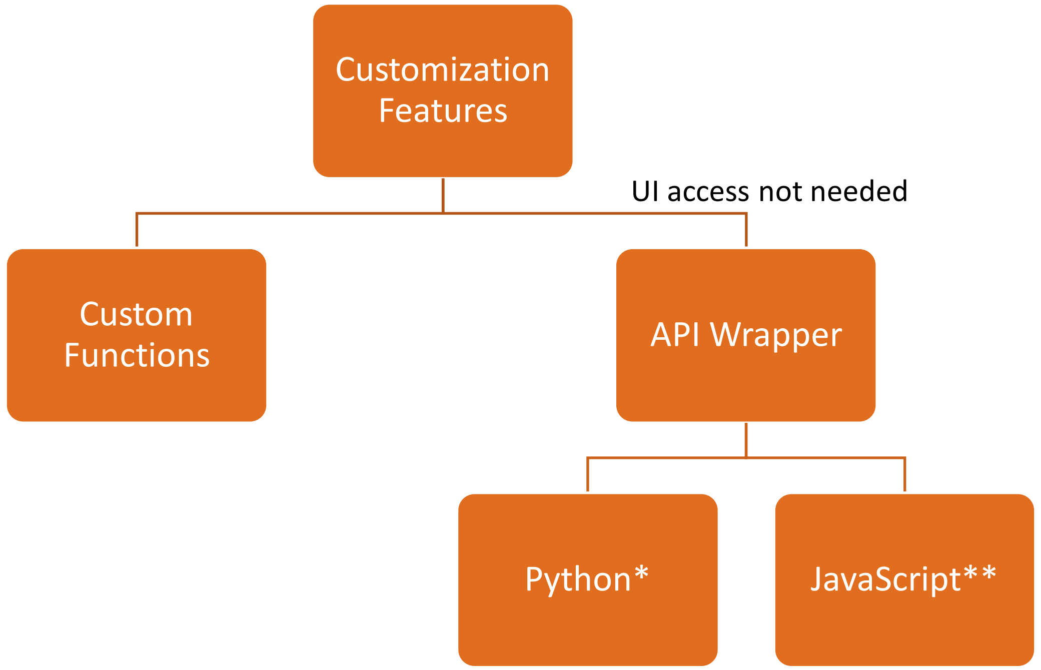


CUSTOM VIEWS IN IMPACT

MODELON IMPACT API

- Modelon Impact is designed to maximize model value generation
- Impact has a REST API that allows customization and lets non-expert users in on the fun


Model democratization
Deployment & value generation
Tool agnostic model encapsulation (FMI)
Model Based Design workflows
Shared engineering workspaces
Web applications
System composition
Drag & drop creation
One architecture through product design
Variants and different fidelity levels
Component creation
A high-level, physical, a-causal language
In-house tool capabilities
Company specific IP
Modelon



* Used for notebooks
** Used for Webapps

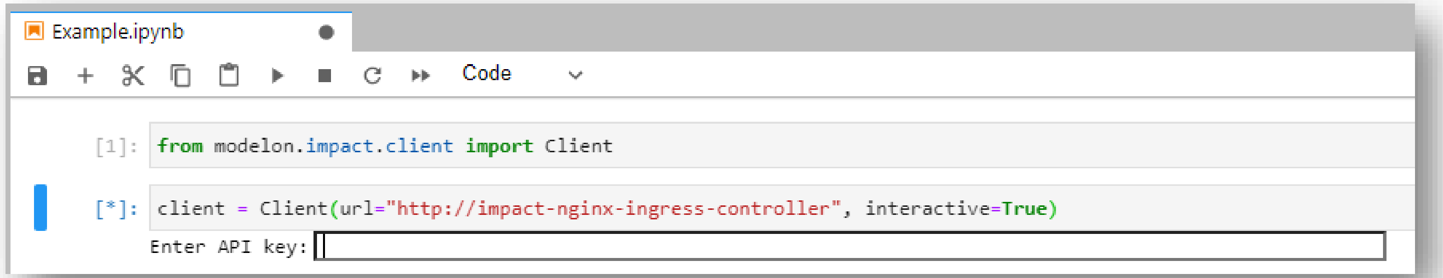
NOTEBOOK VIEW

- Uses the Python wrapper for Modelon Impact API

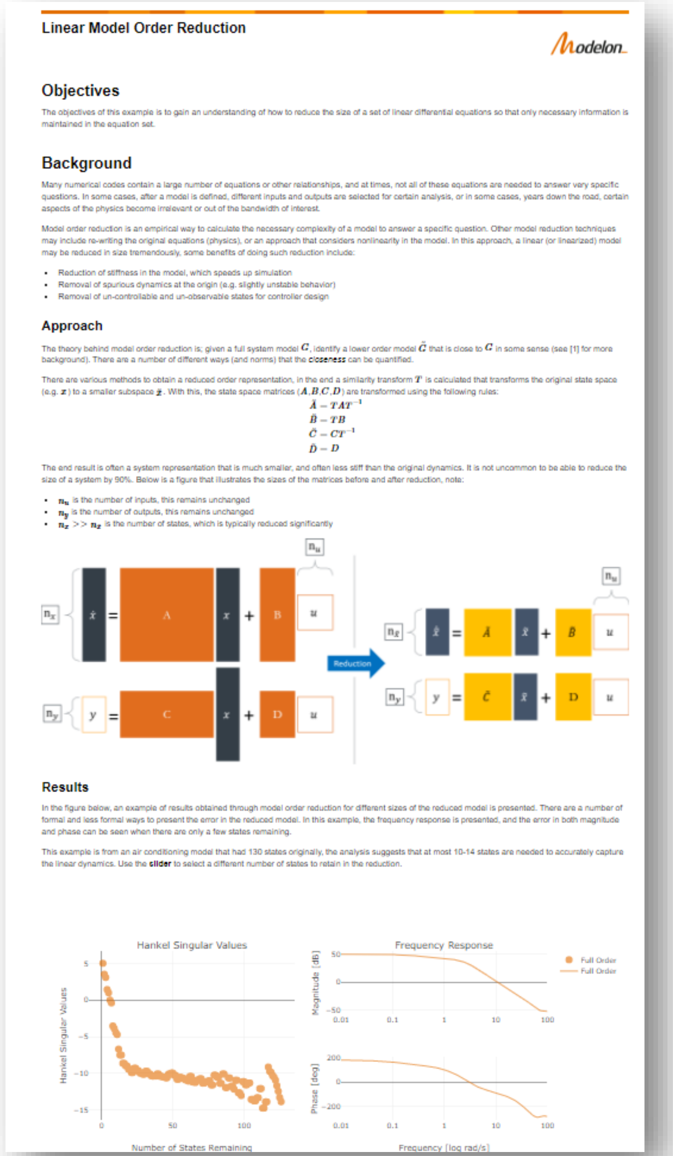
  Open-source, installation with pip:

  ```
  pip install modelon-impact-client
  ```

- Useful for:
  - Automating workflows
  - Creating interactive documentation and reports
  - Leveraging other packages and tools available in Python (for DoE, ML, Optimization etc.)
- Covered in advanced scripting training module



```
[1]: from modelon.impact.client import Client

[*]: client = Client(url="http://impact-nginx-ingress-controller", interactive=True)
     Enter API key:
```



Linear Model Order Reduction

Objectives

The objectives of this example is to gain an understanding of how to reduce the size of a set of linear differential equations so that only necessary information is maintained in the equation set.

Background

Many numerical codes contain a large number of equations or other relationships, and at times, not all of these equations are needed to answer very specific questions. In some cases, after a model is defined, different inputs and outputs are selected for certain analysis, or in some cases, years down the road, certain aspects of the physics become irrelevant or out of the bandwidth of interest.

Model order reduction is an empirical way to calculate the necessary complexity of a model to answer a specific question. Other model reduction techniques may include re-writing the original equations (physics), or an approach that considers nonlinearity in the model. In this approach, a linear (or linearized) model may be reduced in size tremendously, some benefits of doing such reduction include:

- Reduction of stiffness in the model, which speeds up simulation
- Removal of spurious dynamics at the origin (e.g. slightly unstable behavior)
- Removal of un-controllable and un-observable states for controller design

Approach

The theory behind model order reduction is; given a full system model G, identify a lower order model $\breve{G}$ that is close to G in some sense (see [1] for more background). There are a number of different ways (and norms) that the **closeness** can be quantified.

There are various methods to obtain a reduced order representation, in the end a similarity transform T is calculated that transforms the original state space (e.g. x) to a smaller subspace $\breve{x}$. With this, the state space matrices (A,B,C,D) are transformed using the following rules:

$$\breve{A} = TAT^{-1}$$
$$\breve{B} = TB$$
$$\breve{C} = CT^{-1}$$
$$\breve{D} = D$$

The end result is often a system representation that is much smaller, and often less stiff than the original dynamics. It is not uncommon to be able to reduce the size of a system by 90%. Below is a figure that illustrates the sizes of the matrices before and after reduction, note:

- n_u is the number of inputs, this remains unchanged
- n_y is the number of outputs, this remains unchanged
- $n_x >> n_{\breve{x}}$ is the number of states, which is typically reduced significantly

Results

In the figure below, an example of results obtained through model order reduction for different sizes of the reduced model is presented. There are a number of formal and less formal ways to present the error in the reduced model. In this example, the frequency response is presented, and the error in both magnitude and phase can be seen when there are only a few states remaining.

This example is from an air conditioning model that had 130 states originally, the analysis suggests that at most 10-14 states are needed to accurately capture the linear dynamics. Use the **slider** to select a different number of states to retain in the reduction.

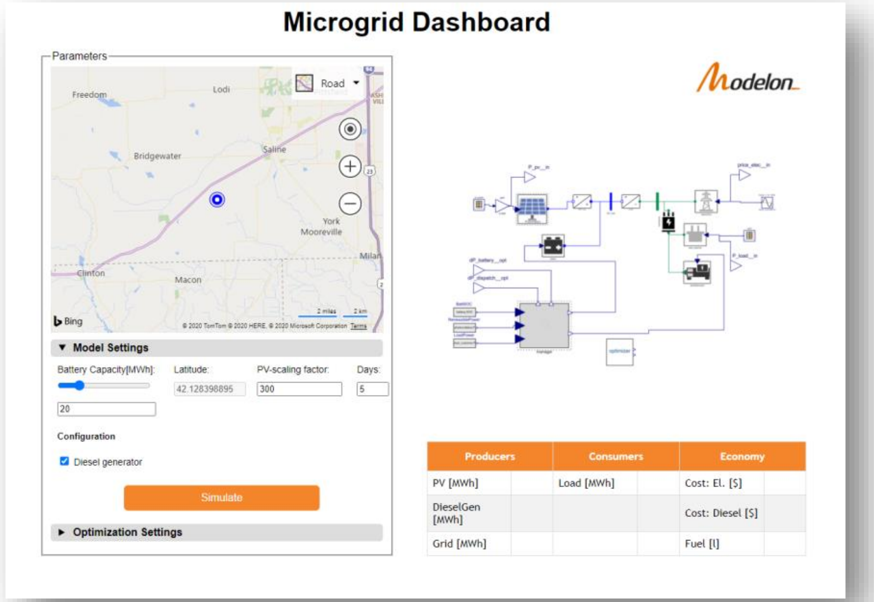
DASHBOARD VIEW

- Uses the JavaScript wrapper for Modelon Impact API

  Open-source installation with npm

  ```
  npm install modelon-client-js
  ```

- Web apps allow:
  - simplified and customized views of models
  - custom post-processing (plots, reports etc.)
  - use of external web resources, APIs or services (maps ,weather etc.)
  - workflows that are not possible directly through the UI [example: plot with 2 y-axes]



▼ Model Settings

Battery Capacity[MWh]: 20

Latitude: 42.128398895

PV-scaling factor: 300

Days: 5

Configuration

✓ Diesel generator

▶ Optimization Settings

Producers		Consumers		Economy	
PV [MWh]		Load [MWh]		Cost: El. [$]	
DieselGen [MWh]				Cost: Diesel [$]	
Grid [MWh]				Fuel [l]	

WORKSHOP 2.3

In this workshop you will:

- Exercise advanced workflows
- Export a model as an FMU
- Run a multistage experiment