

HORIZON-CL5-2022-D3-01-11

Demonstration of innovative forms of storage and their successful operation and integration into innovative energy system and grid architectures



AGISTIN

Advanced Grid Interfaces for
innovative STorage INtegration

Battery Model Insights



Le réseau
de transport
d'électricité



Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Climate, Infrastructure and Environment Executive Agency (CINEA). Neither the European Union nor the granting authority can be held responsible for them.

Agenda

- Project Overview
- Project Directory Structure
- Key Battery Model Files
- Batteries.py - Parameters
- Batteries.py - Variables
- Batteries.py - Constraints
- ExampleBatteryPV_Case_1_131124.py
- Optimization Workflow
- Data Handling & Solver
- Conclusion

Project Introduction

- **Project Overview:** The Battery Model Project aims to enhance synergy between battery storage, photovoltaic systems, and grid performance.
- **Focus Areas:** Key focus includes optimization techniques for maximizing efficiency in energy storage and generation processes.
- **Integration Importance:** A critical aspect is understanding how optimizing both components leads to improved overall energy management.

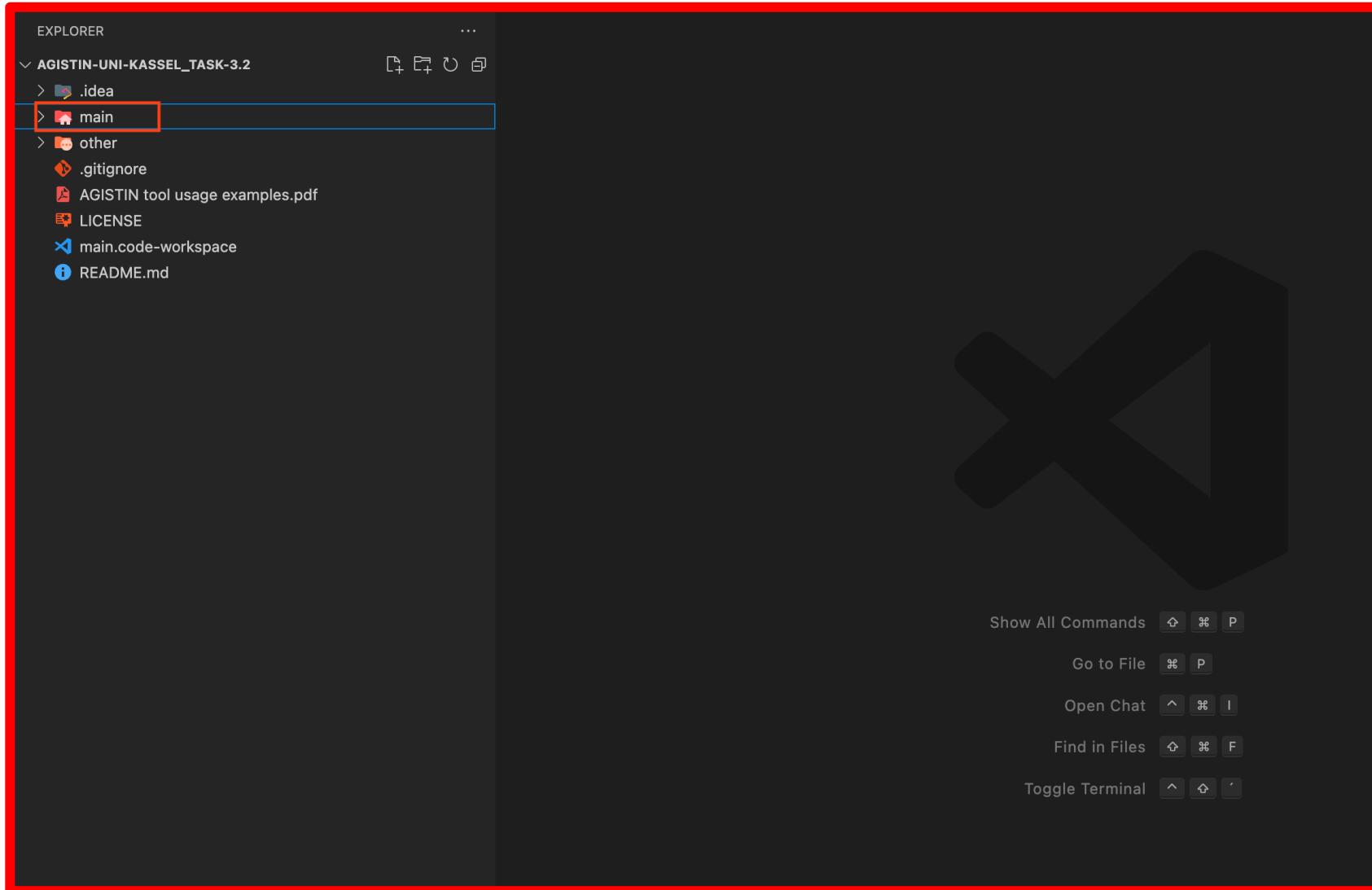
Project Directory Structure

- **Directory Structure:** The directory contains organized folders for efficient project management and accessibility of files.
- **Main Folder:** 'agistin-Uni-Kassel_Task-3.2' serves as the root, centralizing all project-related documentation and substructures.
- **Subfolders Overview:** Subfolders include 'main' for primary files, 'Cases' for data, and 'Devices' for Python scripts.

- 1- This is our GitHub repository.[<https://github.com/sergicosta/agistin/branches>]
- 2- It contains a main file where all official work is updated.
- 3- Additionally, we have a copy of the file where we work, which I have highlighted.

The screenshot shows a GitHub repository interface for the user 'sergicosta' named 'agistin'. The 'Code' tab is selected. The main area displays three sections: 'Default', 'Your branches', and 'Active branches'. The 'Default' section shows the 'main' branch, which was updated 5 hours ago and is marked as 'Default'. The 'Your branches' section shows the 'Uni-Kassel_Task-3.2' branch, which was updated 2 weeks ago and has 198 commits behind and 39 ahead. This branch is highlighted with an orange border. The 'Active branches' section shows the same 'Uni-Kassel_Task-3.2' branch with the same status. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights, along with a search bar and various icons for repository management.

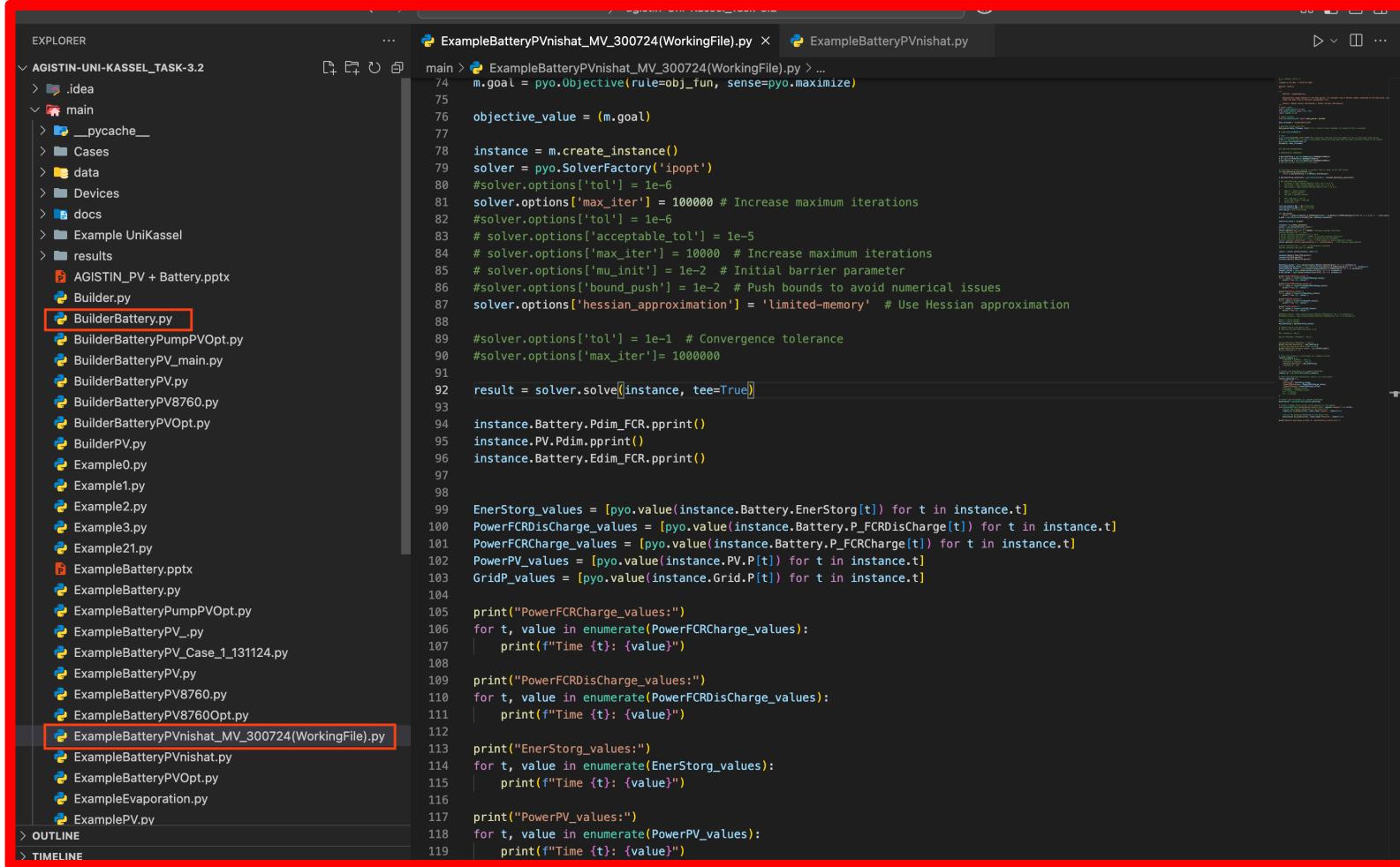
1- In this, we first have a main folder that contains subfolders and files inside it.



1- In this, we have a main file: **ExampleBatteryPVnishat_MV_300724(WorkingFile)**.

2- This file is where we declare the data points, and we can custom-select the ones present in the Excel file.

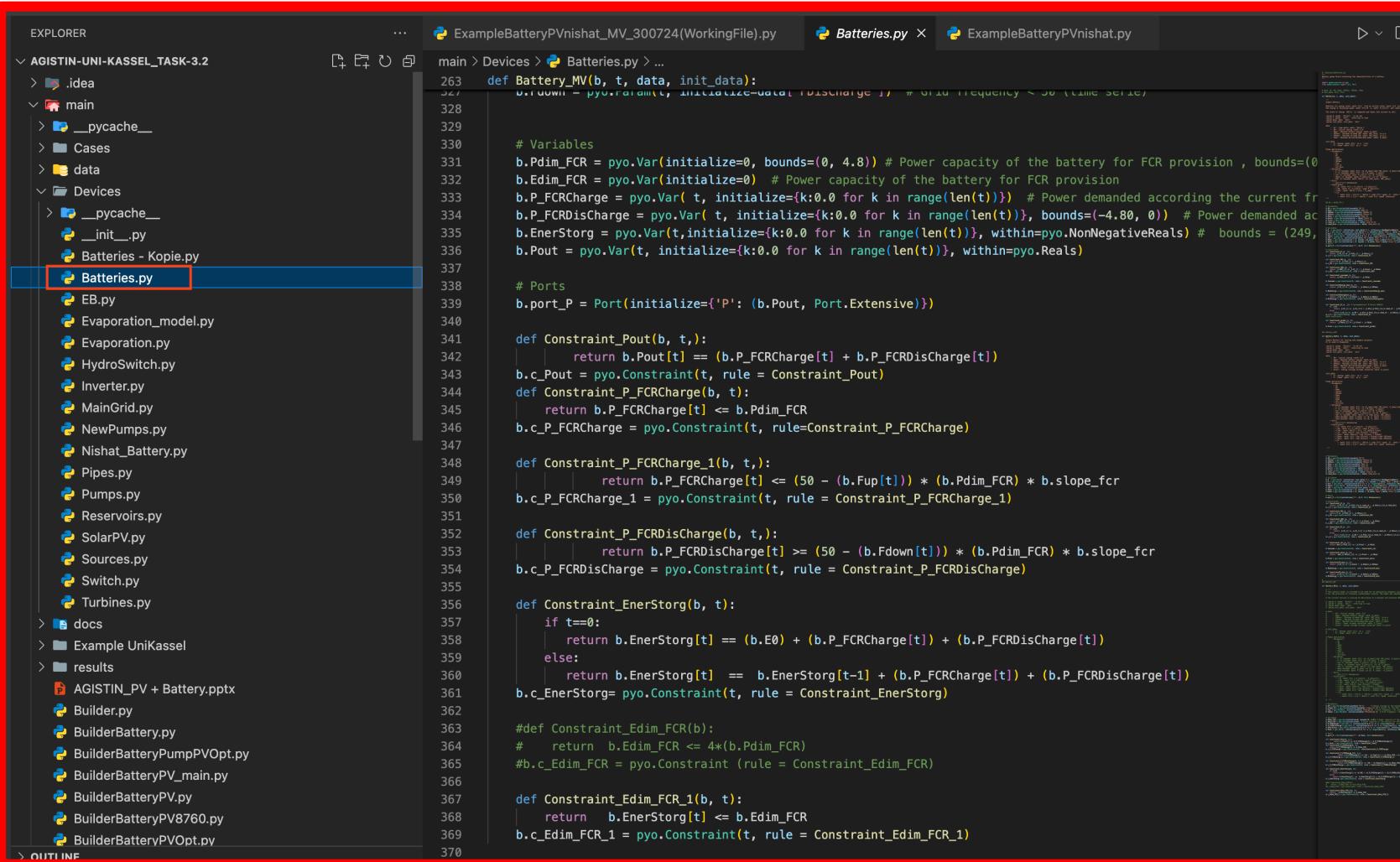
3- There's also a **builderbattery.py** file, which helps in calling our JSON and Excel files. We don't make many changes to this file.



The screenshot shows a code editor interface with a red border. On the left is the Explorer panel, displaying a project structure under 'AGISTIN-UNI-KASSEL_TASK-3.2'. The 'main' folder contains several Python files, including 'BuilderBattery.py' (highlighted with a red box), 'BuilderBatteryPumpPVOpt.py', 'BuilderBatteryPV_main.py', 'BuilderBatteryPV.py', 'BuilderBatteryPV8760.py', 'BuilderPVOpt.py', 'Example0.py', 'Example1.py', 'Example2.py', 'Example3.py', 'Example21.py', 'ExampleBattery.pptx', 'ExampleBattery.py', 'ExampleBatteryPumpPVOpt.py', 'ExampleBatteryPV.py', 'ExampleBatteryPV_Case_1_131124.py', 'ExampleBatteryPV8760.py', 'ExampleBatteryPV8760Opt.py', 'ExampleBatteryPVnishat_MV_300724(WorkingFile).py' (highlighted with a red box), 'ExampleBatteryPVnishat.py', 'ExamplePVOpt.py', 'ExampleEvaporation.py', and 'ExamplePV.py'. On the right is the main editor window showing the content of 'ExampleBatteryPVnishat_MV_300724(WorkingFile).py'. The code uses Pyomo to define an optimization model for a battery system, setting up variables, constraints, and an objective function. It includes solver options like 'ipopt' and various tolerance parameters. The code then solves the model and prints out results for PowerFCRCharge, PowerFCRDischarge, EnerStorg, and PowerPV values across time steps.

```
main > ExampleBatteryPVnishat_MV_300724(WorkingFile).py > ExampleBatteryPVnishat.py
74     m.goal = pyo.Objective(rule=obj_fun, sense=pyo.maximize)
75
76     objective_value = (m.goal)
77
78     instance = m.create_instance()
79     solver = pyo.SolverFactory('ipopt')
80     #solver.options['tol'] = 1e-6
81     solver.options['max_iter'] = 100000 # Increase maximum iterations
82     #solver.options['tol'] = 1e-6
83     # solver.options['acceptable_tol'] = 1e-5
84     # solver.options['max_iter'] = 10000 # Increase maximum iterations
85     # solver.options['mu_init'] = 1e-2 # Initial barrier parameter
86     #solver.options['bound_push'] = 1e-2 # Push bounds to avoid numerical issues
87     solver.options['hessian_approximation'] = 'limited-memory' # Use Hessian approximation
88
89     #solver.options['tol'] = 1e-1 # Convergence tolerance
90     #solver.options['max_iter']= 1000000
91
92     result = solver.solve(instance, tee=True)
93
94     instance.Battery.Pdim_FCR pprint()
95     instance.PV.Pdim pprint()
96     instance.Battery.Edim_FCR pprint()
97
98
99     EnerStorg_values = [pyo.value(instance.Battery.EnerStorg[t]) for t in instance.t]
100    PowerFCRDischarge_values = [pyo.value(instance.Battery.P_FCRDischarge[t]) for t in instance.t]
101    PowerFCRCharge_values = [pyo.value(instance.Battery.P_FCRCharge[t]) for t in instance.t]
102    PowerPV_values = [pyo.value(instance.PV.P[t]) for t in instance.t]
103    GridP_values = [pyo.value(instance.Grid.P[t]) for t in instance.t]
104
105    print("PowerFCRCharge_values:")
106    for t, value in enumerate(PowerFCRCharge_values):
107        print(f"Time {t}: {value}")
108
109    print("PowerFCRDischarge_values:")
110    for t, value in enumerate(PowerFCRDischarge_values):
111        print(f"Time {t}: {value}")
112
113    print("EnerStorg_values:")
114    for t, value in enumerate(EnerStorg_values):
115        print(f"Time {t}: {value}")
116
117    print("PowerPV_values:")
118    for t, value in enumerate(PowerPV_values):
119        print(f"Time {t}: {value}")
```

- 1- In this, we have a **devices** folder containing the **batteries.py** file.
- 2- In this file, we declare different variables, constraints, and functions that we want to print or use.



The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "AGISTIN-UNI-KASSEL_TASK-3.2". The "Devices" folder is expanded, and "Batteries.py" is selected and highlighted with a red border.
- Code Editor**: The main window displays the content of "Batteries.py". The code uses Pyomo to define variables and constraints for battery operations. Key parts include:
 - Initialization: `def Battery_MV(b, t, data, init_data):`
 - Variables: `b.Pdim_FCR = pyo.Var(initialize=0, bounds=(0, 4.8))`
 - Constraints:
 - FCR provision: `def Constraint_Pout(b, t):`
 - FCR charge limit: `def Constraint_P_FCRCharge(b, t):`
 - FCR discharge limit: `def Constraint_P_FCRDischarge(b, t):`
 - FCR discharge rate limit: `def Constraint_P_FCRDischarge_1(b, t):`
 - Energy storage balance: `def Constraint_EnerStorg(b, t):`
 - Edim_FCR constraint: `def Constraint_Edim_FCR(b):`
 - Edim_FCR discharge limit: `def Constraint_Edim_FCR_1(b, t):`
 - Port definition: `b.port_P = Port(initialize={'P': (b.Pout, Port.Extensive)})`
- Right Panel**: A vertical panel on the right side of the code editor shows a detailed log or output of the code execution, likely related to the Pyomo solver results.

- 1- In this, we have a **cases** folder that contains 3 files, which are marked.
- 2- The important file is **examplebatteryPV_time**, where we declare our data points.
- 3- To run different cases, these 3 files are used.

The screenshot shows a code editor interface with the following details:

- EXPLORER** pane on the left:
 - Project: AGISTIN-UNI-KASSEL_TASK-3.2
 - Files and Folders:
 - .idea
 - main
 - __pycache__
 - Cases
 - ~-ExampleBatteryPV_time copy 55 steps.xlsx
 - ~-ExampleBatteryPV_time.xlsx
 - Case_1.xlsx
 - Example3_cost.json
 - Example3_cost.xlsx
 - Example3_time.xlsx
 - Example3.json
 - Example3.xlsx
 - ExampleBattery_cost.xlsx
 - ExampleBattery_time.xlsx
 - ExampleBattery.xlsx
 - ExampleBatteryPumpPV_cost.json
 - ExampleBatteryPumpPV_cost.xlsx
 - ExampleBatteryPumpPV_time.xlsx
 - ExampleBatteryPumpPV.json
 - ExampleBatteryPumpPV.xlsx
 - ExampleBatteryPV_cost.json
 - ExampleBatteryPV_cost.xlsx
 - ExampleBatteryPV_time - Kopie.xlsx** (highlighted with an orange box)
 - ExampleBatteryPV_time copy 2 with FRC CHARGE VAL... (highlighted with an orange box)
 - ExampleBatteryPV_time copy 55 steps.xlsx
 - ExampleBatteryPV_time copy.xlsx
 - ExampleBatteryPV_time_with_2190.xlsx
 - ExampleBatteryPV_time..55.xlsx
 - ExampleBatteryPV_time.WRONC_DATA.xlsx
 - ExampleBatteryPV_time.xlsx** (highlighted with an orange box)
 - ExampleBatteryPV.json
 - ExampleBatteryPV.xlsx
 - ExampleBatteryPV(old).xlsx
 - ExampleBatteryPV8760_cost.json
 - ExampleBatteryPV8760_cost.xlsx
 - ExampleBatteryPV8760_time.xlsx- Editor** pane on the right:
 - File: ExampleBatteryPVnishat_MV_300724(WorkingFile).py
 - Content pane:
 - File: ExampleBatteryPV.json
 - File: ExampleBatteryPV_time.xlsx
 - Table view:
 - Header: A, B, C, D, E
 - Data:

	A	B	C	D	E
1	Battery_F	Battery_FCharge	Battery_FDisCharge		
2	50.20		50		
3	50.20	50.20		50	
4	50.20	50.20		50	
5	50.20	50.20		50	
6	50.20	50.20		50	
7	50.10	50.10		50	
8	50.20	50.20		50	
9	50.10	50.10		50	
10	50.20	50.20		50	
11	50.20	50.20		50	
12	50.20	50.20		50	
13	50	50		50	
14	50	50		50	
15	50	50		50	
16	50	50		50	
17	50	50		50	
18	50	50		50	
19	50	50		50	
20	50	50		50	
21	50	50		50	
22	50	50		50	
23	50	50		50	
24	50	50		50	
25	50	50		50	

Key Battery Model Files

- **Core Model File:** 'BuilderBatteryPV' is crucial for assembling and developing the comprehensive battery model architecture efficiently.
- **Test Case Definition:** 'ExampleBatteryPVnishat_MV_300724' outlines specific test conditions to ensure model robustness and reliability.
- **File Integration:** Integration of these files enhances project modularity, facilitating easier adjustments and future developments.

```
28 ExampleBatteryPV_Case_1...
29 ExampleBatteryPV.py
30 ExampleBatteryPV8760.py
31 ExampleBatteryPV8760Opt....
32 ExampleBatteryPVnishat_M...
33 ExampleBatteryPVnishat.py
34 ExampleBatteryPVOpt.py
35 ExampleEvaporation.py
36
37 m = pyo.ConcreteModel()
38 |
39 # time
40 l_t = list(range(24)) #999 #TODO this sh
41 #TODO it would be nice to have a consist
42 m.t = pyo.Set(initialize=l_t)
43 builder(m, data_filename)
```

Batteries.py - Parameters

- **Initial Energy Storage:** Parameter `b.E0` indicates the initial stored energy within the battery, facilitating mathematical optimization.
- **FCR Performance Slope:** The variable `b.slope_fcr` represents the effective change in frequency response during charging and discharging phases.
- **Grid Frequency Deviations:** Parameters `b.Fup` and `b.Fdown` define acceptable grid frequency thresholds for efficient battery operation management.

```
📂 TestCase.json      323
📂 TestTurbine.json   324
> 📂 data            325
  ↘ Devices          326
    ↘ __pycache__     327
    __init__.py        328
  Batteries - Kopie... 329
  Batteries.py        330
                            331
                            332

# Parameters
b.E0 = pyo.Param(initialize=data['E0'])           # Energy storage at the bat-
b.Pmax = pyo.Param(initialize=data['Pmax'])         # Battery maximum power tha-
b.Emax = pyo.Param(initialize=data['Emax'])         # Battery maximum energy ca-
b.Einst = pyo.Param(initialize=data['Einst'])       # Battery installed rated e-
b.Pinst = pyo.Param(initialize=data['Pinst'])       # Battery installed rated p-
b.slope_fcr = pyo.Param(initialize=data['slope_fcr']) # statik for the FCI
b.F = pyo.Param(t, initialize=data['F'])             # Grid frequency (time serie
```

Batteries.py - Variables

- **Max Power Capacity:** Variable b.Pdim_FCR defines the battery's maximum power output during operational scenarios and peak demands.
- **Energy Capacity:** The energy storage capacity, represented by b.Edim_FCR, determines total energy storage available for discharge cycles.
- **Net Battery Output:** b.Pout indicates the actual output power from the battery, factoring in charging and discharging events.

```
> └── data          335
    └── Devices      336      # Variables
        ├── __pycache__ 337
        ├── __init__.py   338
        ├── Batteries - Ko... 339
        └── Batteries.py  340
    └── EB.py         341
    └── Einst.py       342
    └── Pinst.py       343
    └── Pmax.py        344

    335
    336      # Variables
    337
    338      b.EstrgOut = pyo.Var(t, initialize= 0.0, domain=pyo.Reals)
    339      b.SOC = pyo.Var(t, initialize= 0.0, bounds = (0,1), within=pyo.NonNegativeReals)
    340      b.EstrgIni = pyo.Var(t, initialize= 0.0, bounds = (0,data['Einst']), domain=pyo.NonNegativeReals)      # E
    341      b.Pdemanded = pyo.Var(t, initialize={k: 0.0 for k in range(len(t))}, domain=pyo.Reals)
    342      b.Pout = pyo.Var(t, initialize={k: 0.0 for k in range(len(t))},bounds = (-data['Pinst'],data['Pinst']), domain=pyo.Reals)
    343      b.Edim = pyo.Var(initialize = 0, bounds = (0,data['Emax']),within = pyo.NonNegativeReals)      # Additional e
```

Batteries.py - Constraints

- Power Output Constraint:**
Constraint_Pout ensures balance between charging and discharging power, maintaining operational stability and efficiency.
- State of Charge Tracking:**
Constraint_EnerStorg monitors the State of Charge, aligning with initial energy levels for optimal performance.
- Energy Capacity Limits:**
Constraint_Edim_FCR_1 defines the maximum and minimum energy capacities, ensuring the battery operates within safe thresholds.

```
Devices
> __pycache__
  __init__.py
  Batteries - Kopie.py
  Batteries.py
  EB.py
  Evaporation_model.py
  Evaporation.py
  HydroSwitch.py
  Inverter.py
  MainGrid.py
  NewPumps.py
  Nishat_Battery.py
  Pipes.py
  Pumps.py
  Reservoirs.py
  SolarPV.py
  Sources.py
  Switch.py
  Turbines.py
  docs
  Example UniKassel

# Ports
b.port_P = Port(initialize={'P': (b.Pout, Port.Extensive)})

def Constraint_Pout(b, t):
    return b.Pout[t] == (b.P_FCRCharge[t] + b.P_FCRDisCharge[t])
b.c_Pout = pyo.Constraint(t, rule = Constraint_Pout)

def Constraint_P_FCRCharge(b, t):
    return b.P_FCRCharge[t] <= b.Pdim_FCR
b.c_P_FCRCharge = pyo.Constraint(t, rule=Constraint_P_FCRCharge)

def Constraint_P_FCRCharge_1(b, t):
    return b.P_FCRCharge[t] <= (50 - (b.Fup[t])) * (b.Pdim_FCR) * b.slope_fcr
b.c_P_FCRCharge_1 = pyo.Constraint(t, rule = Constraint_P_FCRCharge_1)

def Constraint_P_FCRDisCharge(b, t):
    return b.P_FCRDisCharge[t] >= (50 - (b.Fdown[t])) * (b.Pdim_FCR) * b.slope_fcr
b.c_P_FCRDisCharge = pyo.Constraint(t, rule = Constraint_P_FCRDisCharge)

def Constraint_EnerStorg(b, t):
    if t==0:
        return b.EnerStorg[t] == (b.E0) + (b.P_FCRCharge[t]) + (b.P_FCRDisCharge[t])
    else:
        return b.EnerStorg[t] == b.EnerStorg[t-1] + (b.P_FCRCharge[t]) + (b.P_FCRDisCharge[t])
b.c_EnerStorg= pyo.Constraint(t, rule = Constraint_EnerStorg)
```

ExampleBatteryPV_Case_1_131124.py

- **Cost Parameters:** Fixed daily costs for capacities are crucial to evaluate financial feasibility of the battery system.
- **Time Steps:** We can customize them in the Excel file, for example, in **ExampleBatteryPV_time**, and then declare them in the main file: **ExampleBatteryPVnishat_MV_300724(Working File)**.
- **Decision Variables:** Key decision variables **Power** and **Energy Capacity** of the battery for optimization.

```
28 ExampleBatteryPV_Case_1_...
29 m = pyo.ConcreteModel()
30 |
31 # time
32 l_t = list(range(24)) #999 #TODO this sh
33 #TODO it would be nice to have a consist
34 m.t = pyo.Set(initialize=l_t)
35 builder(m, data_filename)
36
37
```

Optimization Workflow

- **Input Data Parsing:** This initial phase involves organizing and cleansing datasets for accurate model input during optimization.
- **Model Building with Pyomo:** Utilizing Pyomo, models integrate various components, defining relationships crucial for effective energy management solutions.
- **Results Extraction and Analysis:** Post-optimization analysis evaluates outputs, assessing model performance against predefined benchmarks for continuous improvement.

```
56     #     #m.W.value = W
57
58 cost_new_battery_MW  = 55 # Euros/day
59 cost_new_battery_MWh = 82 # Euros/day
60 cost_new_pv = 1.1 # Euros/day
61
62 def obj_fun(m):
63     return ((sum((m.Battery.P_FCRCharge[t]*22) + (m.Battery.P_FCRDisCharge[t])*-44 for t in l_t)) ) -
64             (cost_new_battery_MW*m.Battery.Pdim_FCR) - (cost_new_battery_MWh*m.Battery.Edim_FCR)) - (cost_new_pv*m.PV.Pdim)
65 m.goal = pyo.Objective(rule=obj_fun, sense=pyo.maximize)
66
67 objective_value = (m.goal)
68
69 instance = m.create_instance()
70 solver = pyo.SolverFactory('ipopt')
71 #solver.options['tol'] = 1e-6
```

Data Handling & Solver

- **Data Handling Sources:** The project utilizes external JSON files for structured data input, enhancing accessibility and integration.
- **Solver Configuration:** Ipopt settings include customized max iterations.
- **Execution of optimization:** Execute the constraints and the main file.



Conclusion

- **Integrated Model Overview:** The comprehensive framework combines battery storage, PV systems, and grid interaction for optimized energy management.
- **Benefits of Integration:** Enhanced efficiency, cost reduction, and improved reliability are key benefits observed from the integrated model implementation.
- **Next Steps:** Focus on further testing and optimization will ensure sustained improvements in model performance and scalability.

Thank you for
listening
