

# An Introduction to FINE

## Part I – Installing Software and Simple Model Runs

**Webinar/ Workshop/ Tutorial**

PRESENTER(S): ...

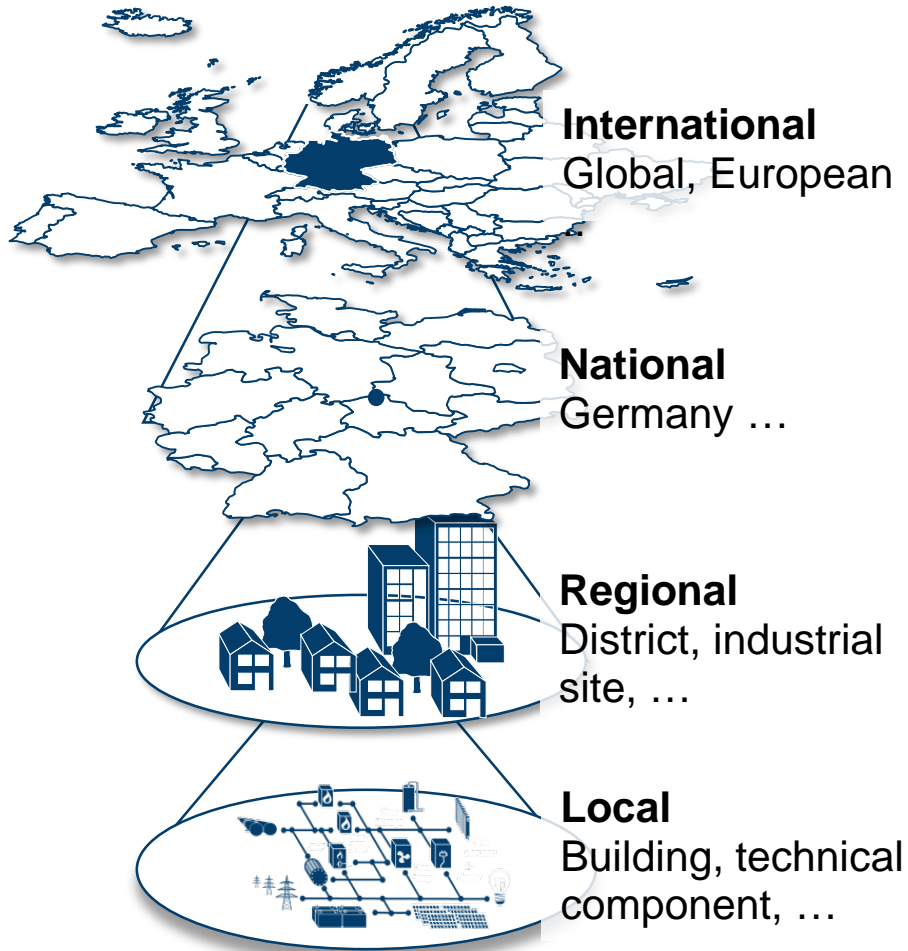
AUTHORS: LARA WELDER, THERESA GROß,  
JOCHEN LINßEN, MARTIN ROBINIUS, DETLEF STOLTEN

[l.welder@fz-juelich.de](mailto:l.welder@fz-juelich.de)

Techno-economic Systems Analysis (IEK-3)

# Which scenarios do we want to model?

## Application Levels



## Examples

- International resource supply pathways (coal, oil, natural gas/ synthetic gases)
- European energy supply networks (electricity, natural/ synthetic gases)
- One-nodal national energy systems with a focus on sectoral representation
- National energy supply systems with a focus on energy supply infrastructure
- Districts
- Single houses
- Single plants with individually modeled plant components

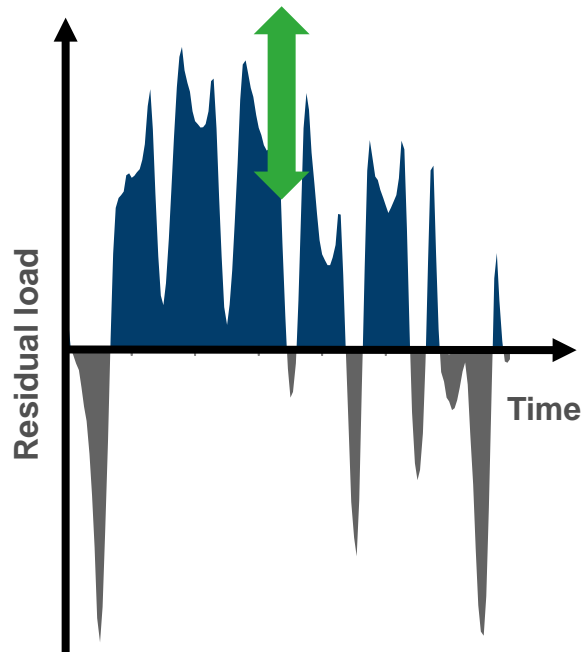
General questions  
and brain teasers

What are common modeling assumptions in these applications?

Which scenario would you like to model?

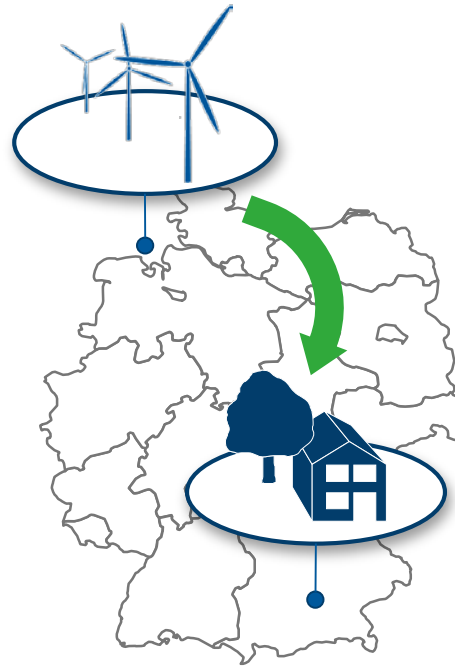
# Challenge: Capturing the Multi-dimensionality of Energy (Supply) Systems

## Temporal



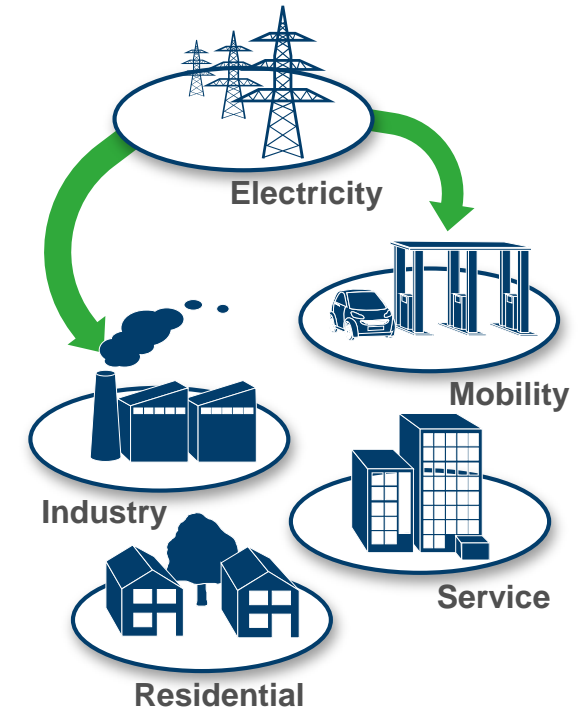
- Capture demand and renewable electricity production time series
- Design storage adequately (from daily to seasonal storage)

## Spatial



- Transmission infrastructure design and operation
- More specific placements recommendations
- Reduces copper plate assumption errors

## Cross-linked Sectors and Commodities



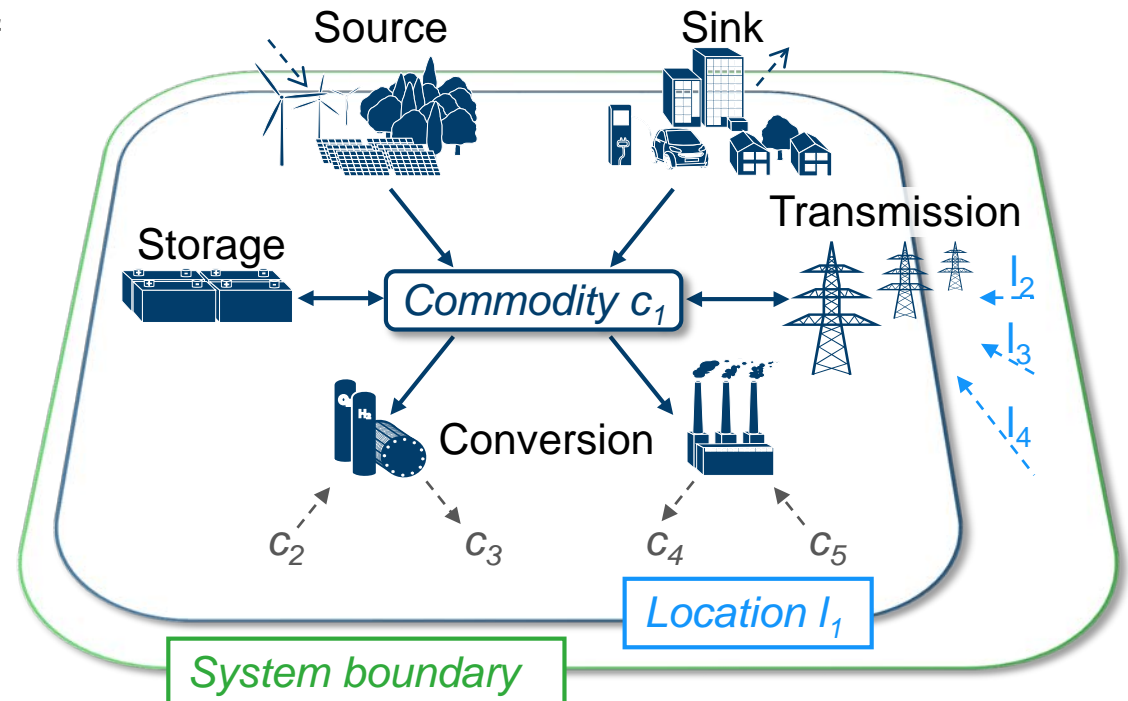
- Sector-coupling: connecting sectors to implement decarbonization strategies
- I.e.: Power-to-Gas, Power-to-Heat, Power-to-Chemicals

# How can we model such scenarios?

How would you model a single pumped hydro power plant?

## Mathematical abstraction of the energy system:

- Spatially and temporally discretization of commodity balance equations
- Storage conservation equation
- Balance contribution terms
  - Source and sink terms
  - Transmission term
  - Storage term
  - Conversion term
- (In)equalities for technical, economical and ecological boundary conditions



## Determine variables using optimization

- Variables are associated with costs → minimize cost (objective)
- Balances and boundary conditions → modeled with constraints

## Energy system model

- Generic equations filled with specific parameters → An energy system model
- Optimized variables reflect the optimal design and operation of the modelled energy system

# How can we determine the cost-optimal solution of these models?

## Implementation of the models into generic energy system optimization frameworks:

There are a number of great energy system optimization frameworks out there! In this tutorial, we will present FINE, a Framework for Integrated Energy system assessment, implemented in Python.

### ♥ Why are we proud of FINE? ♥

- FINE is open-source available at <https://github.com/FZJ-IEK3-VSA/FINE> or as an easy to install Python module
- FINE is licensed with an MIT license, a permissive free software license
- FINE is well documented on the “Read the Docs” website: <https://vsa-fine.readthedocs.io/en/latest/>
- FINE has a large developer base which is not only located in Jülich Research Center but spans across several institutions
- FINE is constantly evolving:
  - Documentation is being updated and expanded to make FINE easy to use
  - New modeling features are added to tackle the research questions of tomorrow
  - New plotting functions are integrated to make a beautiful visualization of your results possible
  - User and data interfaces are improved and added to provide easy access to code and data

#### FINE - Framework for Integrated Energy System Assessment

The FINE python package provides a framework for modeling, optimizing and assessing energy systems. With the provided framework, systems with multiple regions, commodities and time steps can be modeled. Target of the optimization is the minimization of the total annual cost while considering technical and environmental constraints. Besides using the full temporal resolution, an interconnected typical period storage formulation can be applied, that reduces the complexity and computational time of the model.

# How can FINE be used? / Agenda of this Tutorial

## 1. Software installation

- A comprehensive installation guide will be presented to set up a working environment for FINE
- The setup includes the installation of a Python distribution, the modification of a working environment and, if desired, the installation of an additional, fast performing optimization solver

## 2. Using FINE with a Jupyter notebook (or a Python script)

- A “close to code” approach to use FINE is via a Jupyter notebook or a basic Python script
- A Jupyter notebook provides a very easy access to code
- An example will be explained in detail

## 3. Using FINE with an Excel only

- Does not require any expertise in coding
- Short example will be presented

## 4. Short presentation of existing applications of FINE in literature

## 5. Summary and concluding remarks

## 1. **Software installation**

2. Using FINE with a Jupyter notebook (or a Python script)
3. Using FINE with an Excel only
4. Short presentation of existing applications of FINE in literature
5. Summary and concluding remarks

# Software Installation

# Installation Python (for Windows)

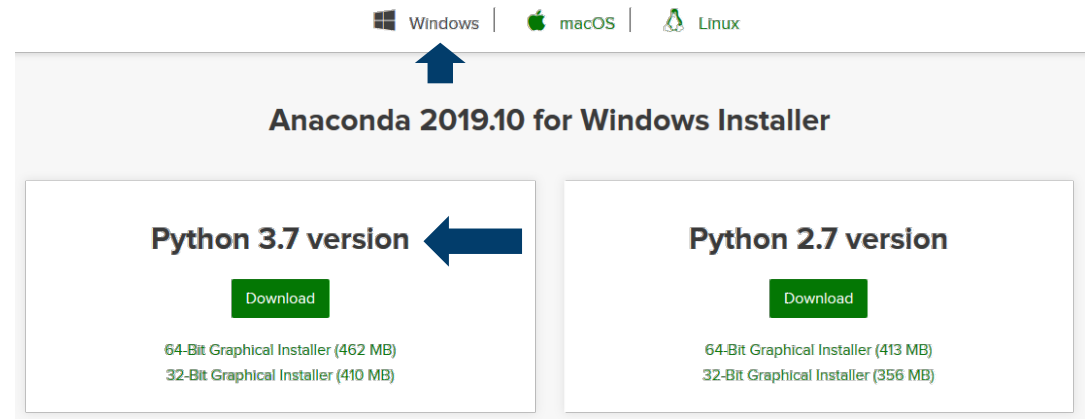
## Download Anaconda:

1. Download a Python distribution ( $\geq 3.6$ ) at:

<https://www.anaconda.com/distribution/>

2. Execute the downloaded file and select
  - a) **Next**
  - b) **I agree** (if you agree)
  - c) either **Just me (recommended)** or **All users (requires admin privileges)**
  - d) an install location in a destination folder of your liking
  - e) **Add Anaconda to my PATH environment variables\***
  - f) **Install**

\* If not selected, all conda commands have to be run exclusively from the anaconda prompt



Let us know if/ where this installation instruction could be improved!



# Installation FINE (for Windows)

## Create an environment on your computer

- Open up the command prompt: (windows key → type `cmd` and hit `Enter`)
- Navigate to the folder where the `environment.yml` file is located
  - If your folder is, for example, located on drive E and your command prompt shows that you are on drive C type `E` and hit `Enter` to switch the drive
  - type `cd pathToFolder` and hit `Enter` (e.g. `pathToFolder` =E:\Notebooks\FINE\_webinar)
- Type `conda env create -n envFINE -f environment.yml` and hit `Enter`\*
- Type, depending on the operating system, `activate envFINE` or `conda activate envFINE` and hit `Enter` (type `deactivate` or `conda deactivate` and hit `Enter` for exiting the environment)

Let us know if/ where this installation instruction could be improved!

\* this installs jupyter/ notebook to access a web application for tangible code interactions and visualizations, the optimization solver GLPK, geopandas for geo-referenced plotting, a specific version of pandas for data analysis and FINE

# Installation Gurobi (for Windows & Academic Institutions)

## Register at the Gurobi website:

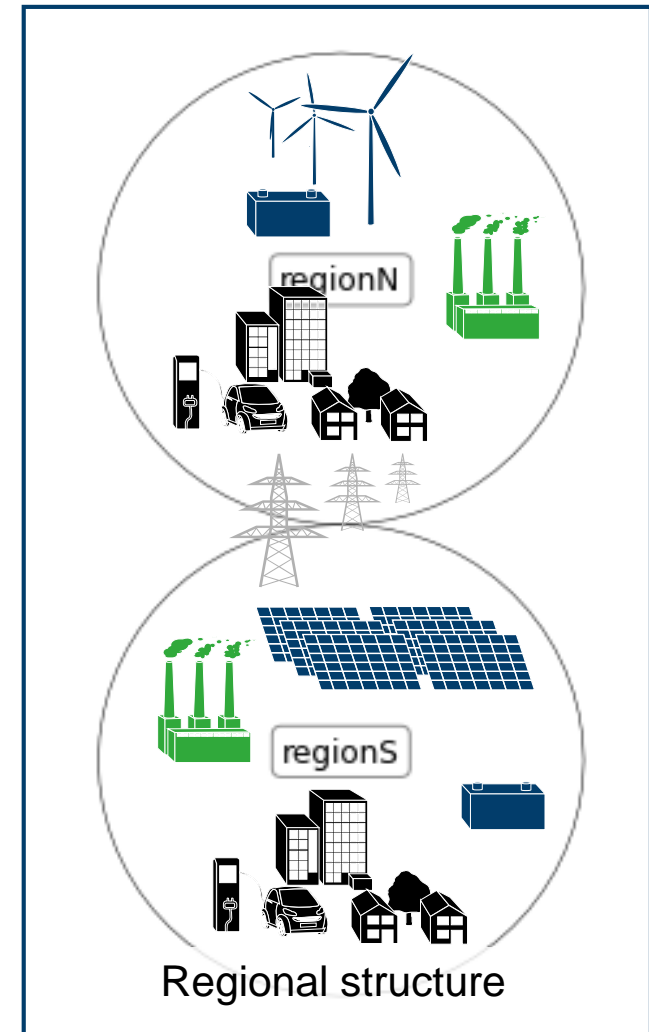
1. <https://www.gurobi.com/>
2. Click on [Register](#)
3. Fill out your contact info and select [Academic](#) (if applicable)
4. Download the latest version of Gurobi at <https://www.gurobi.com/downloads/> (choose the current 64-bit Windows version)
5. Execute the downloaded file and select (if applicable)
  - a) [I accept the terms in the License Agreement](#)
  - b) a destination folder of your liking
  - c) [Install](#)

Let us know if/ where this installation instruction could be improved!

## Request an academic license (if applicable):

1. Go to <https://www.gurobi.com/downloads/end-user-license-agreement-academic/>
2. Once you accepted the terms, you get a license ID which you can use to activate Gurobi: copy the line at the bottom of the page starting with [grbgetkey](#), hit the Windows key, type [cmd](#) and then paste the line.
3. A command prompt window will open and you can either hit enter or select a different directory to store your license key and hit Enter once more. Then, the license will be implemented.

1. Software installation
2. **Using FINE with a Jupyter notebook (or a Python script)**
3. Using FINE with an Excel only
4. Short presentation of existing applications of FINE in literature
5. Summary and concluding remarks

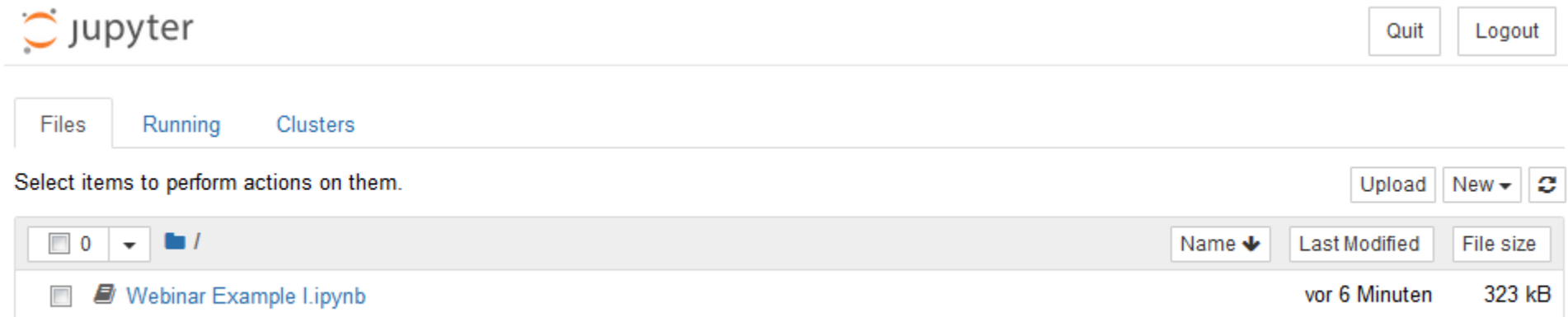


# Jupyter Notebook Example

# Jupyter Notebook – A Quick Introduction (Windows) I

## Starting a Jupyter Notebook:

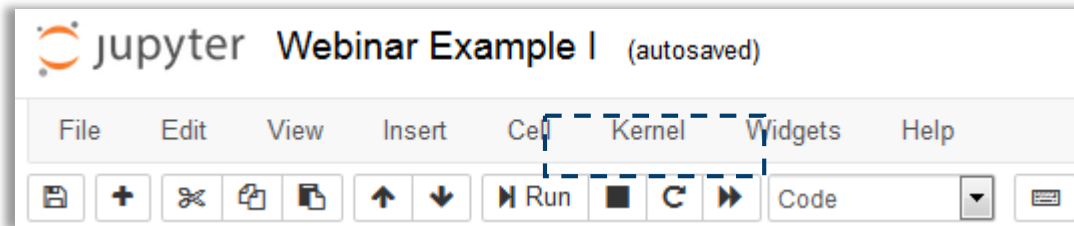
1. Hit the [windows key](#), type [cmd](#) and hit [Enter](#)
2. Type [X:](#) and hit [Enter](#) (X is the drive where your folder is located, e.g. E:)
3. Type [cd pathToFolder](#) and hit [Enter](#) (e.g. [pathToFolder](#) =E:\Notebooks\FINE\_webinar)
4. Type [activate envFINE](#) and hit [Enter](#) (requires that the environment envFINE is set up and configured)
5. Type [jupyter notebook](#) and hit [Enter](#)
6. Create a new notebook/ select an existing one (select the provided notebook “Webinar Example I”)



# Jupyter Notebook – A Quick Introduction (Windows) II

## Working with a Jupyter Notebook:

- Hitting *Shift + Enter* and *Control + Enter* execute a cell (the former also jumps to a new cell)
- The execution of a cell leads to its code being executed → parameters are initialized, processed and stored so that consecutively executed cells can use them
- In the navigation bar, the options “Cell” and “Kernel” provide several useful options when running the notebook (e.g. “Restart & Run All” clears all previous output and runs the entire notebook)



Call the documentation of the EnergySystemModel class in FINE

- *Esc + m / Esc + y* turn a code cell into a text cell and vice versa
- If a function/ object from a python package is called with a “?” its documentation is displayed
- If you get a warning or error message, first try to read the last line of it and then work your way through the rest of it
- Note: Deleting cells can be undone in the edit options, but changes within a cell cannot be undone automatically.

```
In [3]: fn.EnergySystemModel?

Init signature:
fn.EnergySystemModel(
    locations,
    commodities,
    commodityUnitsDict,
    numberOfTimeSteps=8760,
    hoursPerTimeStep=1,
    costUnit='1e9 Euro',
    lengthUnit='km',
    verboseLogLevel=0,
)
Docstring:
EnergySystemModel class
```

# Jupyter Notebook – Grasping FINE's Functionalities

You want to learn about available functions/ input parameters in FINE?

- Checkout <https://vsa-fine.readthedocs.io/en/master/> for FINE's documentation!



- Input parameters are described in detail for each object at: <https://vsa-fine.readthedocs.io/en/master/packageDoc.html>

**Required arguments:**

Parameters: `commodity` (*string*) – to the component related commodity.

**Default arguments:**

Parameters:

- `chargeRate` ( $0 \leq \text{float} \leq 1$ ) – ratio of the maximum storage inflow (in commodityUnit/hour) to the storage capacity (in commodityUnit). Example:
  - A hydrogen salt cavern which can store 133 GWh<sub>H2\_LHV</sub> can be charged 0.45 GWh<sub>H2\_LHV</sub> during one hour. The chargeRate thus equals 0.45/133 1/h.

Docs » Python package description [Edit on GitHub](#)

## Python package description

The python package description gives a general description of the code itself. It describes the package structure (modules and classes, with their functions and variables).

Contents:

- `EnergySystemModel` class
- `Components`
  - `Component` class
  - `Source` and `Sink` classes
  - `Conversion` class
  - `Transmission` class
  - `Storage` class
  - `LinearOptimalPowerFlow` class
- `Input Output Management`
  - Standard inputs and outputs

# Jupyter Notebook Example – Structure

Structure of the notebook/ a classical FINE workflow:

## **FINE Webinar Part I: 2-nodal Electricity Supply System**

In this application of the FINE framework, an energy supply system, consisting of two-regions, is modeled and optimized.

The workflow is structured as follows:

- Required packages are imported
- An energy system model instance is created
- Commodity sources are added to the energy supply system model
- Commodity conversion components are added to the energy supply system model
- Commodity storages are added to the energy supply system model
- Commodity transmission components are added to the energy supply system model
- Commodity sinks are added to the energy supply system model
- The energy supply system model is optimized
- Selected optimization results are presented

# Jupyter Notebook Example – Import Packages

## Import required packages

The FINE framework is imported which provides the required classes and functions for modeling the energy system.

```
import FINE as fn          # Provides objects and functions to model an energy system
import pandas as pd        # Used to manage data in tables
import geopandas as gpd    # Used to display geo-referenced plots
import shapely as shp      # Used to generate geometric objects
import numpy as np         # Used to generate random input data
np.random.seed(42)         # Sets a "seed" to produce the same random input data in each model run
```

- The `import FINE as fn` command calls the FINE package which provides objects and functions to model an energy system (these can be called via: `fn.XYZ(a=b)`)
- The other packages provide additional function which are used to generate import data or display output (these can vary for each individual modeling case)



# Jupyter Notebook Example – Model an energy system

## - Create an energy system model instance

### Create an energy system model instance

The structure of the energy supply system model is given by the considered locations, commodities, the number of time steps as well as the hours per time step.

The commodities are specified by a unit (i.e. 'GW\_electric', 'GW\_naturalGas\_lowerHeatingValue', 'Mio. t CO<sub>2</sub>/h') which can be given as an energy or mass unit per hour. Furthermore, the cost unit and length unit are specified.

Define input  
parameters

```
# Input parameters
locations = {'regionN', 'regionS'}
commodityUnitDict = {'electricity': r'GW$_{el}$', 'naturalGas': r'GW$_{CH_{4}},LHV}$',
                    'CO2': r'Mio. t$_{CO_2}$ /h'}
commodities = {'electricity', 'naturalGas', 'CO2'}
numberOfTimeSteps, hoursPerTimeStep = 8760, 1
costUnit, lengthUnit = '1e6 Euro', 'km'

# Code
esM = fn.EnergySystemModel(locations=locations, commodities=commodities,
                           numberOfTimeSteps=numberOfTimeSteps, commodityUnitsDict=commodityUnitDict,
                           hoursPerTimeStep=hoursPerTimeStep, costUnit=costUnit, lengthUnit=lengthUnit, verboseLogLevel=1)
```

Call `EnergySystemModel` class

Specify  
keyword  
arguments

The techno-economic parameters specified during the model setup have to be specified in reference to the commodities, cost and length units (e.g. 1e6 Euro/GW<sub>el</sub>, 1e6 Euro/(GW<sub>el</sub>·km), 1e6 Euro/(GW<sub>CH<sub>4</sub></sub>·h))

# Jupyter Notebook Example – Model an energy system

## - Add source components I

### Add source components

Source components generate commodities across the energy system's virtual boundaries.

```
# Input parameters
name, commodity = 'Wind turbines', 'electricity'
hasCapacityVariable = True
operationRateMax = pd.DataFrame([[np.random.beta(a=2,b=7.5), np.random.beta(a=2,b=9)]
                                for t in range(8760)],
                                index=range(8760), columns=['regionN', 'regionS'])
capacityMax = pd.Series([200, 100], index=['regionN', 'regionS'])
investPerCapacity, opexPerCapacity = 1200, 1200*0.02
interestRate, economicLifetime = 0.08, 20

# If data should be read from an excel file:
writer = pd.ExcelWriter('windTurbineProfile.xlsx') # writes data to an excel file
operationRateMax.to_excel(writer)                  # (not required if excel file
writer.save()                                     # already exists)
operationRateMax = pd.read_excel('windTurbineProfile.xlsx') # reads an excel file located in
                                                            # the current working directory

# Code
esM.add(fn.Source(esM=esM, name=name, commodity=commodity, hasCapacityVariable=hasCapacityVariable,
                  operationRateMax=operationRateMax, capacityMax=capacityMax, investPerCapacity=investPerCapacity,
                  opexPerCapacity=opexPerCapacity, interestRate=interestRate, economicLifetime=economicLifetime))
```

Random max. capacity factor  
time series for each region

pd.DataFrame/ pd.Series  
creates a two-/ one-  
dimensional input data  
object (dimension(s):  
spatial/ temporal)

Add to EnergySystemModel class

Call `Source`  
class and  
specify  
keyword  
arguments

The boolean parameter `hasCapacityVariable` specifies if a component has a “physical” capacity (wind turbine → True)

# Jupyter Notebook Example – Model an energy system

## - Add source components II

```
# Input parameters
name, commodity = 'PV', 'electricity'
hasCapacityVariable = True
dailyProfileSimple = [0,0,0,0,0,0,0,0,0.05,0.15,0.2,0.4,0.8,0.7,0.4,0.2,0.15,0.05,0,0,0,0,0,0]
operationRateMax = pd.DataFrame([[u,u] for day in range(365) for u in dailyProfileSimple],
                                index=range(8760), columns=['regionN', 'regionS'])
capacityMax = pd.Series([100, 100], index=['regionN', 'regionS'])
investPerCapacity, opexPerCapacity = 800, 800*0.02
interestRate, economicLifetime = 0.08, 25

# Code
esM.add(fn.Source(esM=esM, name=name, commodity=commodity, hasCapacityVariable=hasCapacityVariable,
                 operationRateMax=operationRateMax, capacityMax=capacityMax, investPerCapacity=investPerCapacity,
                 opexPerCapacity=opexPerCapacity, interestRate=interestRate, economicLifetime=economicLifetime))
```

```
# Input parameters
name, commodity = 'Natural gas import', 'naturalGas'
hasCapacityVariable = False
commodityCost = 0.03

# Code
esM.add(fn.Source(esM=esM, name=name, commodity=commodity, hasCapacityVariable=hasCapacityVariable,
                 commodityCost=commodityCost))
```

Add a cell to the Jupyter Notebook (+) and check out what the `operationRateMax` and the `capacityMax` parameters look like (type in the name of the parameter and execute the cell)

The boolean parameter `hasCapacityVariable` specifies if a component has a “physical” capacity (natural gas import → False)

# Jupyter Notebook Example – Model an energy system

## - Add conversion components

### Add conversion components

Conversion components convert m commodities into n other commodities

```
# Input parameters
name, physicalUnit = 'Gas power plants', r'GW$_{el}$'
commodityConversionFactors = {'electricity':1, 'naturalGas':-1/0.63, 'CO2':201*1e-6/0.63}
hasCapacityVariable=True
investPerCapacity, opexPerCapacity = 650, 650*0.03
interestRate, economicLifetime = 0.08, 30

# Code
esM.add(fn.Conversion(esM=esM, name=name, physicalUnit=physicalUnit,
commodityConversionFactors=commodityConversionFactors, hasCapacityVariable=hasCapacityVariable,
investPerCapacity=investPerCapacity, opexPerCapacity=opexPerCapacity,
interestRate=interestRate, economicLifetime=economicLifetime))
```

Physical unit: The maximum capacity limitations, cost parameters and the operation time series refer to the specified physical unit.

Burning natural gas releases CO<sub>2</sub>

Positive conversion factor:  
produced commodity  
Negative conversion factor:  
consumed commodity

In this example, the physical unit is GW<sub>el</sub>, i.e. the installed capacities refer to the maximal amount of electricity which can be produced within one hour. The cost parameters also refer to the process of producing electricity.

Which parameters must be changed if the physical unit is GW<sub>CH<sub>4</sub></sub>?

# Jupyter Notebook Example – Model an energy system

## - Add storage components

### Add storage components

Storage components can store commodities across time steps.

The self discharge of a storage technology is described in FINE in percent per hour. If the literature value is given in percent per month, e.g. 3%/month, the self discharge per hours is obtained using the equation

$$(1 - \text{selfDischarge}_{\text{hour}})^{30 \cdot 24} = 1 - \text{selfDischarge}_{\text{month}}$$

```
# Input parameters
name, commodity = 'Batteries', 'electricity'
hasCapacityVariable=True
chargeEfficiency, dischargeEfficiency, selfDischarge = 0.95, 0.95, 1-(1-0.03)**(1/(30*24))
chargeRate, dischargeRate = 1, 1,
investPerCapacity, opexPerCapacity = 150, 150*0.01
interestRate, economicLifetime, cyclicLifetime = 0.08, 22, 10000

# Code
esM.add(fn.Storage(esM=esM, name=name, commodity=commodity, hasCapacityVariable=hasCapacityVariable,
chargeEfficiency=chargeEfficiency, cyclicLifetime=cyclicLifetime,
dischargeEfficiency=dischargeEfficiency, selfDischarge=selfDischarge, chargeRate=chargeRate,
dischargeRate=dischargeRate, investPerCapacity=investPerCapacity,
opexPerCapacity=opexPerCapacity, interestRate=interestRate, economicLifetime=economicLifetime))
```

In this example, components are in general modeled with continuous capacities. However, they can also be modeled with discrete values. Moreover, they can be associated with additional binary variables to model, for example, nonlinear cost functions or a minimal plant unit size.

# Jupyter Notebook Example – Model an energy system

## - Add transmission components

### Add transmission components

Transmission components transmit commodities between regions.

```
# Input parameters
name, commodity = 'AC cables', 'electricity'
hasCapacityVariable = True
capacityFix = pd.DataFrame([[0, 30], [30, 0]], columns=['regionN', 'regionS'],
                             index=['regionN', 'regionS'])
distances = pd.DataFrame([[0, 400], [400, 0]], columns=['regionN', 'regionS'],
                           index=['regionN', 'regionS'])
losses=0.0001

# Code
esM.add(fn.Transmission(esM=esM, name=name, commodity=commodity,
                        hasCapacityVariable=hasCapacityVariable, capacityFix=capacityFix,
                        distances=distances, losses=losses))
```

two-dimensional input data objects (dimension: spatial-spatial, in the sense of from region A to region B)

	regionN	regionS
regionN	0	400
regionS	400	0

- Note: The AC cable connection is here modeled as a “simple” transmission technology. However, AC cables can also be modeled with linear power flow equations in FINE (for a 2-nodal system, the modeling approaches equal each other).



# Jupyter Notebook Example – Model an energy system

## - Add sink components

### Add sink components

Sinks remove commodities across the energy system's virtual boundaries.

```
# Input parameters
name, commodity = 'Electricity demand', 'electricity',
hasCapacityVariable = False
dailyProfileSimple = [0.6,0.6,0.6,0.6,0.6,0.7,0.9,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0.9,0.8]
operationRateFix = pd.DataFrame([(u+0.1*np.random.rand())*25, (u+0.1*np.random.rand())*40]
                                for day in range(365) for u in dailyProfileSimple],
                                index=range(8760), columns=['regionN', 'regionS'])

# Code
esM.add(fn.Sink(esM=esM, name=name, commodity=commodity, hasCapacityVariable=hasCapacityVariable,
               operationRateFix=operationRateFix))
```

```
# Input parameters
name, commodity = 'CO2 to enviroment', 'CO2',
hasCapacityVariable=False
commodityLimitID, yearlyLimit = 'CO2 limit', 366*(1-0.8)

# Code
if yearlyLimit > 0:
    esM.add(fn.Sink(esM=esM, name=name, commodity=commodity,
                   hasCapacityVariable=hasCapacityVariable, commodityLimitID=commodityLimitID,
                   yearlyLimit=yearlyLimit))
```

yearlyLimit: Limit the commodity flow across the energy system's virtual boundary to a certain amount per year, e.g. limit the CO<sub>2</sub> emissions.

# Jupyter Notebook Example – Optimize the Model

Optimize the set of equations provided alongside the added components (cost minimization), if required reduce the model's complexity beforehand by using times series aggregation:

## Optimize energy system model

All components are now added to the model and the model can be optimized. If the computational complexity of the optimization should be reduced, the time series data of the specified components can be clustered before the optimization and the parameter `timeSeriesAggregation` is set to `True` in the optimize call.

```
# Input parameters
numberOfTypicalPeriods=30

# Code
esM.cluster(numberOfTypicalPeriods=numberOfTypicalPeriods)
```

For example:  
Consider 30 typical days

Call the cluster function of the  
EnergySystemModel instance

```
Clustering time series data with 30 typical periods and 24 time steps per period...
(0.9800 sec)
```

```
# Input parameters
timeSeriesAggregation=True
solver='glpk'

# Code
esM.optimize(timeSeriesAggregation=timeSeriesAggregation, solver=solver)
```

Any pyomo compatible solver can  
be used here (default is 'gurobi')

Call the optimize function of the  
EnergySystemModel instance

Run the model  
without time series  
aggregation. How  
does the time series  
aggregation effect the  
computation time?



# Jupyter Notebook Example – Optimization Output Options

**FINE provides different output options to support the assessment of the optimization results:**

## Output of selected results

For the assessment of the optimization result, several result output functions are available. They can be categorized into output in form of tables, geo-referenced output visualization and the full time series visualization.

**Examples are:**

1. Output in form of tables
  - a) Optimization summaries of each component class
  - b) Excel file with all optimization summaries and optimal variables
2. Geo-referenced data output (requires a regional shape file; can be manually created)
  - a) Installed capacities within a region
  - b) Installed transmission capacities between regions
  - c) Operation within a region
3. Time series profiles
  - a) Operation profile as a line
  - b) Operation profile as a color map

# Jupyter Notebook Example – Regional Shape File

## Create a regional shape file and visualize it

Information on the geometrical shape of the investigated regions can either be downloaded from a website (e.g. from <https://gadm.org/>) or manually created. In this notebook, the geometries are manually created.

```
# Create two circles, representing the two regions, and store their geometries in a shape file
shpRegionS = shp.geometry.Point(0.5,0.5).buffer(0.5)
shpRegionN = shp.geometry.Point(0.5,1.5).buffer(0.5)
regionsGdf = gpd.GeoDataFrame({'geometry': [shpRegionN, shpRegionS],
                                'regionName': ['regionN', 'regionS']},
                                index=['regionN', 'regionS'], crs='epsg:3035')

regionsGdf.to_file('regions.shp')

# Create a line, representing the connection between the two regions, and store its geometry in a
# shape file
lines = shp.geometry.LineString([(0.5,0.5), (0.5,1.5)])
linesGdf = gpd.GeoDataFrame({'geometry': [lines, lines], 'loc0': ['regionN', 'regionS'],
                                'loc1': ['regionS', 'regionN']},
                                index=['regionN_regionS', 'regionS_regionN'], crs='epsg:3035')

linesGdf.to_file('lines.shp')

# Visualize the geometric representation of the two regions
fig, ax = fn.plotLocations('regions.shp', indexColumn='regionName', plotLocNames=True)
```

Point objects by via x and y coordinate and the radius

GeoDataFrame holding geometries, region names and the coordinate reference system

LineString via start+end coordinates



The creation of the shape files is not required if existing shape files can be used, in this case, only the last line is required.

# Jupyter Notebook Example – Display Optimization Summaries

## Display optimization summaries

For each modeling class, an optimization summary can be stored and displayed.

```
srcSnkSummary = esM.getOptimizationSummary("SourceSinkModel", outputLevel=1)
display(esM.getOptimizationSummary("SourceSinkModel", outputLevel=2))
```

```
convSummary = esM.getOptimizationSummary("ConversionModel", outputLevel=1)
display(esM.getOptimizationSummary("ConversionModel", outputLevel=2))
```

```
storSummary = esM.getOptimizationSummary("StorageModel", outputLevel=1)
display(esM.getOptimizationSummary("StorageModel", outputLevel=2))
```

```
transSummary = esM.getOptimizationSummary("TransmissionModel", outputLevel=1)
display(esM.getOptimizationSummary("TransmissionModel", outputLevel=2))
```

			regionN	regionS
Component	Property	Unit		
Gas power plants	TAC	[1e6 Euro/a]	220.142	3139.46
	capacity	[GW <sub>el</sub> ]	2.85018	40.6467

Cell output

Component	Property	Unit	LocationIn	regionN	regionS
AC cables	capacity	[GW <sub>el</sub> ]	regionN	NaN	30
			regionS	30	NaN
	operation	[GW <sub>el</sub> *h/a]	regionN	NaN	85932.6
			regionS	7063.1	NaN

## Cell output

Component	Property	Unit	regionN	regionS
Batteries	TAC	[1e6 Euro/a]	825.936	59.8084
	capacity	[GW <sub>el</sub> *h]	50.9685	3.69078
	capexCap	[1e6 Euro/a]	740.482	54.2722
	invest	[1e6 Euro]		
	operationCharge	[GW <sub>el</sub> *h/a]		
	operationDischarge	[GW <sub>el</sub> *h/a]		
	opexCap	[1e6 Euro/a]	76.4528	5.53617

## Cell output

Component	Property	Unit	regionN	regionS
CO2 to enviroment	operation	[Mio. tCO <sub>2</sub> /h*a]	2.85679	70.3432
Electricity demand	operation	[GW <sub>el</sub> *h/a]	205208	328596
Natural gas import	TAC	[1e6 Euro/a]	426.387	10499
	commodCosts	[1e6 Euro/a]	426.387	10499
	operation	[GW <sub>CH<sub>4</sub>,LHV</sub> *h/a]	14212.9	349966
PV	TAC	[1e6 Euro/a]	0	2633.57
	capacity	[GW <sub>el</sub> ]	0	28.9585
	capexCap	[1e6 Euro/a]	0	2170.24
	invest	[1e6 Euro]	0	23166.8
	operation	[GW <sub>el</sub> *h/a]	0	32766.5
	opexCap	[1e6 Euro/a]	0	463.336
Wind turbines	TAC	[1e6 Euro/a]	22024.9	0
	capacity	[GW <sub>el</sub> ]	150.626	0
	capexCap	[1e6 Euro/a]	18409.9	0
	invest	[1e6 Euro]	180751	0
	operation	[GW <sub>el</sub> *h/a]	277595	0
	opexCap	[1e6 Euro/a]	3615.01	0

NaN stands for "not a number" and is written here because the connection does not exist

# Jupyter Notebook Example – Display Plots

## - Wind turbines

### Wind turbines

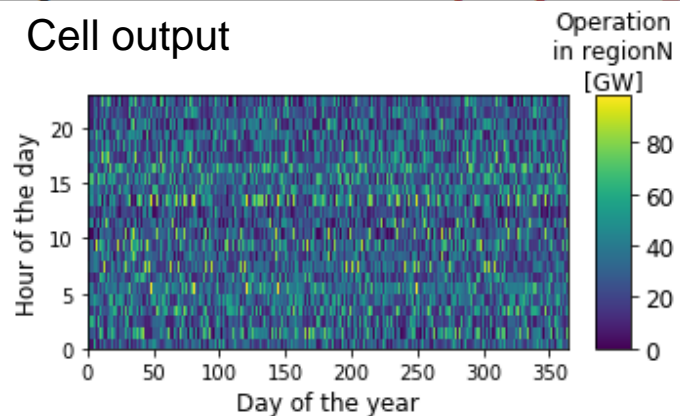
```
# If wind turbines are built, their capacities are displayed in a geo-referenced plot
if srcSnkSummary.loc(['Wind turbines', 'capacity', '[GW$_{el}$']]).sum() > 0:
    fig, ax = fn.plotLocationalColorMap(esM, 'Wind turbines', 'regions.shp', 'regionName',
    perArea=False, xlabel='Capacity\nn[GW]\n', figsize=(3,4))
else:
    print('No wind turbines built.')

# If wind turbines are built in regionN, their operation is displayed as heatmap
if srcSnkSummary.loc(['Wind turbines', 'capacity', '[GW$_{el}$']], 'regionN'] > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'Wind turbines', 'regionN', figsize=(5,3),
    xlabel='Hour of the day', ylabel='Day of the month', zlabel='Operation\nnin regionN\nn[GW]')

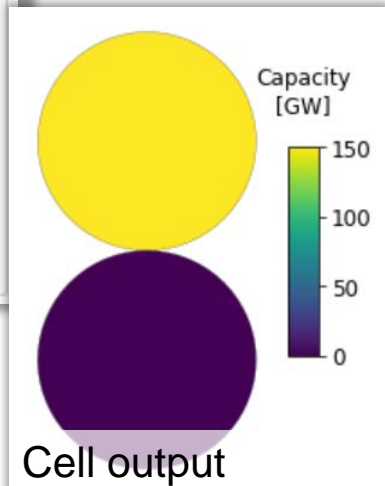
# If wind turbines are built in regionS, their operation is displayed as heatmap
if srcSnkSummary.loc(['Wind turbines', 'capacity', '[GW$_{el}$']], 'regionS'] > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'Wind turbines', 'regionS', figsize=(5,3),
    xlabel='Hour of the day', ylabel='Day of the month', zlabel='Operation\nnin regionS\nn[GW]')
```

Check if capacities are built first

Cell output



Wind turbines are preferably built in the North (higher full load hours)



# Jupyter Notebook Example – Display Plots

## - PV

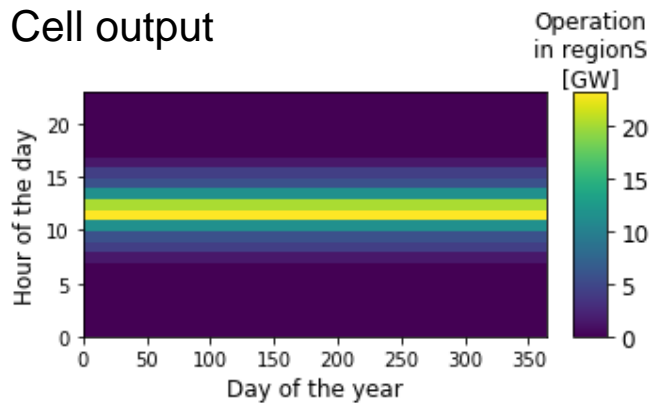
### PV systems

```
# If PV systems are built, their capacities are displayed in a geo-referenced plot
if srcSnkSummary.loc[('PV','capacity','[GW$_{el}$']')].sum() > 0:
    fig, ax = fn.plotLocationalColorMap(esM, 'PV', 'regions.shp', 'regionName', perArea=False,
        xlabel='Capacity\n[GW]\n', figsize=(3,4))
else:
    print('No PV systems built.')

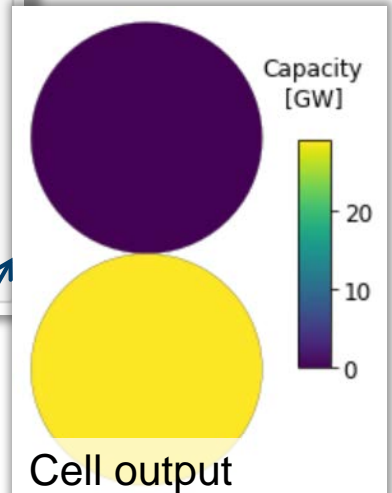
# If PV systems are built in regionS, their operation is displayed as heatmap
if srcSnkSummary.loc[('PV','capacity','[GW$_{el}$']'),'regionN'] > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'PV', 'regionN', figsize=(5,3),
        xlabel='Hour of the day', ylabel='Day of the month', zlabel='Operation\nin regionN\n[GW]')

# If PV systems are built in regionS, their operation is displayed as heatmap
if srcSnkSummary.loc[('PV','capacity','[GW$_{el}$']'),'regionS'] > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'PV', 'regionS', figsize=(5,3),
        xlabel='Hour of the day', ylabel='Day of the month', zlabel='Operation\nin regionS\n[GW]')
```

Cell output



PV systems are modeled with daily profiles and provide renewable electricity generation in the South



# Jupyter Notebook Example – Display Plots

## - Gas Power Plant

### Gas power plants

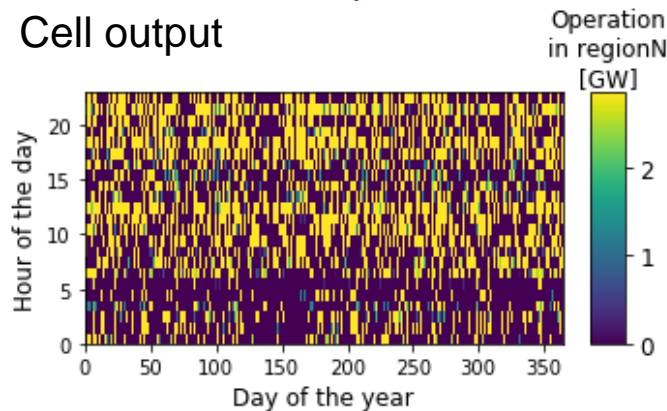
```
# If CCGT plants are built, their capacities are displayed in a geo-referenced plot
if convSummary.loc(['Gas power plants', 'capacity', '[GW$_{el}$'])].sum() > 0:
    fig, ax = fn.plotLocationalColorMap(esM, 'Gas power plants', 'regions.shp', 'regionName',
    perArea=False, zlabel='Capacity\n[GW]\n', figsize=(3,4))
else:
    print('No CCGT plants built.')

# If CCGT plants are built in regionS,
if convSummary.loc(['Gas power plants', 'capacity', '[GW$_{el}$'])].sum() > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'Gas power plants', 'regionN', 'regionN',
    xlabel='Hour of the day', ylabel='Operation\nin regionN\n[GW]')

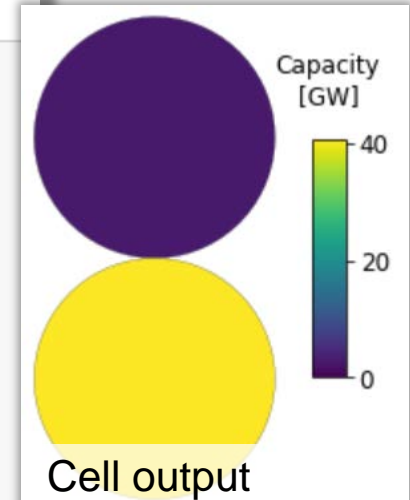
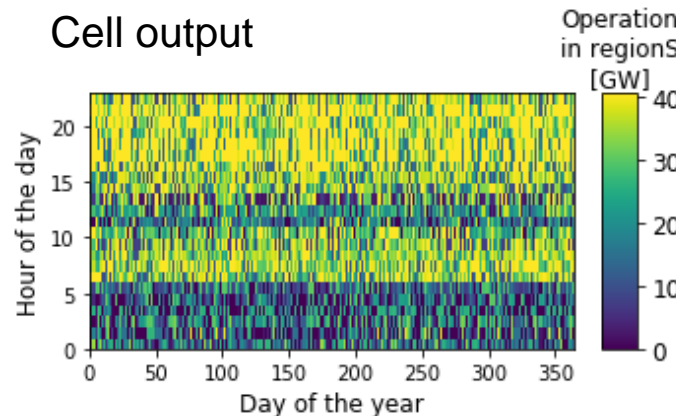
# If CCGT plants are built in regionS,
if convSummary.loc(['Gas power plants', 'capacity', '[GW$_{el}$'])].sum() > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'Gas power plants', 'regionS', 'regionS',
    xlabel='Hour of the day', ylabel='Operation\nin regionS\n[GW]')
```

Power plants  
operation in the  
North: el. demand  
high + (wind) lull

Cell output



Cell output



Power plants  
operation in the  
South: el. demand  
high & generation by  
PV systems low



# Jupyter Notebook Example – Display Plots

## - Batteries

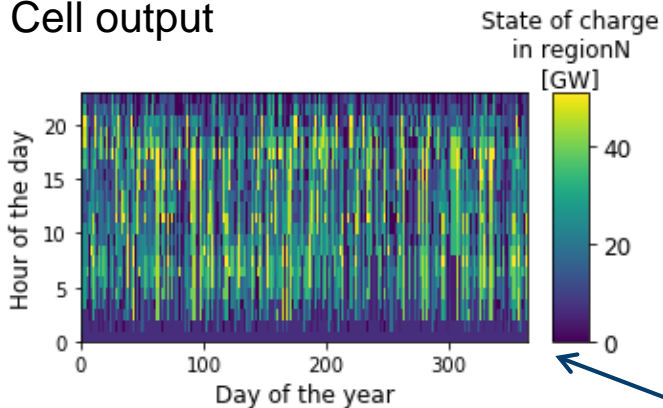
### Batteries

```
# If batteries are built, their capacities are displayed in a geo-referenced plot
if storSummary.loc[('Batteries', 'capacity', '[GW${el}$*h')].sum() > 0:
    fig, ax = fn.plotLocationalColorMap(esM, 'Batteries', 'regions.shp', 'regionName',
        perArea=False, xlabel='Capacity\n[GW]\n', figsize=(3,4))
else:
    print('No batteries built.')

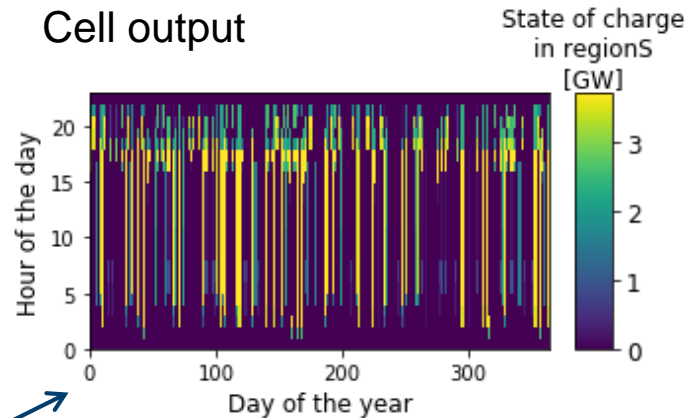
# If batteries are built in regionS, their storage inventory is displayed as heatmap
if storSummary.loc[('Batteries', 'capacity', '[GW${el}$*h'), 'regionN'] > 0:
    fig, ax = fn.plotOperationColorMap(esM, 'Batteries', 'regionN', figsize=(5,3),
        xlabel='Hour of the day', ylabel='Day of the month',
        xlabel='State of charge\nin regionN\n[GW]',
        variableName='stateOfChargeOperationVariablesOptimum')

# If batteries are built in regionS, their storage inventory is displayed as heatmap
if s
```

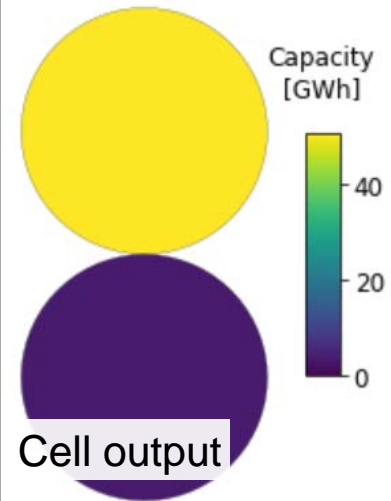
Cell output



Cell output



Daily storage



# Jupyter Notebook Example – Display Plots

## - AC cables and electricity demand

### AC cables

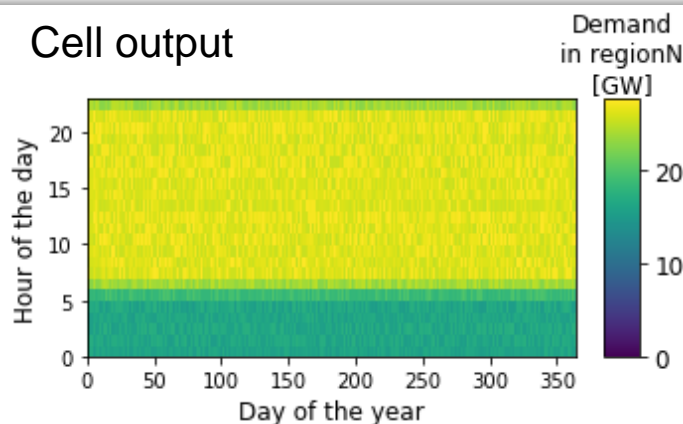
```
# The built AC cable capacities are displayed
fig, ax = fn.plotLocations('regions.shp', indexColumn='regionName')
fig, ax = fn.plotTransmission(esM, 'AC cables', 'lines.shp', loc0='loc0', loc1='loc1',
                             fig=fig, ax=ax, cbHeight=0.4,)
```

### Electricity demand

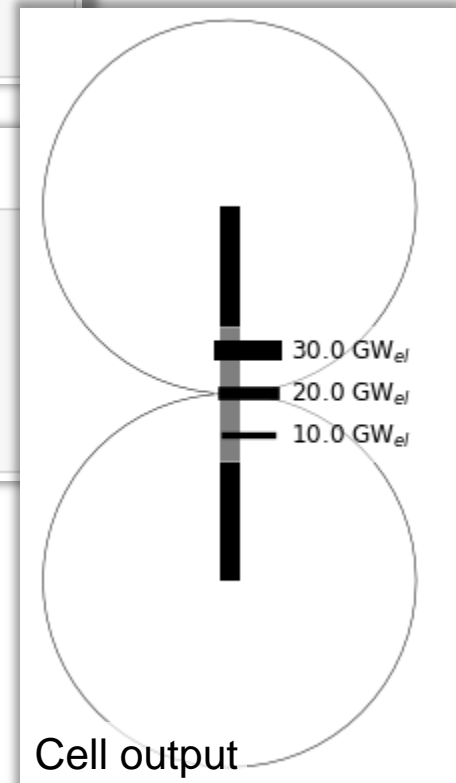
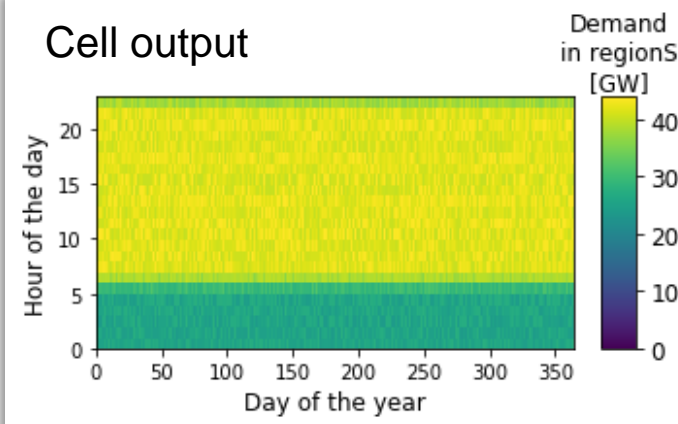
```
# The electricity demand time series in regionN is displayed
fig, ax = fn.plotOperationColorMap(esM, 'Electricity demand', 'regionN', figsize=(5,3),
                                   xlabel='Hour of the day', ylabel='Day of the month', zlabel='Demand\nin regionN\n[GW]')

# The electricity demand time series in regionS is displayed
fig, ax = fn.plotOperationColorMap(esM, 'Electricity demand', 'regionS', figsize=(5,3),
                                   xlabel='Hour of the day', ylabel='Day of the month', zlabel='Demand\nin regionS\n[GW]')
```

Cell output



Cell output





# Jupyter Notebook Example – Exploring the Scenario

## Explore and assess the scenario and its sensitivities. Examples:

- What happens if the reduction target is changed?
- What happens if the number of typical days is changed?
- How effective/ reasonable is time series clustering for the considered time series data in this model?
- How does a varying natural gas cost affect the total annual cost of the system?
- What full load hours do the wind turbines/ PV systems/ power plants have?
- What happens if the optimization solver is switched?
- What happens if the invest of wind turbines, PV systems or batteries are halved/ doubled respectively?
- Is the system feasible when batteries and/or AC cables are removed? Vary in this context also the reduction target.
- Remove the yearly limit of the CO<sub>2</sub> emissions and associated them instead with an emission cost (which parameter has to be added? which ones need to be removed). How much carbon dioxide is emitted if the cost is set to 0 | 50 | 100 | 200 | 300 €/t<sub>CO<sub>2</sub></sub>?
- Add an electrolyzer, a H<sub>2</sub>-storage, a H<sub>2</sub>-pipeline connection and a H<sub>2</sub>-demand and investigate the resulting supply system.
- Which components could be added/ adapted to model a more comprehensive/ realistic energy system?

1. Software installation
2. Using FINE with a Jupyter notebook (or a Python script)
- 3. Using FINE with an Excel only**
4. Short presentation of existing applications of FINE in literature
5. Summary and concluding remarks

## Excel Example

# Running a Scenario from Excel with a Batch File

**A scenario can also be run via an Excel file and a batch script:**

- To do so, all installation steps have to be executed beforehand
- Once everything is installed, an Excel file of a prescribed format and the batch file (e.g. scenarioInput.xlsx & run.bat) have to be placed in the same folder. There, the batch file can be executed.

**Pros:**

- An expertise in the execution of a Jupyter Notebook/ programming is not required
- The input data is stored in a compressed, comprehensive form

**Restrictions:**

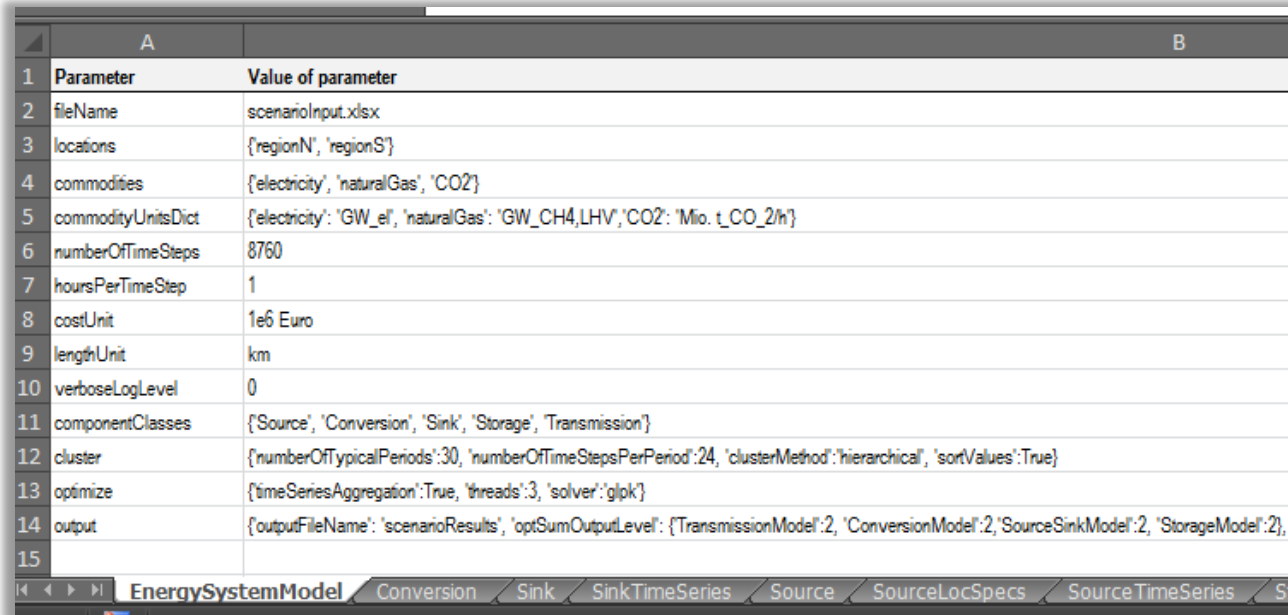
- The model run from Excel is in an early development state
- Output visualization features are currently not available when the model is run from Excel
- Currently, for a stable run of complex scenarios, the usage of a Jupyter notebook or a Python script is suggested

**Outlook:** A FINE GUI with Excel, which does not require the manual installation of Python or an optimization solver, is under development

# Running a Scenario from Excel with a Batch File – Data Input

## 1. Open the provided Excel file (scenarioInput.xlsx)

- In the **EnergySystemModel** worksheet, basic information is written such as the spatial and temporal resolution of the model as well as information for the optimizer and the output style
- For each component, three worksheets can be provided, one with general information, one with time-independent, location specific data and one with time dependent, location specific data (e.g. **Source**, **SourceLocSpecs** and **SourceTimeSeries**)
- The tables' column and/ or row names contain keyword arguments of FINE (e.g. **name**, **commodity**, **hasCapacityVariable**, **capacityMax**, **operationRateFix**). These keyword arguments are the same ones as listed and described on the readTheDocs website: <https://vsa-fine.readthedocs.io/en/master/>



The screenshot shows an Excel spreadsheet with two columns: A (Parameter) and B (Value of parameter). The data is as follows:

Parameter	Value of parameter
fileName	scenarioInput.xlsx
locations	{regionN, regionS}
commodities	{electricity, naturalGas, CO2}
commodityUnitsDict	{electricity: 'GW_el', naturalGas: 'GW_CH4,LHV', CO2: 'Mio. t_CO2/h'}
numberOfTimeSteps	8760
hoursPerTimeStep	1
costUnit	1e6 Euro
lengthUnit	km
verboseLogLevel	0
componentClasses	{Source, Conversion, Sink, Storage, Transmission}
cluster	{numberOfTypicalPeriods:30, numberOfTimeStepsPerPeriod:24, clusterMethod:'hierarchical', sortValues:True}
optimize	{timeSeriesAggregation:True, threads:3, solver:'glpk'}
output	{outputFileName: 'scenarioResults', optSumOutputLevel: {TransmissionModel:2, ConversionModel:2, SourceSinkModel:2, StorageModel:2},

The worksheet tabs at the bottom are: EnergySystemModel, Conversion, Sink, SinkTimeSeries, Source, SourceLocSpecs, SourceTimeSeries, and St.

# Running a Scenario from Excel with a Batch File – Run the Model

## 2. Double-click on the provided batch file (run.bat)

- a) A command prompt window will open and ask for the name of the input data file and your Python environment (the environment `envFINE` was set up at the beginning of this tutorial)

```
Enter the name of the Excel file name which contains the model input data or hit [Enter] when the data is stored in scenarioInput.xlsx:  
Enter the name of your conda environment or hit [Enter] when your regular Python version should be used:envFINE
```

- b) Then, automatically, the input data is read,  
the model is initialized

```
Clustering time series data with 30 typical periods and 24 time steps per period...  
      (4.1646 sec)  
Time series aggregation specifications:  
Number of typical periods:30, number of time steps per periods:24  
Declaring sets, variables and constraints for SourceSinkModel  
      declaring sets...  
      declaring variables...
```

Initialize model

the optimization is started

```
GLPSOL: GLPK LP/MIP Solver, v4.65  
Parameter(s) specified in the command line:  
... (GLPK) ...
```

Start optimization

and an Excel output file is created

```
Writing output to Excel...  
Processing SourceSinkModel ...  
Processing ConversionModel ...  
Processing TransmissionModel ...  
Processing StorageModel ...  
Saving file...
```

Write to excel

# Running a Scenario from Excel with a Batch File – Explore the Results

## 3. Investigate the optimization outputs in the excel file

- For each component class, an optimization summary as well as the optimal values of the time-dependent and time-independent optimal variables are provided.
- If time series aggregation was selected, also the order of the typical periods is written down.

	A	B	C	D	E	F	G	H	I	J	K
1	Component	Property	Unit	regionN	regionS						
2	Batteries	TAC	1e6 Euro/a	825.3437	59.92894						
3		capacity	[GW_el*h/a]	50.93202	3.698219						
4		capexCap	1e6 Euro/a	748.9457	54.38161						
5		invest	[1e6 Euro]	7639.803	554.7329						
6		erationChas	GW_el*h/a	22503.82	826.9594						
7		ationDisch	GW_el*h/a	20303.27	746.0283						
8		opexCap	1e6 Euro/a	76.39803	5.547329						
9											
10											
11											
12											
13											
14											
15											

StorageOptSummary\_1dim Storage\_TDoptVar\_1dim Storage\_TToptVar\_1dim SourceSinkOptSum

(Generic plotting functions are in this case not available but have to be manually created in Excel)

Run the model without time series aggregation. How does the time series aggregation effect the computation time?

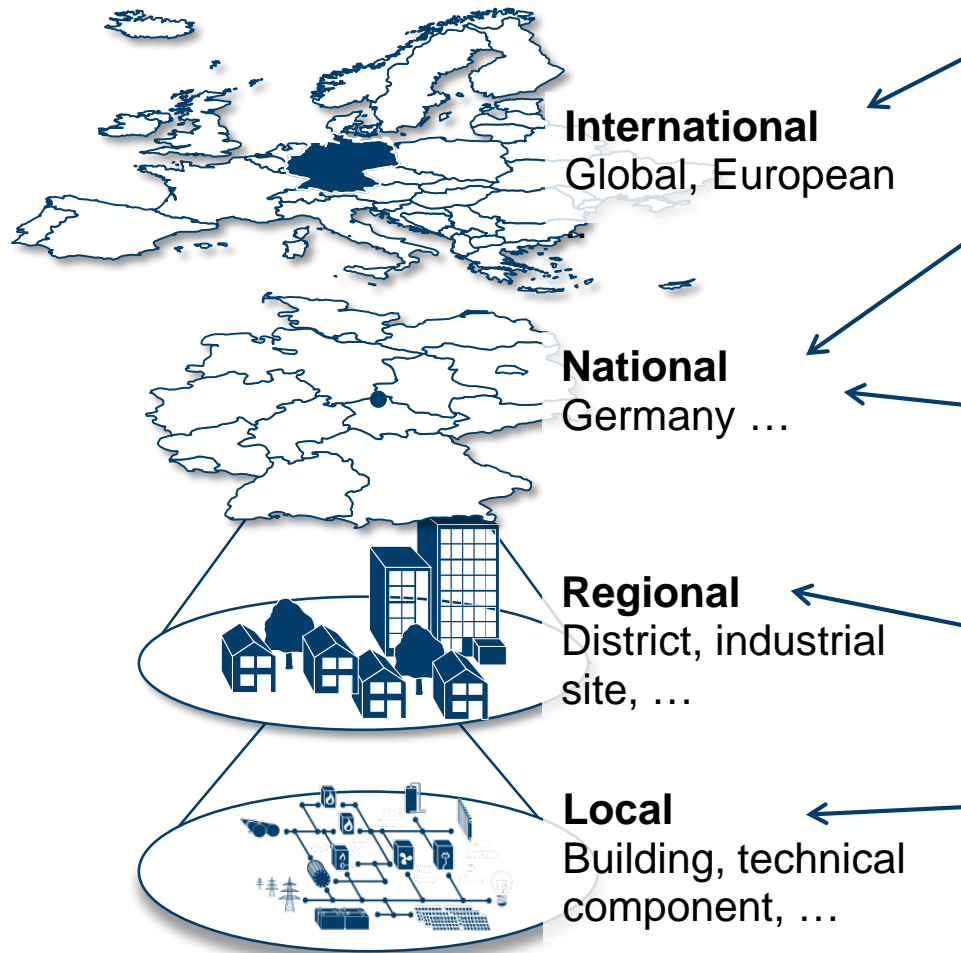
1. Software installation
2. Using FINE with a Jupyter notebook (or a Python script)
3. Using FINE with an Excel only
- 4. Short presentation of existing applications of FINE in literature**
5. Summary and concluding remarks

# Applications of FINE



# Examples

## Application Levels



## Examples

- Caglayan, D.G. et al. *Impact of wind year selection on the design of green hydrogen supply pathways for transport needs*. Preprints, 2019
- Welder, L., et al., *Spatio-temporal optimization of a future energy system for power-to-hydrogen applications in Germany*. Energy, 2018
- Lopion, P. et al. *Cost Uncertainties in Energy System Optimisation Models: A Quadratic Programming Approach for Avoiding Penny Switching Effects*. Preprints, 2019
- Kannengießer et al. *Reducing Computational Load for Mixed Integer Linear Programming: An Example for a District and an Island Energy System*. Energies, 2019
- Kotzur, L., *Future Grid Load of the Residential Building Sector*, in Faculty of Mechanical Engineering. 2018, RWTH Aachen: Aachen.

# Example – Impression of a Europe 2050 Scenario

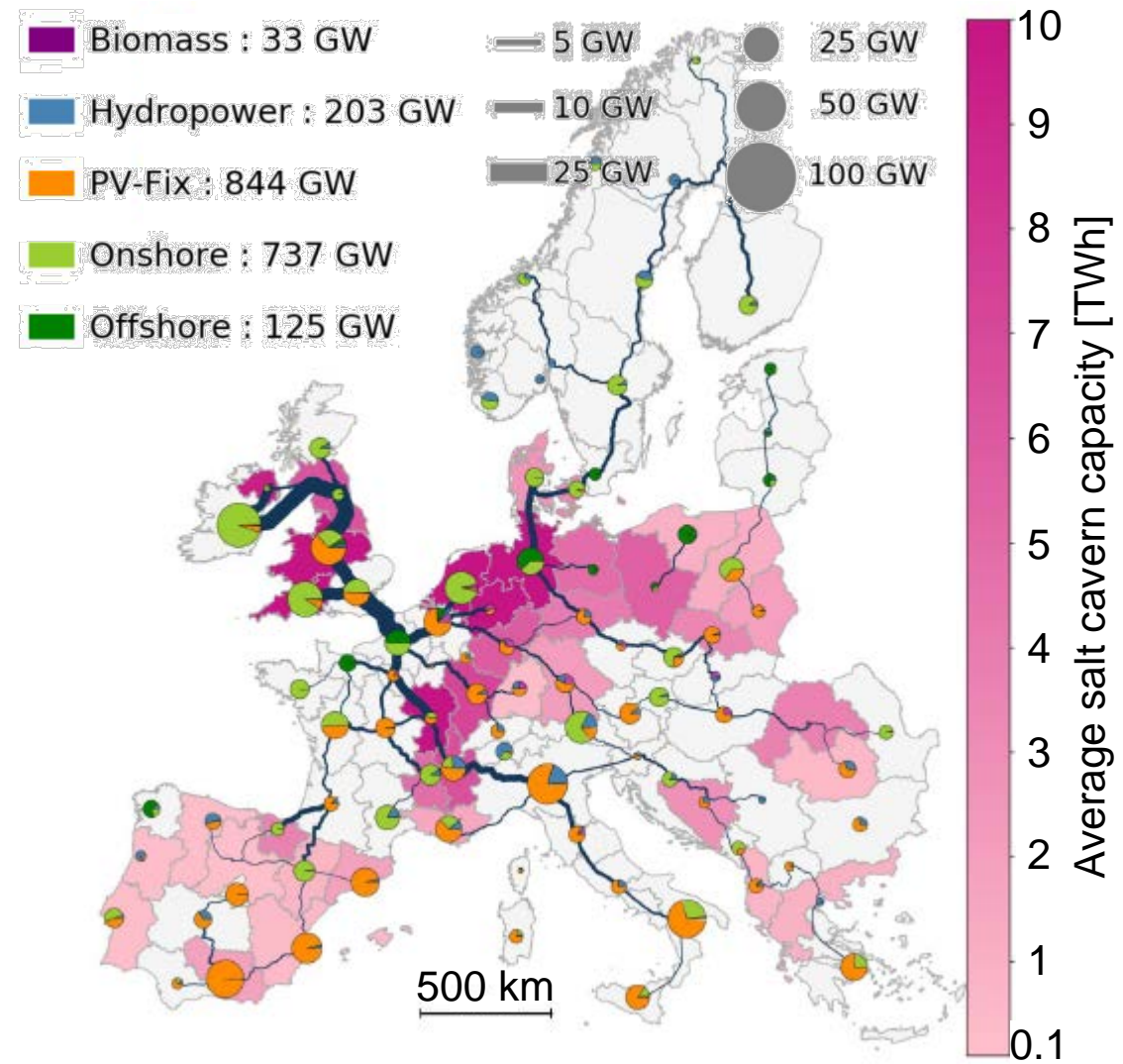
A study by Caglayan, D.G. et al.: *The impact of temporal complexity reduction on a 100% Renewable European Energy System with Hydrogen Infrastructure.*

Preprints, 2019

## FINE-related highlights:

- A high spatial (96 regions) and temporal resolution is modeled (up to 8760 h/a)
- A diverse technology set of cross-linked electricity and hydrogen infrastructure components is considered
- For all technologies, the optimal design and operation can be determined, this includes the design of electricity generation, transmission and storage infrastructure

**Outlook:** Additional standard plotting functions are currently integrated into a new release of FINE

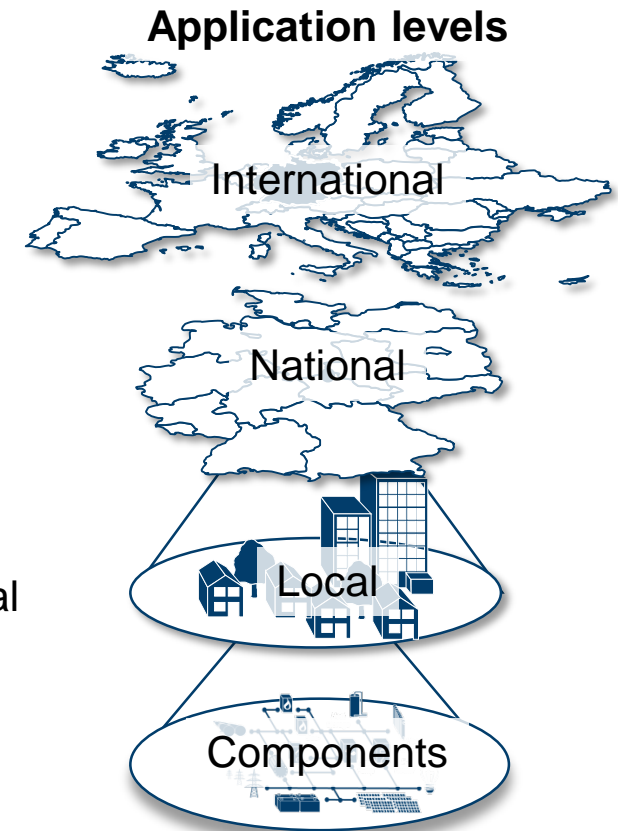


1. Software installation
2. Using FINE with a Jupyter notebook (or a Python script)
3. Using FINE with an Excel only
4. Short presentation of existing applications of FINE in literature
- 5. Summary and concluding remarks**

# Summary and Concluding Remarks

# Summary and Concluding Remarks I

- FINE can be used to generically model energy (supply) system models on various application levels
- With FINE, the cost of a spatially and temporally resolved model is minimized while considering technical and ecological constraints
- For using FINE on your own machine, the installation of a Python distribution is required and was explained step by step in this tutorial
- The installation of alternative commercial optimization software as for example, Gurobi (which has an academic license), is optional
- The online documentation of FINE, which describes all available input parameters, was mentioned (<https://vsa-fine.readthedocs.io/en/master/>)
- Running a FINE model in a Jupyter notebook was presented:
  - A short introduction into Jupyter Notebook was given
  - An example of a simple energy (supply) system model run in a notebook was presented in detail
  - A selection of output functions for geo-referenced plotting and time series visualization was presented
  - Typical question which arise in the context of energy system modeling and assessment were raised → these questions can be answered with FINE



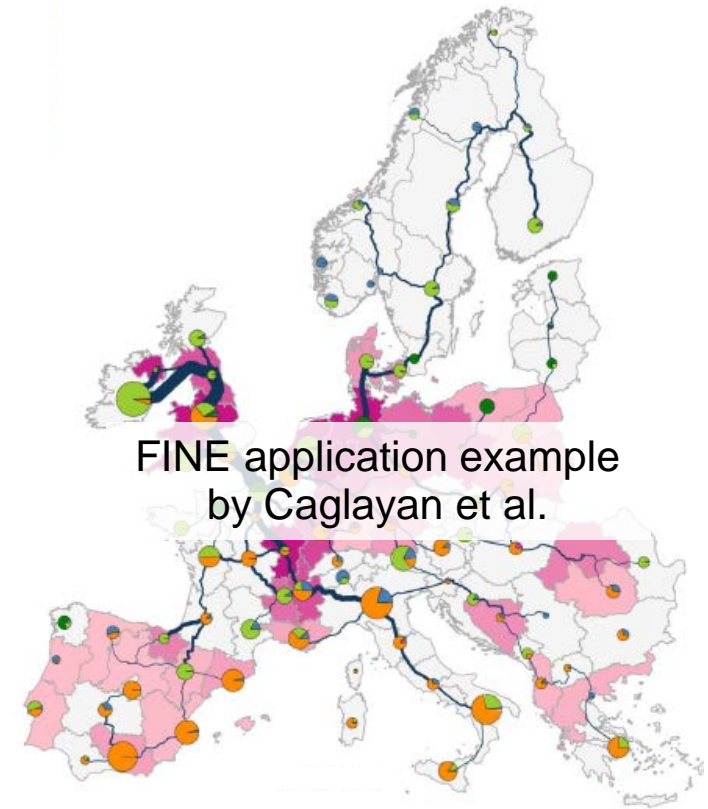
# Summary and Concluding Remarks II

- Running a FINE model via an Excel and a batch script was shortly presented:
  - Here, an exposure to “code” is not given
  - Not all functionalities for result plotting are available
  - In the future, this functionality will be further developed into an Excel GUI
- A large number of studies applying FINE already exist in literature:
  - These cover modeling energy (supply) systems from a district level to an international level
  - An impression of a study was provided to present the application of FINE in a complex scenario setting

## Outlook:

A second part of the tutorial is available which dives deeper into:

- Available modeling options in FINE + which modeling options are smart to use for what application level (district, national, international)
- The object-oriented code of FINE
- The versioning of FINE (FINE is frequently updated, “pip install” installs its latest version)
- Ways to contribute to FINE’s open-source code on GitHub



Let us know how this tutorial can be improved!



# Acknowledgement to the Energy Systems Analysis Team

## Thank you for your attention

