

Python Client Multi-Execution and Advanced Plotting

Presented by Modelon

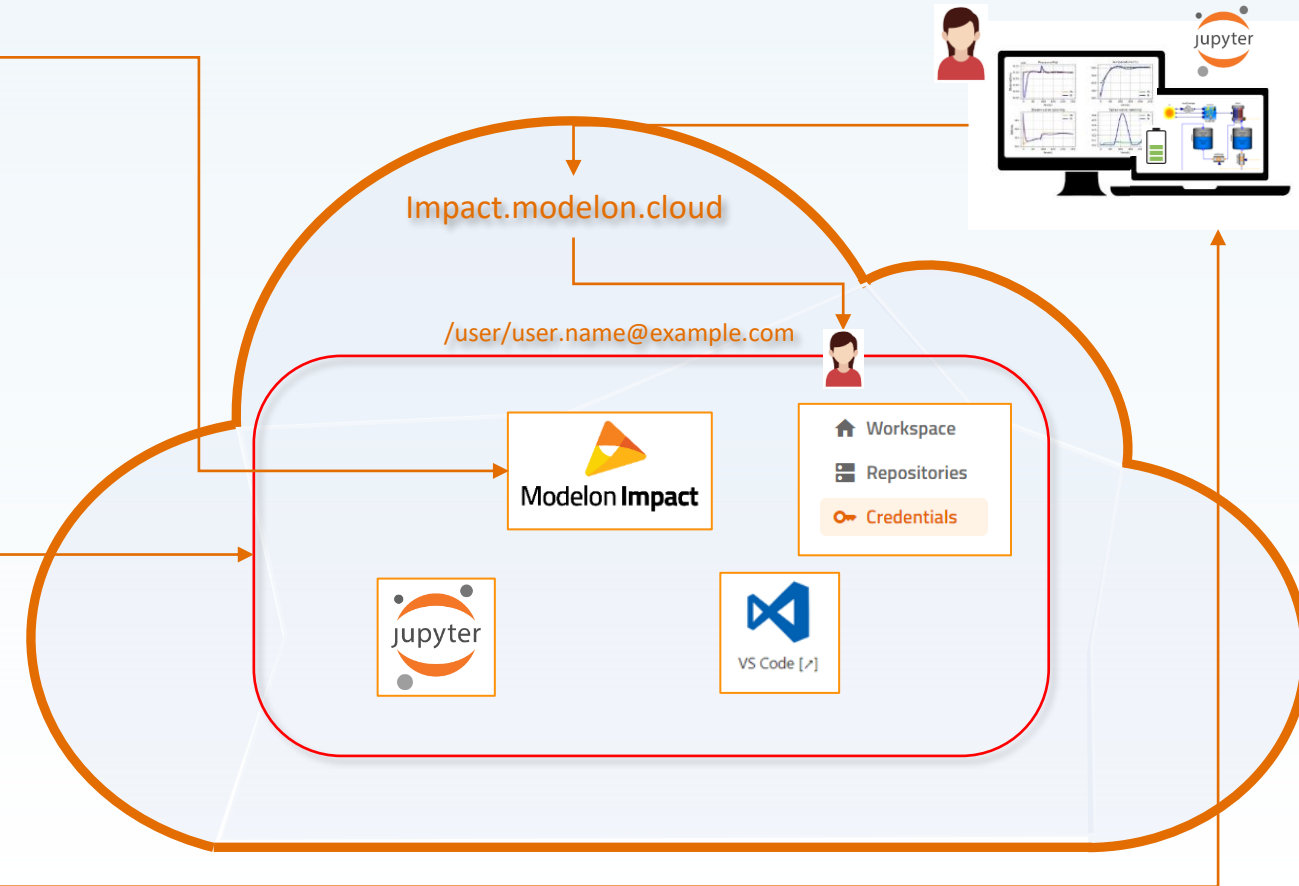


Outline

- Python context
- Inspirational examples:
 - Advanced Post-Processing with Python 3rd party packages
 - Modelon Impact's Multi-execution framework
 - Sharing of simple and advanced notebooks for you to explore

Modelon Impact and Python

- Modelon Impact:
 - Custom Functions
 - OCT
- Modelon Impact Cloud:
 - Workflow Automation/Notebooks
 - OCT
 - Modelon Impact Python Client
- Anywhere:
 - Modelon Impact Python Client



Relevant packages

```
# Modelica Compilation and Simulation
import pyfmi
import pymodelica
import oct
import pyjmi

# Modelon Impact Client
import modelon.impact.client

# Post processing and data visualization tools
import matplotlib
import plotly
import dash

# Data handling
import pandas

# Numerical linear algebra and scientific computations
import numpy
import scipy
```

Modelon Impact Compiler API

PyModelica

- Interface to the compiler
- Compiles Modelica code into FMUs

PyFMI

- Used for FMU interaction
- Standalone
- Open source
- Dynamic simulation, using Assimulo solvers
- Steady-state simulation when combined with OCT

PyJMI

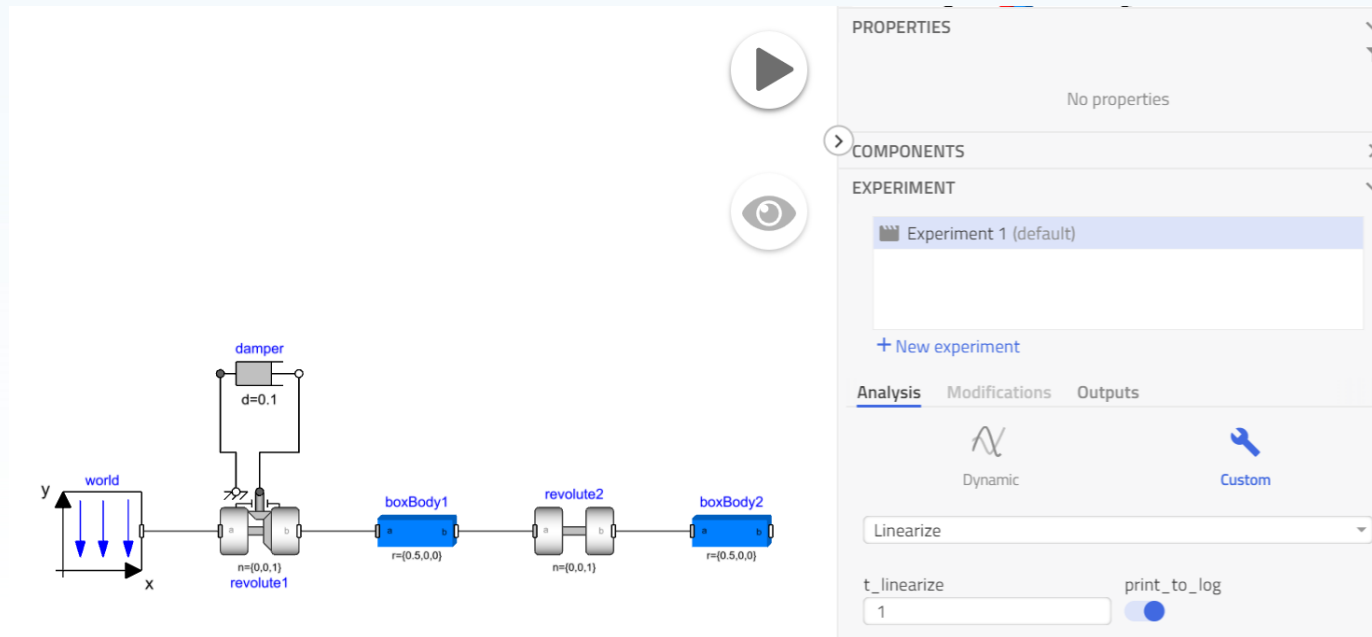
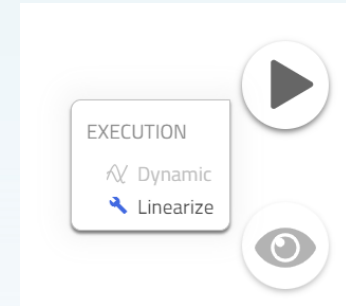
- Used to transfer Modelica code into CasADi
- Does not use FMUs
- Dynamic optimization with local collocation

OCT

- Steady-state simulation and diagnostics
- Dynamic diagnostics

Modelon Impact Compiler Python API

- Create Custom Functions – Embedded in UI
 - Can be called from Impact Python Client



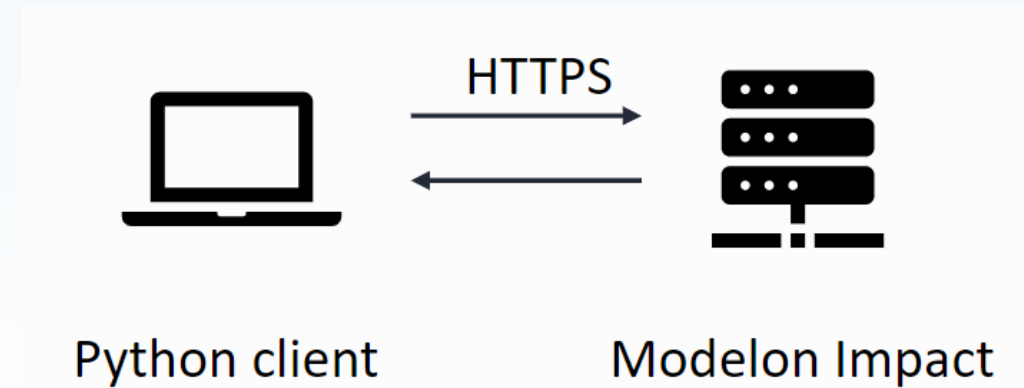
The Modelon Impact Python Client

Wraps the Modelon Impact API and helps with:

- Authentication
- Compiling models
- Downloading FMUs
- Defining and executing Simulations
- Fetching Results

Furthermore:

- Developed in sync with The Modelon Impact API



Plotly

- Creates advanced plots efficiently
- Focus on interactive graphs
- Integrates well with Pandas DataFrames
- [Free and open source](https://plotly.com/python/)
- Maintained by Plotly

<https://plotly.com/python/>

Plotly Open Source Graphing Library for Python

Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts.

Plotly.py is [free and open source](#) and you can [view the source](#), [report issues](#) or [contribute](#) on GitHub.

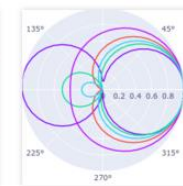
Deploy Python AI Dash apps on private Kubernetes clusters: [Pricing](#) | [Demo](#) | [Overview](#) | [AI App Services](#)

Fundamentals

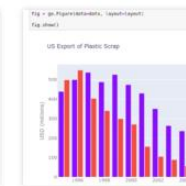
[More Fundamentals »](#)



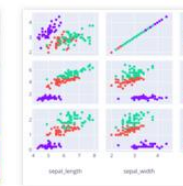
The Figure Data Structure



Creating and Updating Figures



Displaying Figures



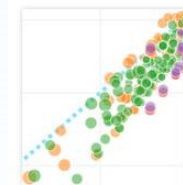
Plotly Express



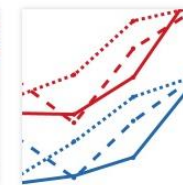
Analytical Apps with Dash

Basic Charts

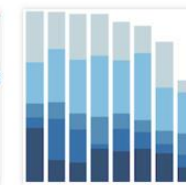
[More Basic Charts »](#)



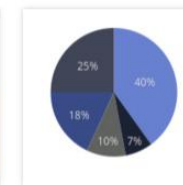
Scatter Plots



Line Charts



Bar Charts



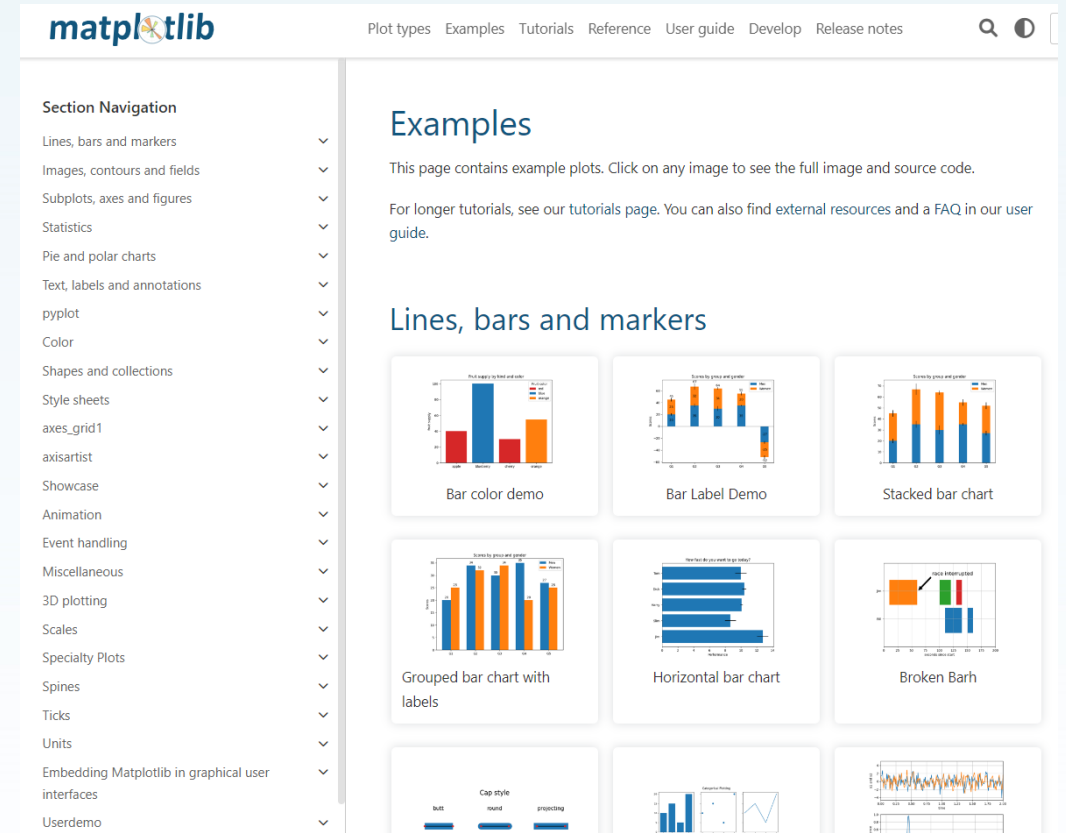
Pie Charts



Bubble Charts

Matplotlib

- Community project maintained for and by its users
- Quick and straightforward to get basic plots
- By default static plots (although interactivity is achievable)
- API designed to resemble plotting API of MATLAB

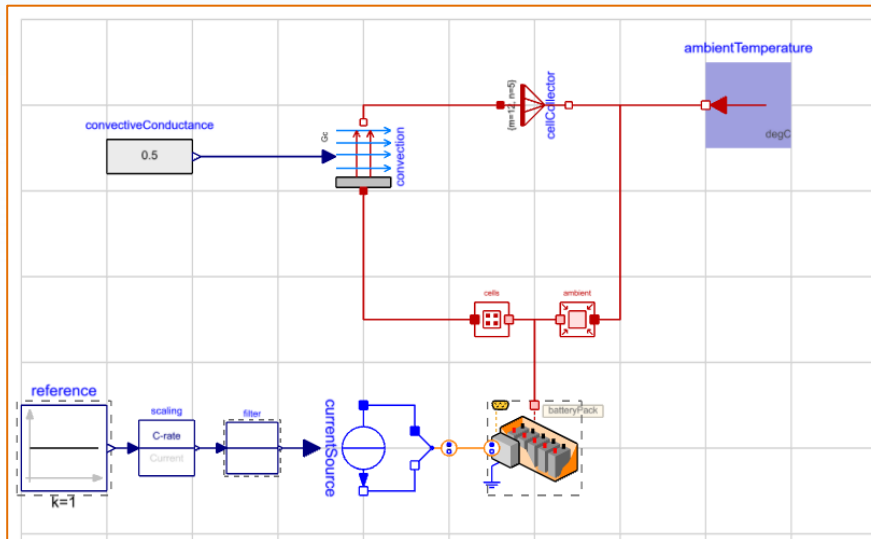
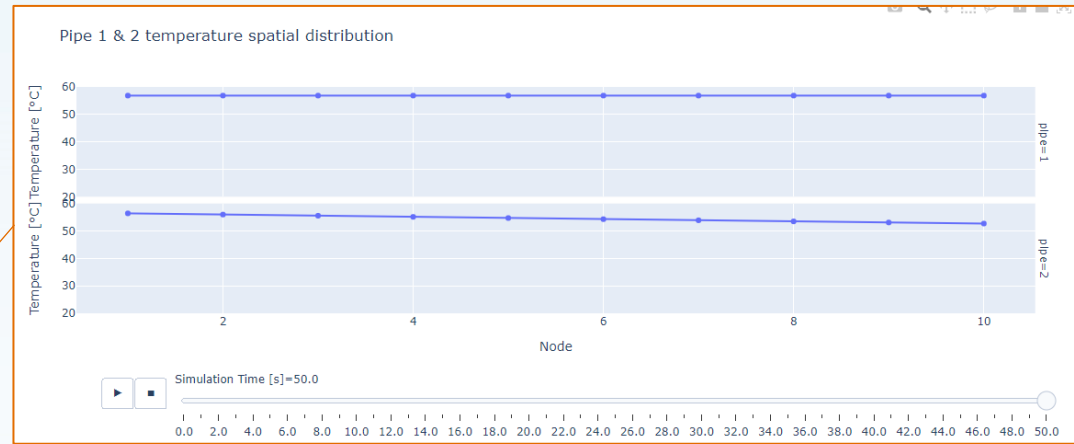
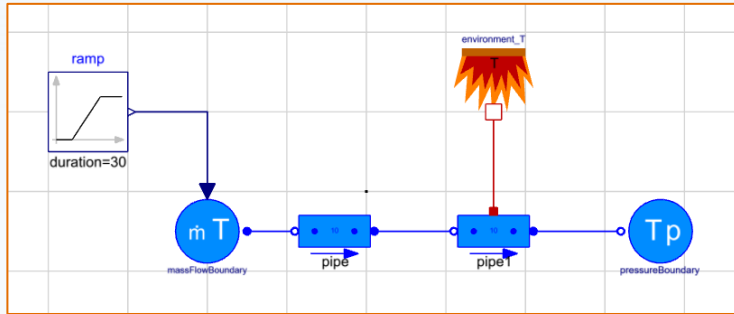


<https://matplotlib.org/stable/gallery/index.html>

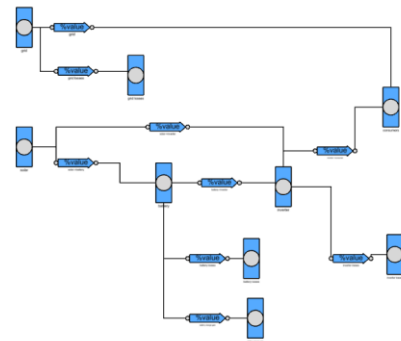
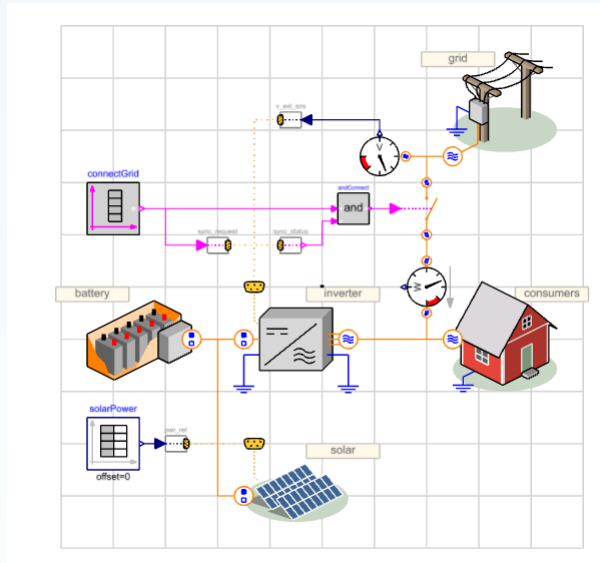
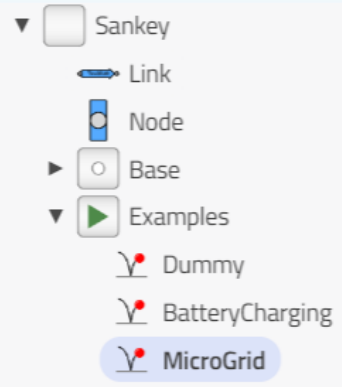
How to run the Demo examples

- Import the relevant workspace from Modelon Impact landing page.
- From the workspace, open JupyterHub.
- Find and open the corresponding notebook found in the filebrowser.
- More information can be found inside the notebooks.

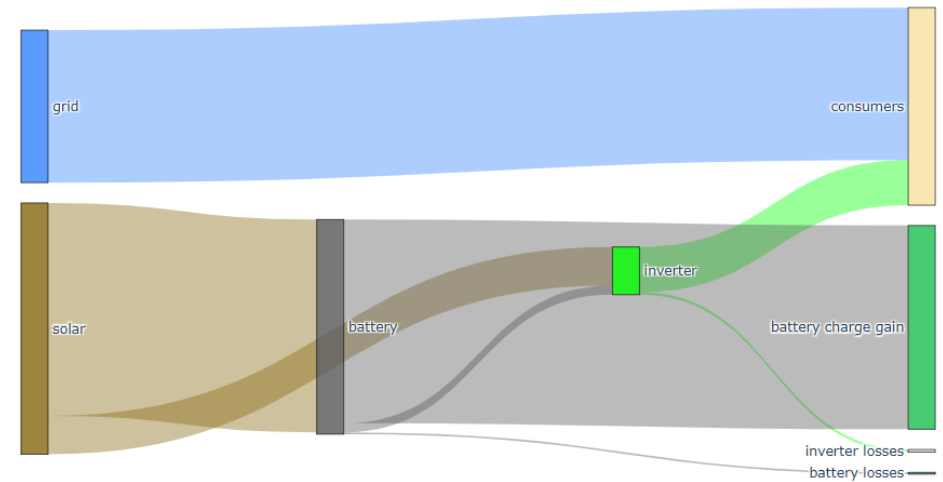
Demo 1: Spatial plotting



Demo 2: Sankey Diagram



Sankey.Examples.MicroGrid



Demo 3:

- Battery Monte-Carlo
- Client
- Multi-execution

Plot results

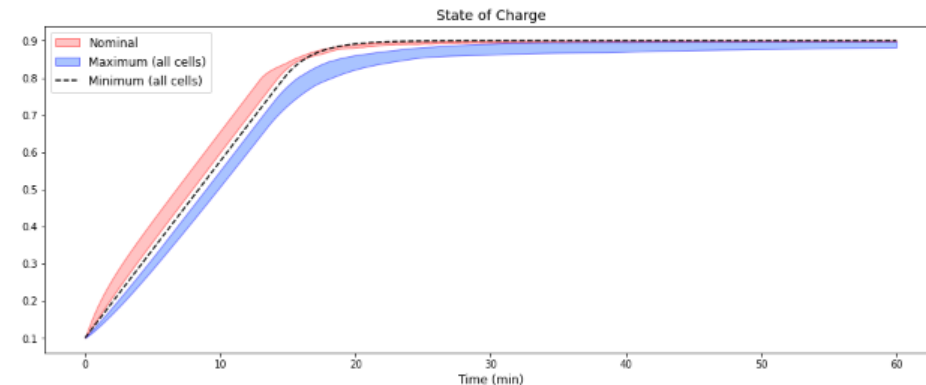
Instead of plotting the trajectories for all of the simulations, we plot an area that show the bounds for each variable over all simulations.

When we look at the SoC, we can see that there is some variation in the results. And we especially note that the minimum SoC no longer reach all the way up to the 90 % target

We also see a spread in the results when we look at the maximum and minimum cell currents. We conclude that there are cases that are even worse than the 80 A we saw in th

```
# Plot
plt.figure(51,figsize=(16, 6))
plt.fill_between(timeMin,socMaxMinM, socMaxMaxM,
                alpha=0.5, edgecolor='#FF0000', facecolor='#FF8888',linewidth=1, antialiased=True)
plt.fill_between(timeMin,socMinMinM, socMinMaxM,
                alpha=0.5, edgecolor='#0000FF', facecolor='#5588FF',linewidth=1, antialiased=True)
plt.plot(resiTimeMin, resi['battery.summary.Soc'],'k--')
plt.title('State of Charge', fontsize=14)
plt.legend(['Nominal',
            'Maximum (all cells)',
            'Minimum (all cells)'], fontsize=12)
plt.xlabel('Time (min)', fontsize=12)
plt.show()
plt.close(51)

plt.figure(52,figsize=(16, 6))
plt.fill_between(timeMin,iMinMinM, iMinMaxM,
                alpha=0.5, edgecolor='#0000FF', facecolor='#5588FF',linewidth=1, antialiased=True)
plt.fill_between(timeMin,iMaxMinM, iMaxMaxM,
                alpha=0.5, edgecolor='#FF0000', facecolor='#FF8888',linewidth=1, antialiased=True)
plt.plot(resiTimeMin, resi['battery.summary.i_cell_max'],'k--')
plt.title('Cell current', fontsize=14)
plt.legend(['Nominal',
            'Maximum (all cells)',
            'Minimum (all cells)'], fontsize=12)
plt.xlabel('Time (min)', fontsize=12)
plt.grid()
plt.show()
plt.close(52)
```



Multi-execution with MI Python Client

- Declarative
- Works against workspace setup
- Easy to get started with built in operators
 - Range(), Choices()
- Full flexibility with extensions
 - Unique solver settings, analysis parameters for every case
- Efficient FMU reuse
- Aggregates results in Cases

```
from modelon.impact.client import Choices

experiment_definition = experiment_definition.with_modifiers({'PI.k': Choices(10, 20, 30, 40)})
```

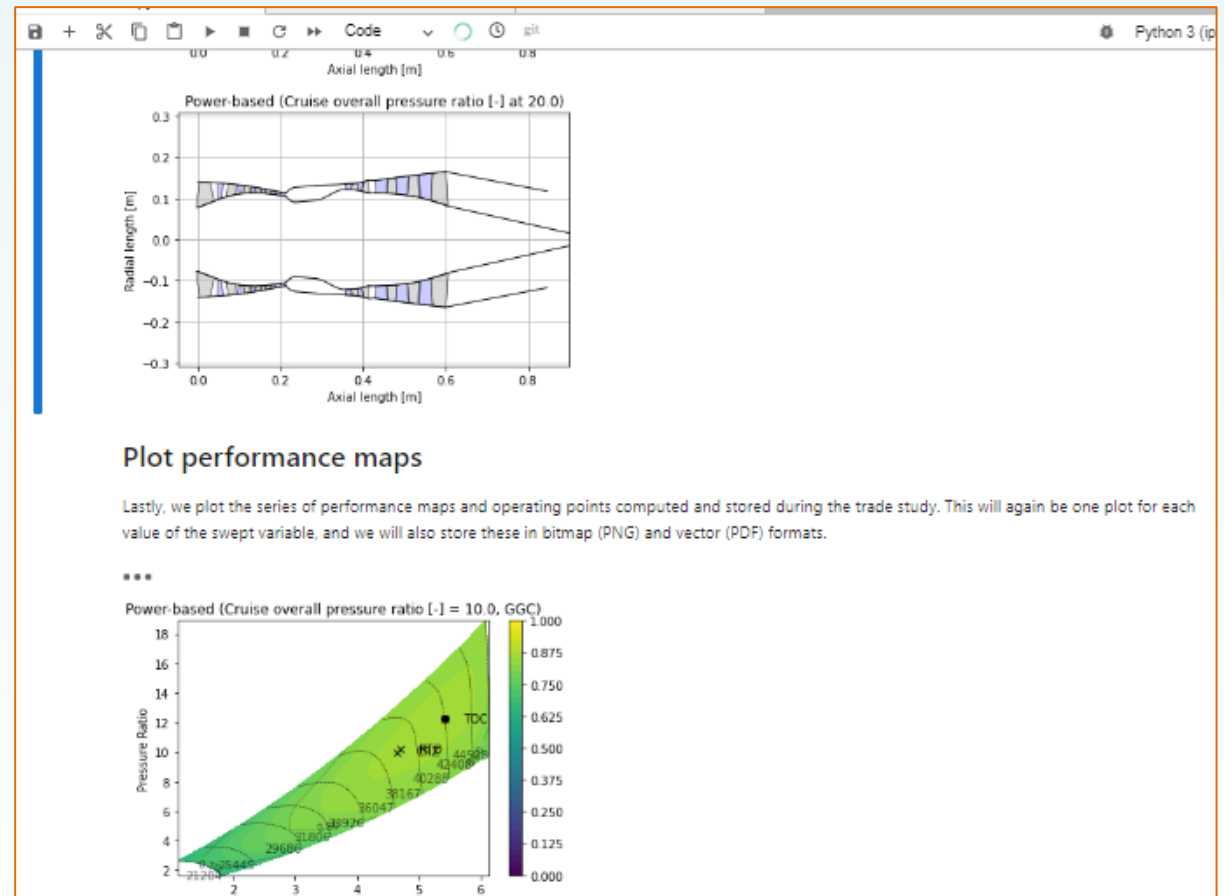
```
from modelon.impact.client import SimpleExperimentExtension

experiment_extension_1 = SimpleExperimentExtension(
    parameter_modifiers={'final_time': 2.0},
    solver_options={'atol': 1e-9},
    simulation_options=dynamic.get_simulation_options().with_values(ncp=1500),
)
experiment_extension_2 = SimpleExperimentExtension(
    parameter_modifiers={'final_time': 5.0},
    solver_options={'atol': 1e-10},
    simulation_options=dynamic.get_simulation_options().with_values(ncp=1200),
)
```

```
experiment_definition = experiment_definition.with_extensions(
    [experiment_extension_1, experiment_extension_2]
)
```

Demo 4:

- OCT, SteadyState
- Specialized report -> ppt



System

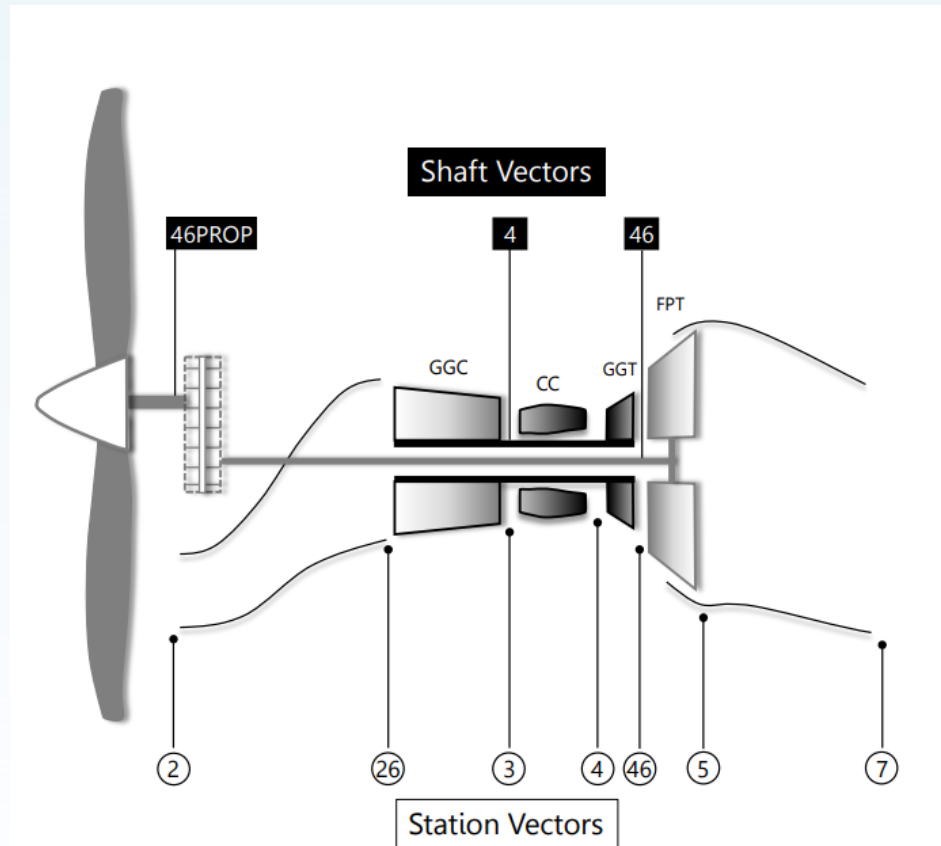


FIGURE 1. Parallel hybrid turboprop cycle schematic

Analysis

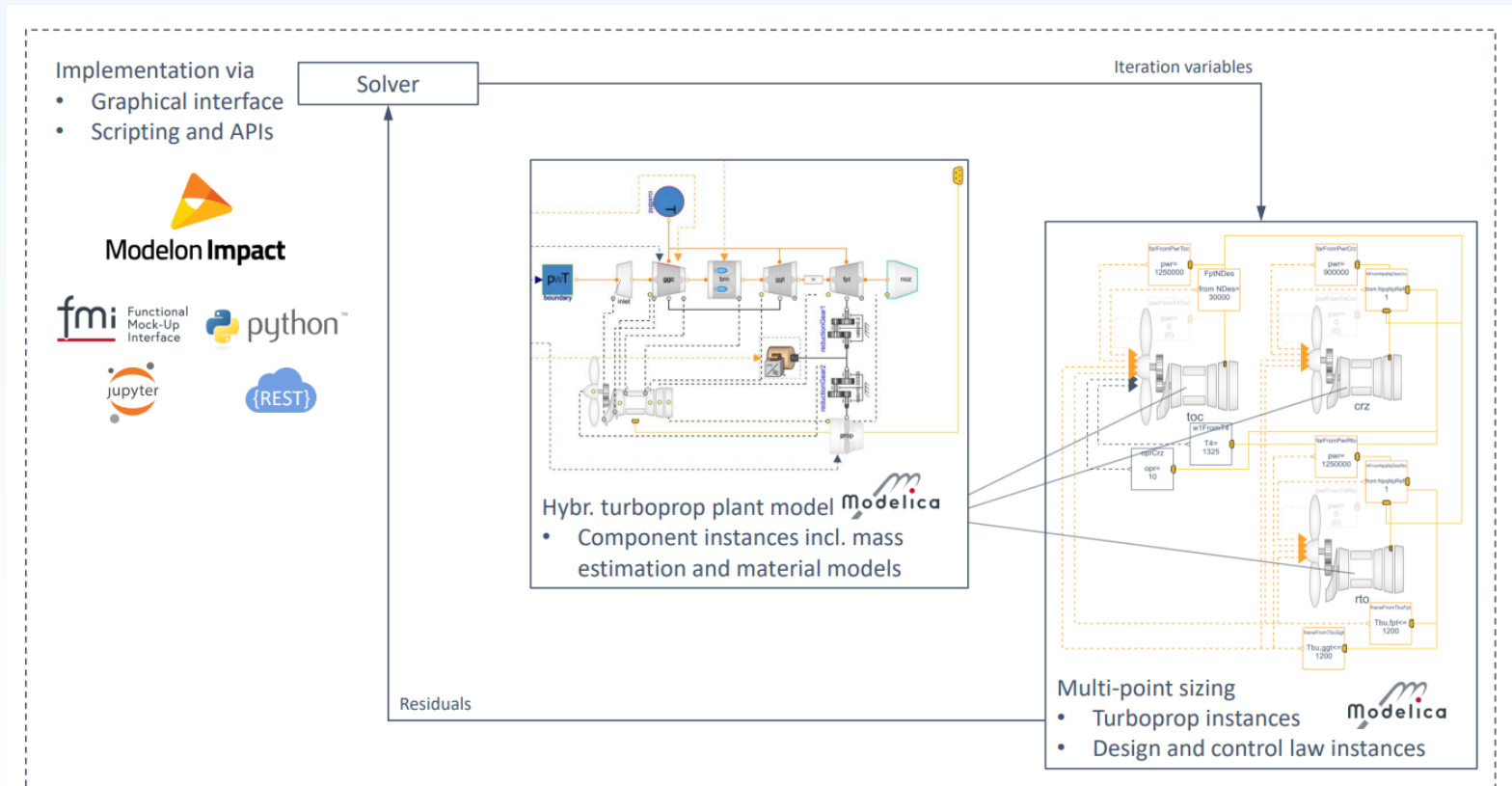


FIGURE 2. Modeling and simulation methodology schematic

Results

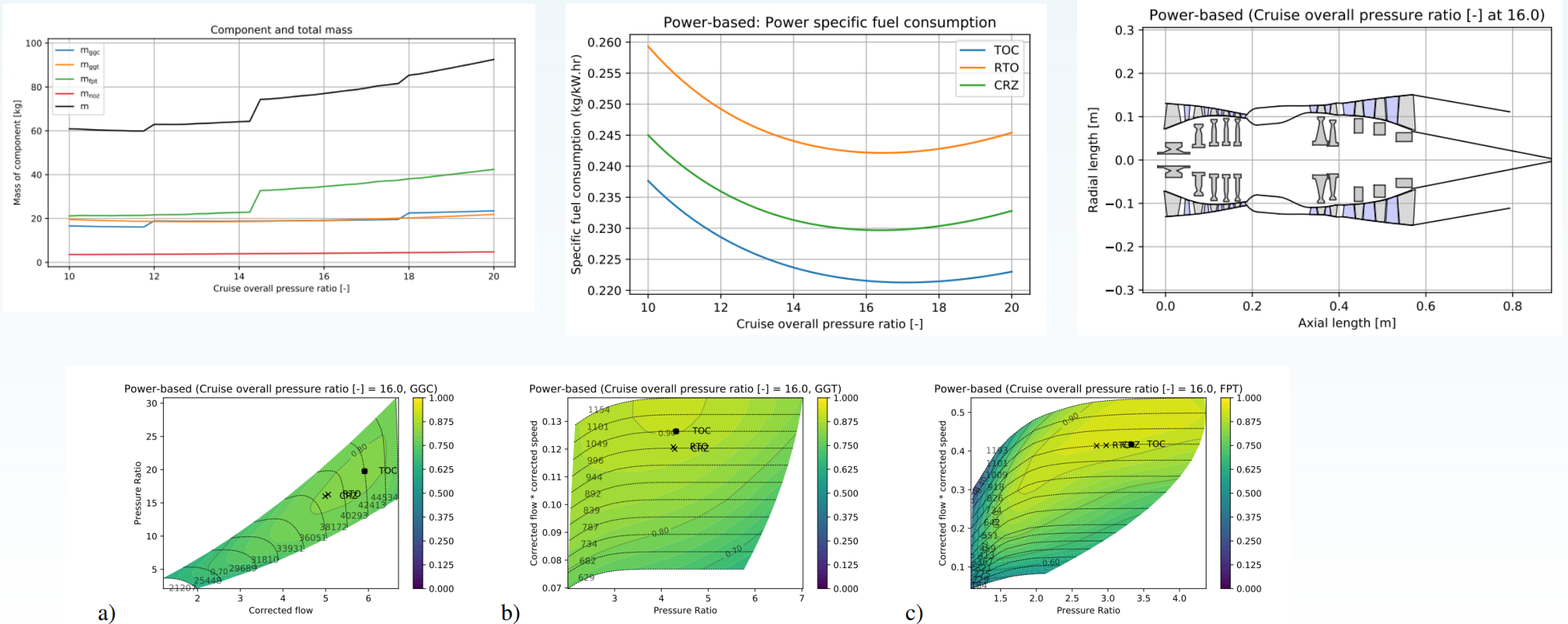


FIGURE 6. Performance map operating points of conventional design with OPR_{CRZ} of 16 (GGC, GGT, FPT from left to right)

Summary

- Post-Processing using a variety of Python libraries
- Multi-execution using the Modelon Impact Python Client
- Report generation to ppt.



Accurate Simulations. Better Decisions.