

## Table of Contents:

- Table of Contents:
- I. Prep Work:
  - A. In the *Timeseries Sheet*
    - \* IMV Gamma Method:
    - \* Borselle V Method
    - \* Parkwind Method:
  - B. In the *Parametric Analysis* sheet:
    - \* Parameters:
    - \* Output 1 (All other methods):
    - \* Output 2 (Borselle V Method):
- II. Launching the app:
  - Via Python
  - Using the provided executables (RECOMMENDED)
  - Compiling it yourself
    - \* On Windows (RECOMMENDED)
    - \* On MacOS/Linux (RECOMMENDED)
    - \* On any platform
      - Using Python
      - Using Make
- III. Explaining the source code:
  - A. File Structure (folder by folder):
    - \* Src
    - \* Frontend
    - \* Libs
    - \* Methods
    - \* Modify
  - B. The Business Case Logic:
- IV. Further development (ex: adding new Business Cases):
  - 1. Preparing the environment
  - 2. Write your new code
    - \* New Business Case
    - \* New plot function
    - \* Additional entries
  - 3. Add references to the new stuff in the old code
    - \* New Business Case
    - \* New plot function
  - 4. Recompile

## I. Prep Work:

Make sure that your excel sheets are correctly labeled (the code will crash if there are spelling mistakes)

Make sure that your Timeseries sheet and Parametric Analysis sheets have the

same columns as the **IMV Gamma Case**, *or Borssele V or Parkwind Bus Case*. If not, modify the code accordingly.

Below you will find a list of all columns that are needed/can be put (I will anotate with **OPT** the one that you should only put if you are doing a mixed solar/wind case). The column order is crucial for everything after duration (including duration itself). The method employed for the Borssele V BC has unique columns and the code is written to accomodate for that.

### **A. In the *Timeseries Sheet***

#### **IMV Gamma Method:**

- Date
- Time
- Day-Ahead Prices [Euro/MWh]
- Imbalance Prices [Euro/MWh]
- Wind Speed [m/s]
- Capacity Constraint [MW]

#### **Borselle V Method**

- Date
- Time
- Wind Speed [m/s]
- Potential Generation [MW]
- Actual Generation [MW]
- Delta [MW]
- (Type A) PPA [MW]
- (Type B) Maintenance [MW]
- (Type B) Environmental [MW]
- (Type A) Utility [MW]
- (Type B) Owner Stops [MW]
- Capacity Constraint [MW]
- Generation Constraint [MW]
- Day-Ahead Prices [Euro/MWh]
- Imbalance Prices [Euro/MWh]

#### **Parkwind Method:**

- Starting Time
- Day-Ahead Prices [Euro/MWh]
- Imbalance Prices [Euro/MWh]
- Belwind (181MW)
- OOE Production (15MWp) [MW]
- Capacity Constraint [MW]

## B. In the *Parametric Analysis* sheet:

### Parameters:

- Scenario
- PPA Price
- Market Type
- Wind Power (MW)
- Solar Installed (MWp) (**OPT**)
- Balancing Market Participation
- Storage Power Rating
- Duration

### Output 1 (All other methods):

- Annual Energy Potential
- Annual Energy Generated
- Annual Energy Exported
- Annual Energy Curtailed
- Lost to Storage Inefficiency
- Still In Storage
- Storage CAPEX
- Storage OPEX
- Baseline Revenue
- Revenue Direct Production to Balancing Market [A]
- Revenue Stored Energy to Balancing Market [B]
- Revenue Extra Generation-Based Income [C]
- Revenue with Storage Baseline = [A] + [B] + [C]
- IRR
- NPV

### Output 2 (Borselle V Method):

- Annual Yield Potential [A]
- Type B Curtailment [B]
- Annual Generation Potential [C] = [A] - [B]
- Type A Curtailment [D]
- Annual Energy Generated [E] = [C] - [D]
- Lost to Storage Inefficiency [F]
- Still In Storage [G]
- Annual Energy Exported [H] = [E] - [F] - [G]
- Storage CAPEX
- Storage OPEX
- Baseline Revenue
- Revenue Direct Production to Balancing Market [A]
- Revenue Stored Energy to Balancing Market [B]
- Revenue Extra Generation-Based Income [C]

- Revenue with Storage Baseline =  $[A] + [B] + [C]$
- IRR
- NPV

## II. Launching the app:

### Via Python

Check the pyproject.toml file for information on the packages needed to run the code. The code's entry point is src/main.py so you simply need to run that file to get started (however you need to make sure you have installed all the relevant packages).

You can also run manually:

On windows (run these commands one by one):

*REM I am using py here but use whatever the name of you python command is for the first line*  
`py -m venv venv`  
`.\venv\Scripts\activate`  
`python -m pip install ".[all,build]"`  
`python src/main.py`

On MacOS/Linux (run these commands one by one):

`python3.x -m venv venv` *#replace x with your version*  
`source venv/bin/activate`  
`python -m pip install ".[all,build]"`  
`python src/main.py`

### Using the provided executables (RECOMMENDED)

You can simply use the app by going to the **executables** folder and unzipping the relevant folder: win.zip if you are on Windows, and unix.zip if you are on Mac. In the newly unzipped folder you will find the packaged application ready to go!

### Compiling it yourself

This process is a bit more tedious as you may run into unforeseen errors. However I did my best to manage as many of the errors as possible

#### On Windows (RECOMMENDED)

Simply launch the prep.bat file, either by double clicking it or by running it in the terminal. The compiled app will appear in the dist folder.

### On MacOS/Linux (RECOMMENDED)

Simply launch the prep.sh file, either by double clicking it or by running it in the terminal The compiled app wil appear in the dist folder.

### On any platform

**Using Python** On windows (run these commands one by one):

```
py -m venv venv REM or use whatever the name of your python executable is
.\venv\Scripts\activate
python -m pip install tomllib tomli REM (whichever works)
python -m pip install pyinstaller
python build.py
```

On MacOS/Linux (run these commands one by one):

```
python3.x -m venv venv #replace x with your version
source venv/bin/activate
python -m pip install tomllib tomli # whichever works
python -m pip install pyinstaller
python build.py
```

The compiled app wil appear in the dist folder.

**Using Make** Run in the following order:

```
make venv
make install
make build
```

The compiled app wil appear in the dist folder.

## III. Explaining the source code:

### A. File Structure (folder by folder):

#### Src

This is the main folder where the project's code is stored. It is further subdivided into subfolders for more clarity.

---

File Name	Purpose
main.py	This file is the entrypoint to the app. Just run it to run the app directly
tests.py	Allows accessing functionality defined for the BCA without having to go through the GUI.

---

## Frontend

This folder contains all the python code linked to the frontend and User Interface

File Name	Purpose
frontend.py	The core functionality related to the user interface (There should be no need to modify this).
popup.py	A file which contains the code for the progress bar which appears as a popup.

## Libs

This folder collects all the miscellaneous functions used throughout the program and useful functionality.

File Name	Purpose
excel.py	Contains all functionality which directly interacts and modifies the excel file.
extra.py	A collection of useful functions (Feel free to add functions here if needed).
logger.py	Sets up the logger which is used throughout the program to maintain a clear backtrace

## Methods

File Name	Purpose
bv_method.py	This file defined the Borssele V method for calculating the business case
inv_method.py	This file defines the IMV Gamma method for calculating the business case
parkwind_method.py	This file defines the Parkwind method for calculating the BC
general_methods	This file defines the Generalized method for calculating the BC

If you write any new methods add them here !

## Modify

File Name	Purpose
settings.py	This file contains all the freely modifiable settings.
plots.py	The functions defining the different plots offered to the user.
bca_entrypoint.py	This file contains the entrypoint into the BC logic.
bca_class.py	Contains the class which contains all the variables that are used throughout the BC.

## B. The Business Case Logic:

See the relevant file

## IV. Further development (ex: adding new Business Cases):

### 1. Preparing the environment

You need to make sure that you have all the required packages installed. You can manually check `pyproject.toml` or execute the `prep.bat` or `prep.sh` file or follow this handy guide: First you need to create a virtual environment for python to download the packages: First you need to open a terminal window in your project folder (ex: in the `BVA_v1.1` folder), which is normally an option when you right click a folder Replace "py" here with whatever the command for your python is (ex: "py", "python", "python3", "python3.10", "python3.11", "python3.12", "python3.13", "python3.14")

Warning

You need a version of python  $\geq 3.10$

```
py -m venv venv
```

Now you want to activate this virtual environment: On Windows:

```
.\venv\Scripts\activate.bat
```

On Mac/Linux:

```
source venv/bin/activate
```

Now that your environment is ready you can simply do (this time you HAVE TO write "python" specifically)

```
python -m pip install ".[all,build]"
```

This will install the packages.

### 2. Write your new code

Now that your environment is ready, you can freely write code for a new BC method, or a new plot technique.

#### New Business Case

Please write all your new code to its own file named after the BC, and then put the file in the `methods` subfolder!

A `Business_Case` class was created to contain all the important variables needed throughout a BC method. Currently there is:

1. *Variables defined before the BC truly begins:* These variables save all the input from the user interface and make them easily accessible for the BC
  - **df:** The Timeseries sheet as a pandas DataFrame

- **param\_df**: The Parametric Analysis sheet as a pandas DataFrame
- **input\_values**: The values inputted in the frontend by the user (saved a dictionary)
- **method**: The method being used (annotated with a whole number  $\geq 0$ )
- **case\_type**: The current case type (same way of annotating)
- **plotting**: A Boolean which when False makes it so that the computed python plots won't show up automatically on screen (but they will still be saved to Downloads/BCA\_Plots)
- **scenario\_list**: A list of all the scenarios the user wants the computer to calculate

2. *Variables that get defined during the BC:*

- **years\_covered**: Amount of years covered by the BC
3. *Variables defined on a per-scenario basis that needed to be saved* The following variables are defined differently for every scenario, but needed to be saved to this class so that they can be accessed by the plots. If you have any other variables which are calculated during the BC logic that you want to keep around and be used by the plots save them here.
- **power\_level**: The power level ?

Please note that ALL business cases should only take as input the Business Case class (defined above) and the scenario index. And should output nothing. So if you need extra inputs from the user interface to be carried over, or want outputs from your functions saved, please add them to this class.

So your new file MUST look like this:

```
def name_of_your_new_bc_method(business_case, scenario_index, debug_mode):
    # Define all your variables
    var1 = ...
    var2 = ...
    ...

    result = run_bus_case(
        business_case,
        var1,
        var2,
        ...
    )

    business_case.param_df.iloc[scenario_index, x:y] = result
    # Here x is the position of the first column of the parametric Analysis sheet you have
    return

def run_bus_case(business_case, var1, var2, ...)
    # Your code:
```



```

...

result = [
    # ...
]
return result

```

Of course you can add more functions and code than just these two, but please be aware that the only part of your code that will be accessed by the rest of the app is the `name_of_your_bc_method` function

### New plot function

Simply write your new function to the plots file and follow some key rules that the first two plots are using:

- the only inputs should be the `business_case` class and the `scenario_index` and `debug_mode` (even if you don't use it)
- and you must add the following line near the beginning:

```

if business_case.plotting:
    # Create a new popup Tkinter window
    popup = tk.Toplevel()
    popup.title(f"Scenario {scenario_name}: Plot Name")
    popup.geometry("1400x900")

```

This code ensures that the plot will not create a popup if the user chooses to not have the plots popup

- And then where you would normally `plot.show()` write instead:

```

if business_case.plotting:
    # Embed the figure in the Tkinter window
    canvas = FigureCanvasTkAgg(fig, master=popup)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    log_print(f"PLOT NAMED plot displayed in popup for scenario {scenario_name}.")
    save_figure(fig, "plotname", scenario_name)
    log_print(f"PLOT NAMES plot saved for scenario {scenario_name}.")

```

This code fixes issues that occur when running matplotlib through a graphical user interface, and ensure the plot will be saved.

Alternatively, if you don't want to manually ensure compatibility between your new plot function and the old ones, you can simply give ChatGPT the two old ones, and your new one and tell him to ensure the new one follows the same format as the old ones by following the rules given above.

### Additional entries

Add your new entry to the INPUT\_FIELDS variable following the same format as the other ones. If you expect your input to be text based then also add it to the STRING\_BASED variable. You can then access this input value (and all input values) through the business\_case.input\_values variable. Ex: if you add a new value called "Value1" then business\_case.input\_values["Value1"] will allow you to access it in your business case.

## 3. Add references to the new stuff in the old code

To allow your new code to be accessed by the program, you must add references to your newly written functions where relevant.

### New Business Case

To have your newly written BC be used you must:

1. Go to the settings file and add your method to the CHOICE\_MATRIX, following the same format as the prior ones. If your BC is for a new case type you can also add an entry there for the case\_type. Then add a reference to the function of your method to the METHOD\_SET variable in the right position (You can determine the number of your method which corresponds to its position in the METHOD\_SET by simply counting left to right, top to bottom, all the methods in the CHOICE\_MATRIX (make sure you count the first entry as the 0th))

And done! Normally your new BC should be perfectly accessible.

2. Remarks: if your new BC requires new inputs, or key variable to be carried over between functions or to the plots, make sure to add your inputs to the Business\_Case class defined in the bca\_class and then have the variable be assigned a value in the setup() function of the class or at the beginning of your BC code.

### New plot function

To have your new plot function be accessed, go to the settings file and in AVAILABLE\_PLOTS variable add:

```
"Plot-name": plot_function_name,
```

And thats it! if you have correctly ensured compatibility with the old plots functions then it should work out of the box.

## 4. Recompile

Once you have made your changes, and have confirmed that the code is running without any issues (which you can do by running main.py), you can go ahead and

recompile it into a permanent and portable executable which doesn't require having access to python and all the required libraries to run. To recompile the app either:

- Using python run the build.py file
- On windows run the prep.bat file
- On MacOS/Linux run the prep.sh file
- If you have make installed you can run `make build`

Once the app has been compiled it will appear in the dist folder.