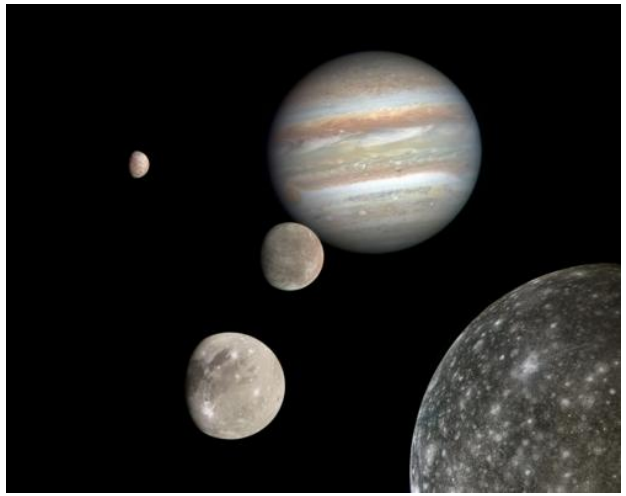


Beyond Jupyter: Energy Modeling with MOSEK

Jens Schulz
COO – Mosek ApS

Jupyter and marimo in numbers

JUPYTER

- ~10 million notebooks on GitHub by 2021

- Growth numbers:
  - 2011: First IPython Notebook released.
    2014: Project Jupyter created.
    2015: 200k on GitHub
    2018: 2.5M on GitHub
    2021: ~10M on GitHub
    (source: Wikipedia)

  - VS code (July 2026): 106.6M downloads
    (July 2022): 40M downloads
    second most popular extension!

MARIMO

- 10k's of users

- Growth numbers
  - Created in 2023
  - Today: Marimo Github repo has 21k+ stars (exceptionally fast!)
  - 180k+ weekly downloads

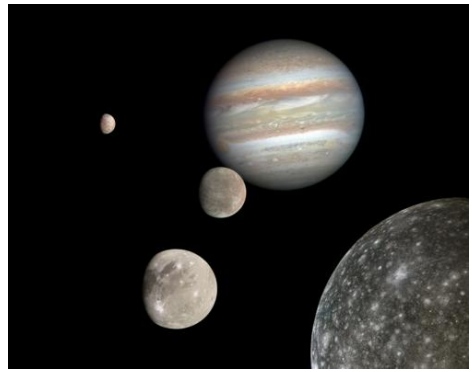
  - VS code (July 2026): 58k downloads

mosek

Jupyter Notebooks - The pain points

- Inconvenient:
  - Change value in a cell -> need to ensure you re-run the correct cells
  - Creates hidden states (internally inconsistent)
- Large JSON files
  - Terrible GIT diffs
  - Cannot execute easily standalone (CI, command line)
  - Limited re-usability (import functions,...)

Advantages of Jupyter



- Huge community
- Many extensions
- Support in virtually every scientific library
- Widespread familiarity in universities
- Frontend interface for cloud users, Amazon Sagemaker Notebook, Google Colab, MS Azure Notebook, GitHub Codespaces
- Interactivity:
  - ipywidgets for interactive components
  - e.g., streamlit for app deployment

mosek

Jupyter versus marimo notebooks – A star for optimization use cases

- Inconvenient:
  - change value in a cell -> need to ensure you re-run the correct cells
  - Creates hidden states (internally inconsistent)
- Large JSON files
  - Terrible GIT diffs
  - Cannot execute easily standalone (CI, command line)
  - Limited re-usability (import functions,..)
- Interactivity:
  - ipywidgets for interactive components
  - e.g., streamlit for app deployment

Marimo notebooks

- Halfway between notebook, dashboard, and web app
- Reactive dependency graph
- Reproducible execution

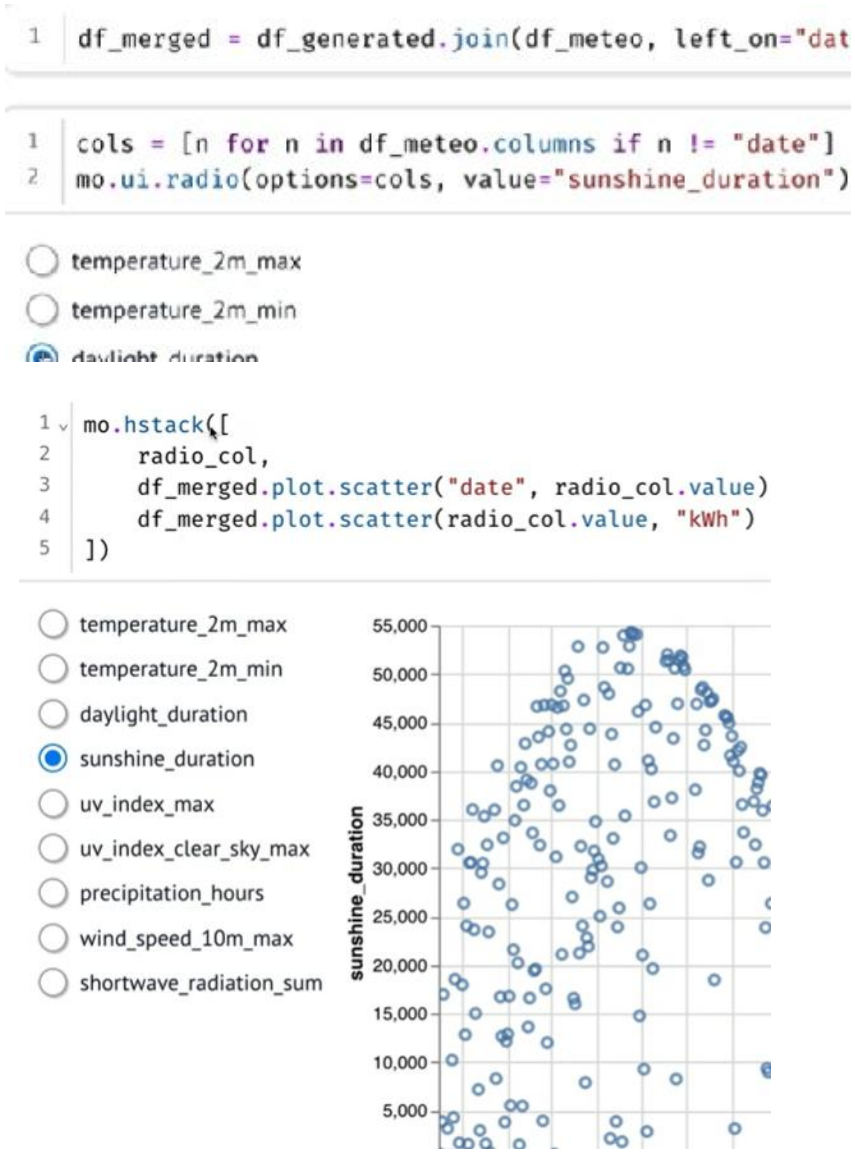
- Stored as Python code
- Clean version control
- Testing of single functions

- Reactive user interfaces
- Deploy as interactive web app w/o additional packages

mosek

Advantages of marimo for optimization use cases

- Marimo notebooks are pure Python scripts
  - Run notebooks as if they were scripts
  - Run notebooks as if they were apps
  - Supports CI/CD workflows
  - Easy to integrate with GIT
  - Ability to re-use function definitions
- Reactive Python environment
  - Dependency graph to re-run only dependent cells
- Ability to deploy as web app
  - Mix UI elements with code
  - No Python backend, fully local browser
  - “Pydite” helps to run on WASM
  - Partly limited, not all Python packages
- Allows to: “re-think the way we are working with data”



Main concepts

- App:
  - Instantiates the marimo application, serving as the container for all reactive cells and their dependency graph
- Cells:
  - Implement reactive computations that automatically update when their dependencies change
- Setup cells:
  - Executed once to define imports, constants, and configuration shared across the notebook
- Functions:
  - Encapsulate reusable Python logic
  - Easy to test

- At top of marimo notebook write:

```
import marimo
app = marimo.App()
```

-
```
@app.cell()
def _(dependencies):
    your code
    return values for other cells
```

- Setup cells:
```
with app.setup:
    import xyz
    global values
```

-
```
@app.function
def compute():
    x = 2
    y = 3
    return x + y
```

mosek

Testing / Continuous Integration: marimo apps support pytest

- Define “test_” functions within app cells
- Use assert condition based on specific values in notebook; or call a function from this marimo app
- Run “pytest solar_battery.py”

Callstack of pytest

- import solar_battery.py
  ↓
  marimo builds notebook
  (register app cells)
  ↓
  marimo executes required cells
  (test_* dependency graph)
  ↓
  pytest executes test_* functions

Example:

- ```
  @app.function
  def test_compute():
      print("Testing compute function...")
      assert compute() == 5
  ```

- Command line:
  ```
  Pytest solar_battery.py
  ```

mosek
  ```

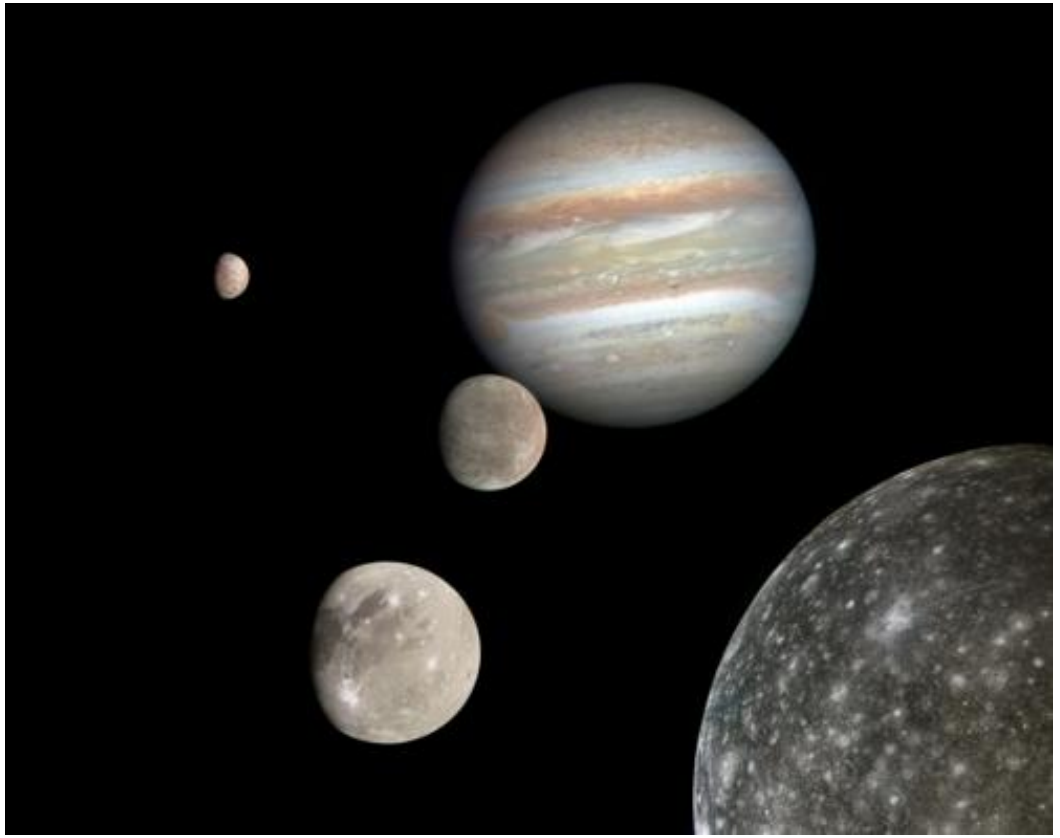
Learnings along the way

- **Avoid Variable/Function Redefinition Errors**
  - Currently, marimo is too conservative!
- Make Functions Reusable/Exportable via “Reusable Cells” = app functions
- Separation of concerns
  - Business logic (load data, solve, generate solution, etc.)
    → app functions
    → unit/smoke tested with pytest
  - Marimo UI
    → Reactive cells
    → manually verified or tested with browser automation if desired.
    E.g., playwright library to test that app starts, widgets exist, optimization completes
    → test that app starts successfully (‘marimo run xxx.py”)

mosek

Beyond Jupyter: Marimo notebooks

- Marimo in early stage v0.23.x
- Promising to cover optimization use cases
- Pure Python:
  - Much better for version control
  - Fulfills CI and testing requirements
- Reactive components that avoid hidden states



mosek

MOSEK

Our solver is the answer for all your LPs, QPs, SOCPs, SDPs, and MIPs. Includes interfaces to C, C++, Java, MATLAB, .NET, Python, Julia, Rust and R.

$$C := \{x \in \mathrm{R}^3 : x_1 \geq \sqrt{x_2^2 + x_3^2}\}$$

Discover

Try

Academic Licenses:

https://www.mosek.com/products/academic-licenses/

Join Mosek:

Specialist for Sparse Linear Algebra

https://www.mosek.com/about/career-options/

mosek

GOR

PMO111

Optimization in the era of Generative AI

Munich, Germany
November 26 & 27, 2026
Hosted by Flix SE

Source: AI-generated image

PMO112

Shaping the Mobility of the Future: A Mathematical Optimization Perspective

Ingolstadt, Germany
March 11 & 12, 2027

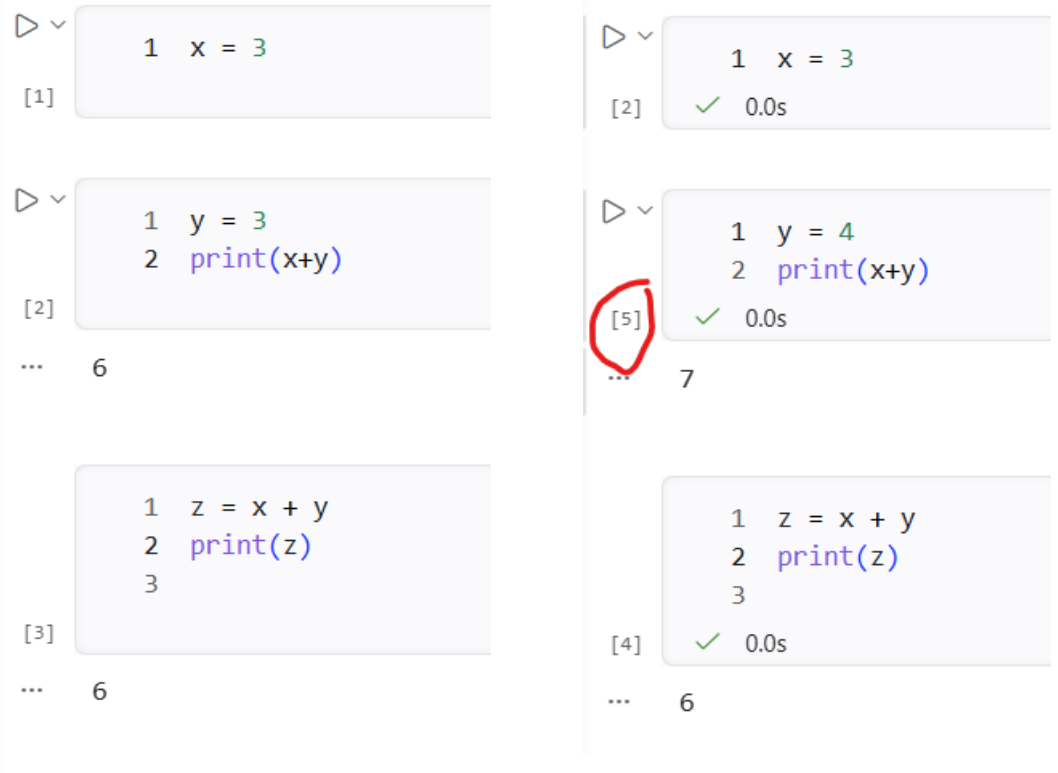
Organized with TH Ingolstadt

Source: https://unsplash.com/de/fotos/luftaufnahmen-von-betonstrassen-7nrsVjvALnA

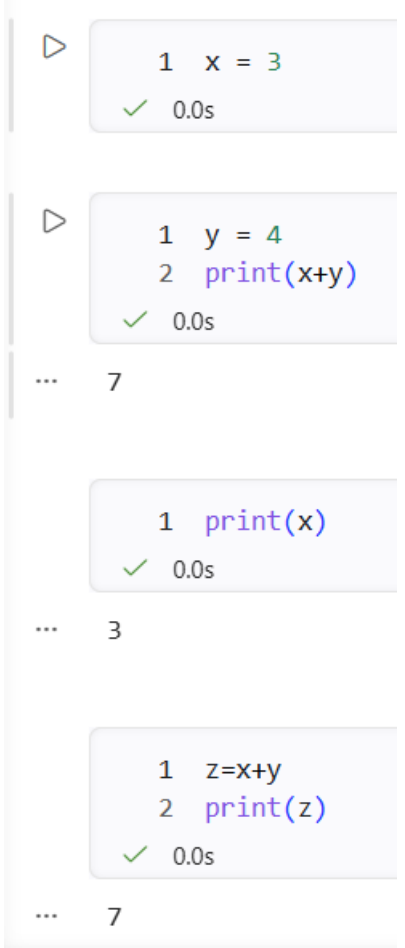
https://www.gor-ev.de/arbeitsgruppen/praxis-der-mathematischen-optimierung


mosek

Jupyter Notebook- inconsistent (hidden) state



Marimo – reactive execution model
dependency graph will execute dependent cells



mosek

```python
def compute():
        x = 3
        y = 5
        return x + y
```

✓ 0.0s compute

```python
z = compute()
print(f"z: {z}")
```

✓ 0.0s

```
z: 8
```

```python
def test_compute():
        print("Testing compute function...")
        assert compute() == 8
```

✓ 0.0s test_compute

```
(.venv) PS C:\Users\jenss\gitprojects\ifors-energy> pytest .\B_compute_test.py
============================= test session starts =============================
platform win32 -- Python 3.13.12, pytest-9.1.1, pluggy-1.6.0
rootdir: C:\Users\jenss\gitprojects\ifors-energy
plugins: anyio-4.14.1
collected 1 item

B_compute_test.py .                                                      [100%]

============================== 1 passed in 0.88s ==============================
```

mosek

```python
def compute():
        x = 3
        y = 5
        return x + y
```

0.0s compute

```python
z = compute()
print(f"z: {z}")
```

0.0s

```
z: 8
```

```python
def test_compute():
        print("Testing compute function...")
        assert compute() == 8
```

0.0s test_compute

```python
import marimo

__generated_with = "0.23.9"
app = marimo.App()


@app.function
def compute():
        x = 3
        y = 5
        return x + y


@app.cell
def _():
    z = compute()
    print(f"z: {z}")
    return


@app.function
def test_compute():
        print("Testing compute function...")
        assert compute() == 8
```

Open as notebook

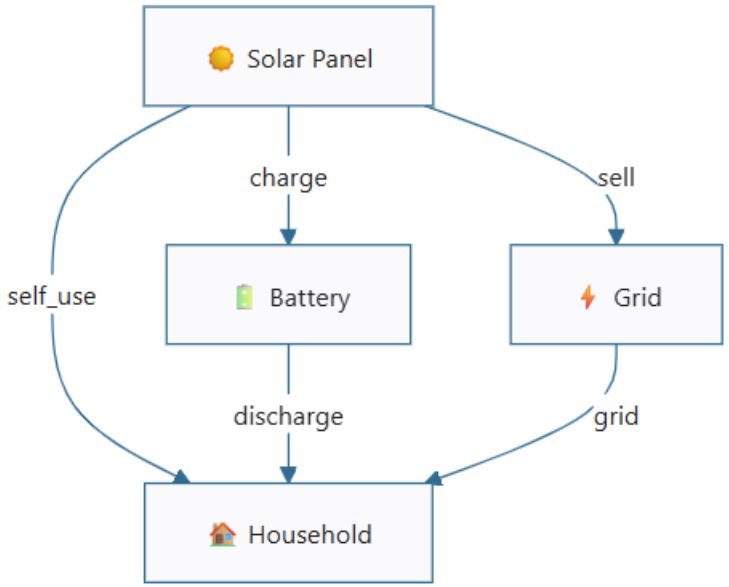
Import and use functions from other notebooks

```python
from B_compute_test import compute
```

```python
print(compute())
```

Household - Solar Energy Optimization with Battery

Optimizing the operation of a battery attached to a solar PV installation over a certain time horizon, e.g., 24-hours.



mosek

```
## Input Data / Parameters


\begin{aligned}
\text{PV generation:} &\quad g_t \\
\text{Household demand:} &\quad d_t \\
\text{Selling price:} &\quad p_t^{sell}\\
\text{Grid purchase price:} &\quad p_t^{grid}\\
\text{Initial battery state of charge:} &\quad E_0\\
\text{Battery capacity:} &\quad E_{max} \\
\text{Charging/discharging efficiency:} &\quad \eta_c, eta_d
\end{aligned}
```

Input Data / Parameters

$$
\begin{aligned}
\text{PV generation:} &\quad g_t \\
\text{Household demand:} &\quad d_t \\
\text{Selling price:} &\quad p_t^{sell}\\
\text{Grid purchase price:} &\quad p_t^{grid}\\
\text{Initial battery state of charge:} &\quad E_0\\
\text{Battery capacity:} &\quad E_{max}
\end{aligned}
$$

```
## Model : Variables, Objective and Constraints

Maximize profit: gain for sold energy less cost to grid
$$ \max \sum_{t=1}^T \large(  p_t^{sell} sell_t - p_t^{grid} grid_t   \large) $$

Subject to:

Demand balance: my demand can be fulfilled from grid, pv production or battery
$$ selfuse_t + discharge_t + grid_t = d_t, \forall t $$

PV allocation: hourly solar energy produced is either used, sold or charged
$$selfuse_t  + sell_t  + charge_t  = pv_t,\forall t $$

Battery dynamics: Charging and discharging a battery following standard model with round trip efficiency
$$SoC_{t+1}=SoC_t  + \eta_c P_t^{charge} - \frac{1}{\eta_d} P_t^{discharge}, \qquad  \forall t=1,...,
T-1$$

Battery initial state:
$$SoC_1 = E_0 $$
Battery capacity:
$$0 \leq SoC_t \leq E_{max}  \forall t$$
```

Model : Variables, Objective and Constraints

Maximize profit: gain for sold energy less cost to grid

$$ \max \sum_{t=1}^T (p_t^{sell} sell_t - p_t^{grid} grid_t) $$

Subject to:

Demand balance: my demand can be fulfilled from grid, pv production or battery

$$ selfuse_t + discharge_t + grid_t = d_t, \forall t $$

PV allocation: hourly solar energy produced is either used, sold or charged

$$selfuse_t + sell_t + charge_t = pv_t,\forall t $$

Battery dynamics: Charging and discharging a battery following standard model with round trip efficiency

$$SoC_{t+1}=SoC_t + \eta_c P_t^{charge} - \frac{1}{\eta_d} P_t^{discharge}, \qquad \forall t=1,...,T-1$$

Battery initial state:

$$SoC_1 = E_0 $$

Battery capacity:

$$0 \leq SoC_t \leq E_{max} \forall t$$

Mathematical Model in Python

```
M = mf.Model("HouseholdEnergy")
#M.setSolverParam("cacheLicense", "off")

# Decision Variables:
self_use = M.variable("self_use", n, mf.Domain.greaterThan(0.0))
sell = M.variable("sell", n, mf.Domain.greaterThan(0.0))
charge = M.variable("charge", n, mf.Domain.greaterThan(0.0))
discharge = M.variable("discharge", n, mf.Domain.greaterThan(0.0))
grid = M.variable("grid", n, mf.Domain.greaterThan(0.0))

soc = M.variable("soc", n, mf.Domain.inRange(0.0, battery_capacity))
```

PV allocation: hourly solar energy produced is either used, sold or charged

$$selfuse_t + sell_t + charge_t = pv_t, \forall t$$

```
# PV allocation: hourly solar energy produced is either used, sold or charged
cons_pv = M.constraint(
    mf.Expr.add([self_use, sell, charge]),
    mf.Domain.equalsTo(pv.tolist()),
)
```

mosek

Household - Solar Energy Optimization with Battery

Optimizing the operation of a battery attached to a solar PV installation over a certain time horizon, e.g., 24-hours.

Data Input: toy

Verbosity Threshold (<=)

Pick demand file

Pick solar production file

Battery Capacity (kWh)

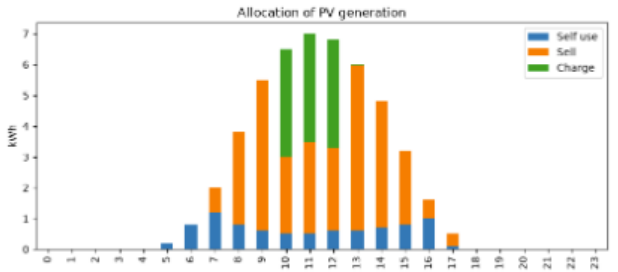
Initial State of Charge (kWh)

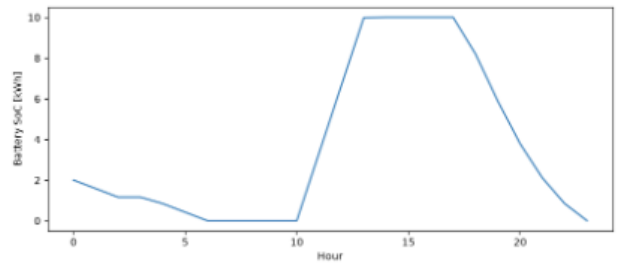
Hour

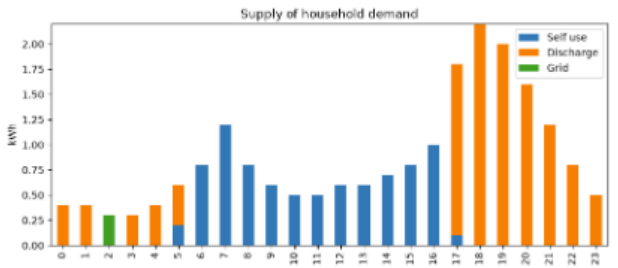
Demand at 0	Solar production at 0	Battery start state at 0
0.4 kWh	0.0 kWh	2.0 kWh

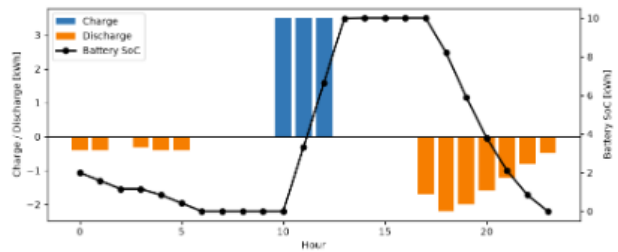
⚡ Grid Purchase	☀ PV Sold	▣ Battery Charge	▣ Battery Discharge
-0.0 kWh	0.0 kWh	0.0 kWh	0.4 kWh

€ Daily Profit	⚡ Grid Purchase	☀ PV Sold	▣ Battery Charge	▣ Battery Discharge
0.84 €	0.3 kWh	29.8 kWh	10.5 kWh	11.9 kWh









mosek