Technische Universität Berlin
Fakultät VII Wirtschaft & Management
Fachgebiet Wirtschafts- und Infrastrukturpolitik (WIP)

Energy Economics - Energy Sector Modelling

An Investment Model for Smart Stand-Alone Micro Grids for Agricultural Farms

Author:
Francisco Jose Manjon Cabeza Garcia (369293) - manjoncabezagarcia@campus.tu-berlin.de
Alessandro James Carenini (475867) - alessandro.james.carenini@campus.tu-berlin.de
Francesca Cioccarelli (475816) - francesca.cioccarelli@campus.tu-berlin.de
Gaspard Robert (475861) - gaspard.robert@campus.tu-berlin.de

Supervisors:
Prof. Dr. Christian von Hirschhausen
Mario Kendziorski

Berlin, Thursday 2nd February, 2023

Statutory declaration

Hereby, we declare that we have developed and written this research completely by ourselves and that we have not used sources or means without declaration in the text. Any external thought, content, media, or literal quotation is explicitly marked and attributed to its respective owner or author.
As of the date of submission, this piece of document and its content have not been submitted anywhere else but to our supervisors.

Berlin, Thursday 2nd February, 2023

Francisco Jose Manjon Cabeza Garcia	Alessandro James Carenini
Francesca Cioccarelli	Gaspard Robert

Abstract

We present a model to optimise the investments necessary to reduce the cost of electricity generation and consumption in the context of agricultural farms. The model can handle electricity consumption from water pumps and water tanks for the irrigation of a field, as well as consumption from other sources, e.g., lighting or cooking. The model optimises the irrigation and water storage processes to reduce the total cost of electricity. The irrigation efficiency throughout time is taken into consideration, so as to reduce the cost of pumping water to the field considering evaporation depending on the time of the day. Moreover, the model returns the optimal investments in dispatchable, non-dispatchable and electricity storage facilities which minimise the total cost of electricity production. It can also take into account the electricity production facilities already installed. An application example of the model is provided for a farm in Algeria.

Contents

List of Figures

List of Tables

List of Acronyms

DZD	Algerian Dinar
GoA	Government of Algeria
Li-ion	Lithium-ion
USD	United States Dollar

1. Introduction

Algeria has a great fossil energy potential, and nowadays it is developing at most its depletion by exploiting its potential and, as a consequence, facing an important imbalance between supply and demand. Different programs have been launched by the Government of Algeria (GoA) in order to exploit the energy transition towards sustainable development at its best.

According to the US Department of Agriculture 2020, the agricultural sector is a priority for the GoA in its efforts to diversify the economy. The government supports innovation in the agricultural sector to achieve self-sufficiency and promote exports. In 2020, despite the pandemic, the Algerian agricultural output exceeded 25 USD billion, up from 23 USD billion in 2019. Furthermore, according to the Global Agriculture Information Network 2021, the overall sector contributes up to 12.4% to the Algerian GDP, and accounts for more than 2.5 million direct jobs.

Seeing the relevance of the agricultural sector in the country, the GoA has focused on farmers in order to start the shift to renewable energy sources. There is indeed a goal to reach the correct balance between crop productivity and economic stability by minimising the use of fossil fuels.

Our main objective is to adapt a system to the load profile and storage strategies of modern and sustainable agricultural practices to serve the typical requirements needed to run farming activities. Furthermore, the process of generating, distributing, and storing electricity and water will be optimised, for example by optimising the electricity costs through investments in PV and batteries.

This paper also studies a specific case of sustainable transition in agriculture: the Belaidouni Mohamed Farm placed in Algeria. Specifically, we answer the question of how the electricity costs can be optimised by investing into PV and batteries.

Figure 1 shows the electricity and water grid that the specific case study requires to optimise. On the left, we see a diesel generator[1] at the bottom of the schema, which is already installed. The dashed rectangle above shows the PV[2] and electricity storage facilities[3] which are not yet installed. Our objective is to find out which PV installation size and storage facility capacity should be installed to optimise the total cost of electricity used by the farm. In the middle

1. https://www.iconfinder.com/icons/172463/engine_icon
2. https://www.iconfinder.com/icons/9045094/solar_panel_icon
3. https://www.iconfinder.com/icons/8541628/database_data_storage_icon

section, we see a water pump[4] at the top, which can pump water directly into the field[5] or to the water tank[6] depicted below it. The water tank is assumed to work with gravity, and does not need electricity to pump the water it stores into the field. We also consider the water usage, and optimise it to minimise the total cost of the investment in PV and storage, as well as the cost of using the diesel generator if needed. The black arcs show the flow of electricity, while the blue arcs show the flow of water.

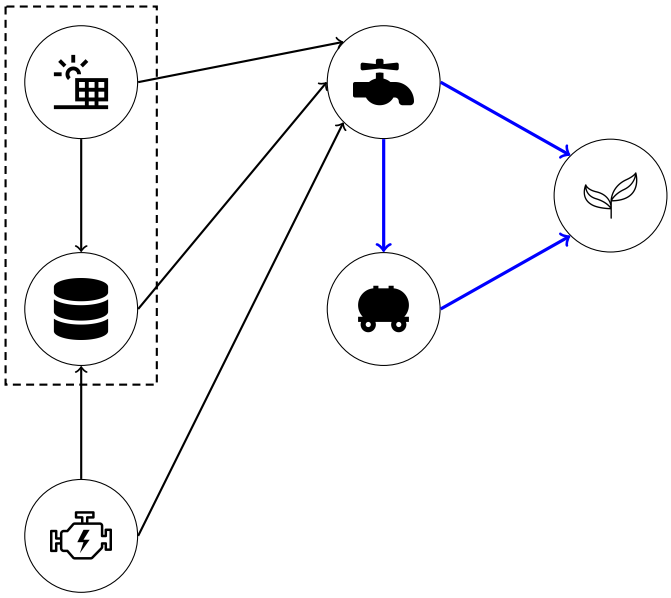


Figure 1: Target Electricity (black) and Water Grid (blue).

In order to structure the paper, different chapters have been created. Apart from this introductory section, section 2 gives a brief literature review to sum up all the research done and which can support our analyses. Next, an explanation of both the methodology used for running the study and the data used to get results for our particular case study are given in sections 3 and 4 respectively. Moreover, the results and a brief discussion to better understand the outcome of the model are presented in section 5. Finally, we conclude this paper with a summary of our findings in section 6.

4. https://www.iconfinder.com/icons/8665381/faucet_tap_water_icon
5. https://www.iconfinder.com/icons/7099593/agriculture_plant_sprout_environment_tree_icon
6. https://www.iconfinder.com/icons/4752989/fuel_gas_oil_tank_tanker_truck_water_icon

2. Literature Review

In this section, we provide a short overview of the published research that we have used to base our model development and parameter values' selection.

The most important paper to conduct our study was Bekrit and Mesli 2022. This master thesis was a relevant source of information for our paper since it focuses on the same farm of our case study.

Solar pumping is a technique to pump water from a water source using solar energy. Bekrit and Mesli 2022 go deeply into the sizing of a solar pumping system for viticulture irrigation, replacing a diesel-based system. However, this is not the only way to use solar energy in agriculture. Some examples are solar tractors, solar dryers, solar greenhouses or the simple use of solar energy for the domestic part of the farm (heating, lighting, etc.). The solar pumping itself has different parameters which are important for the sizing:

- The usage (or not) of batteries to pump water during the night or days without sun;
- The type of pump: displacement vs. centrifugal pump;
- Location of the pump: underwater vs. surface pump;
- Required flow, which depends on the temperatures, precipitations or irrigation methods;
- Total dynamic head, which is the total amount of pressure when water is flowing in the system. It is comprised of two parts: the vertical rise and friction loss.

The study in Bekrit and Mesli 2022 was conducted using the `PVsyst` software, which takes into account several parameters, and outputs different information e.g., electricity production, costs, required space, return on investment etc.. It is clearly a different approach from ours, because our goal is to develop an optimisation model and implement it in Julia ourselves. In Bekrit and Mesli 2022, the main goal is to pump the water from a well to the highest point of the farm. Three scenarios were developed in which a different number of intermediate tanks were considered. This is a more complex approach than the one we present in this paper. The main conclusions in the best-case scenario from Bekrit and Mesli 2022 are:

- Investment cost for the whole solar panels system: 9185102.4 DZD = 65259.58 €[7].
- Pumped water: 79257 m^3/year.
- Annual costs for the solar panels: 80550 DZD = 572.30 €.

7. Exchange rates as of 1st September 2022

- The amortisation period of the new investment is computed using a simple method, which is based on a 25 years lifetime for the new investment, the annual costs over 25 years and the annual expenses for the Diesel-based pumping: $N = \frac{\text{Initial investment + Annual costs over 25 years}}{\text{Annual diesel expenses}}$. The result is an amortisation period of 8.4 years in their study.

This last result is subject to high variance when other literature and case studies are considered. For instance, in Giri et al. 2020, it is 2-6 years for a farm in India. The result appears to be very dependent on the farm and the geography of its location.

3. Methodology

In this section, we present an investment model which optimises the total cost of electricity production and new investments, subject to minimum electricity production and watering requirements to operate a farm. The model does not only optimise the total cost by deciding what investments to undergo and how electricity should be produced and used, but also optimises the irrigation process by deciding when and how much to irrigate.

First, we generalise the grid from figure 1 to depict the grid that our model will optimise. The goal is to define an optimisation model that can be used for a general farm. Later in section 4, we will specify the parameters used for our case study, and apply the model to the simpler grid from figure 1.

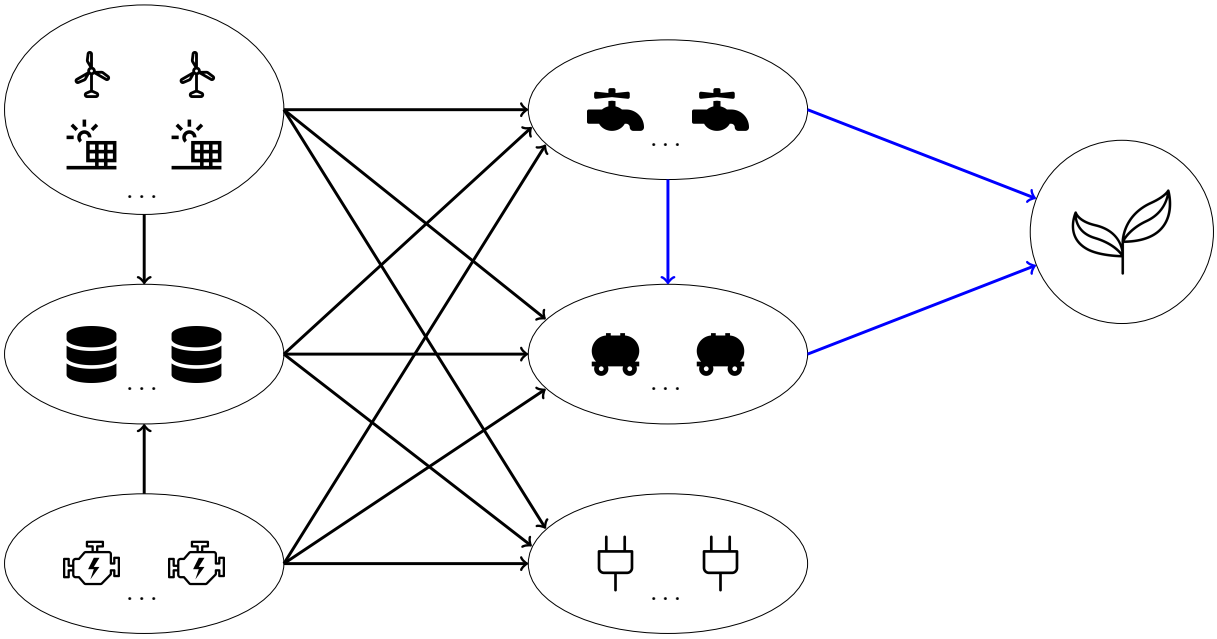


Figure 2: General Target Electricity (black) and Water Grid (blue).

Figure 2 shows a generalised electricity and water grid with their arcs[8] and nodes. The blue arcs show the flow of water, and the black arcs the flow of electricity. On the left side, we have a set of non-dispatchable generators[9] at the top, a set of electricity storage facilities in the middle, and a set of dispatchable generators at the bottom. The electricity generated by these 3 sets of facilities is fed into three sets of electricity consuming nodes, which are depicted in the centre of the graph. In the centre at the top, we have a set of water pumps, in the middle a set of water tanks, and at the bottom a set of other demand nodes[10]. Finally, at the right side of the graph,

8. The arcs in the graph are hyperarcs, i.e., each arc connecting two sets corresponds to a set of arcs connecting each node in the source set to each node in the target set.
9. https://www.iconfinder.com/icons/3737621/energy_plant_power_station_turbine_wind_icon
10. https://www.iconfinder.com/icons/7076597/connector_charge_electricity_plug_power_plug_plugin_power_icon

we have the water grid, with a single field which needs to be irrigated.

As stated above, we aim to model this grid to optimise the total cost of electricity plus investments in dispatchable, non-dispatchable and storage facilities. We do this by optimising the irrigation process considering the water needs of the field, and the evaporation of water depending on the time of the day and the day at which irrigation occurs. By adjusting the model parameters, we can specify the number of facilities to install, and the facilities that are already there. For the mathematical model, we try to define all variables and parameters in matrix form[11].

3.1. Model Parameters

Before we can define the objective function and the constraints of the optimisation model, we need to define all parameters in the model that are assumed to be given, and which are not subject to optimisation in contrast to the variables defined below.

First, we define the parameters concerning the electricity supply side.

- Let $T \in \mathbb{N}$ be the number of time periods over which we want to optimise.
- Let $N_{DISP} \in \mathbb{N}$ be the number of dispatchable generators, N_{NDISP} be the number of non-dispatchable generators and be the number of storage facilities[12]. We define the total number of electricity production facilities $P := N_{DISP} + S + N_{NDISP}$.

  The P dimensional vectors defined below are all assumed to be sorted, i.e., the first N_{DISP} elements correspond to the values of the dispatchable generators, the next S elements to the values of the storage facilities, and the final N_{NDISP} elements to the values of the non-dispatchable generators.
- $\mathit{eff} \in [0,1]^P$ is the vector reporting the efficiency factor of each generator. We set the efficiency factor of non-dispatchable facilities equal to 1, i.e., $\mathit{eff}_i := 1$ for all $i = N_{DISP} + S + 1, \ldots, P$.
- $fc \in \mathbb{R}^P_{\geq 0}$ is the vector of fuel costs. We set the fuel cost of non-dispatchable and storage facilities equal to 0, i.e., $fc_i := 0$ for all $i = N_{DISP} + 1, \ldots, P$.
- $P_{CO_2} \geq 0$ is the price of CO_2 emissions.
- $ef \in \mathbb{R}^P_{\geq 0}$ is the vector of emission factors, i.e., the amount of CO_2 emitted per unit of electricity produced. We set the emission factors of non-dispatchable and storage facilities equal to 0, i.e., $ef_i = 0$ for all $i = N_{DISP} + 1, \ldots, P$.

11. Vectors are nothing else than a single-column or single-row matrix.
12. Even though storage facilities can also be considered dispatchable facilities, we consider them separately.

- $vc \in \mathbb{R}^P_{\geq 0}$ is the vector of variable costs, i.e., the production cost per unit of electricity. We set the variable cost of each unit of electricity produced by non-dispatchable equal to 0, i.e., $vc_i = 0$ for all $i = N_{DISP} + S + 1, \ldots, P$.
- $mc \in \mathbb{R}^P_{\geq 0}$ is the vector of marginal cost of each unit of electricity produced by each generator. We set the marginal cost of each unit of electricity produced by non-dispatchable facilities equal to 0.

$$mc_i := \begin{cases} \frac{fc_i}{eff_i} + P_{CO_2} \frac{ef_i}{eff_i} + vc_i & \text{for} \quad i = 1, \ldots, N_{DISP} + S, \\ 0 & \text{for} \quad i = N_{DISP} + S + 1, \ldots, P. \end{cases}$$

- $LL_{cost} \geq 0$ is the lost load cost. As we will discuss below, we allow for lost load to improve the probability that our optimisation problem is feasible. The lost load cost parameter allows us to penalise that the model incurs in lost load, while still allowing it.
- $IC \in \mathbb{R}^P_{\geq 0}$ is the vector of investment costs of each electricity generation facility per unit of maximum capacity.

$$IC_k := I_k \frac{(1+i)^{lt_k}}{(1+i)^{lt_k} - 1} + fc_i,$$

  where $lt_k \in \mathbb{N}$ is the lifetime in years, $I_k \in \mathbb{R}_{\geq 0}$ the total investment cost over its lifetime and $fc_k \in \mathbb{R}_{\geq 0}$ the yearly fixed costs of the facility $k = 1, \ldots, P$. If a facility is already installed, we can either set its investment cost to 0 plus yearly fixed costs or to the yearly opportunity costs plus yearly fixed costs of the facility. All these figures are defined per unit of power capacity.
- $avail \in [0,1]^{T \times N_{NDISP}}$ is the matrix with the availability of non-dispatchable electricity sources at each point in time.
- $SW, MSW \in \mathbb{R}^P_{\geq 0}$ are the surface area that each facility needs per unit of maximum production capacity, and the maximum surface area that each facility can take. If a facility $i \in \{1, \ldots, P\}$ does not have a surface area constraint, then we set $SW_i = MSW_i = 0$, so that the constraint in equation 9 that we present later is always fulfilled.

Second, we define the parameters concerning the electricity demand side.

- Let $W_{Pumps}, W_{Tanks}, E \in \mathbb{N}$ be the number of water pumps, water tanks, and other electricity consuming facilities respectively. The demand side $D := W_{Pumps} + W_{Tanks} + E$ dimensional vectors are also defined as sorted vectors, such that their first W_{Pumps} elements correspond to water pumps, the next W_{Tanks} correspond to water tanks, and the last E elements correspond to the other electricity consuming facilities.

- $DEW \in \mathbb{R}_{\geq 0}$ is the daily effective amount of water in units of volume that the field needs. Here, effective means the amount of water that actually needs to be absorbed by the plants, i.e., after evaporation.
- $WE \in [0,1]^T$ is a vector with the irrigation efficiency at each point in time. This vectors helps the model to optimise the point in time at which the filed should be irrigated, to minimise the cost of electricity needed for that, but accounting for the water lost to evaporation at each point in time. Watering plants during the day might be more energy efficient, because the electricity used can be generated by a PV installation. However, during the day more water will evaporate. Hence, this trade-off needs to be optimised.
- $WEC \in \mathbb{R}_{\geq 0}^{W_{Pumps}+W_{Tanks}}$ is the cost in energy units to pump 1 unit of water in volume by each water pump and water tank.
- $OD \in \mathbb{R}_{\geq 0}^{T \times E}$ is the electricity demand of each of the other E electricity consuming facilities at each time $t = 1, \ldots, T$.
- $WTC \in \mathbb{R}_{\geq 0}^{W_{Tanks}}$ is the maximum volume capacity of each water tank.

3.2. Model Variables

We define the variables in the model as follows:

- Let $G = [g_{t,i}] \in \mathbb{R}^{T \times P}$, where $g_{i,j}$ is the electricity produced by facility $i = 1, \ldots, P$ at time $t = 1, \ldots, T$.
- Let $CH = [ch_{t,i}] \in \mathbb{R}^{T \times S}$, where $ch_{i,t}$ is the amount of electricity used to charge the storage facility $i = 1, \ldots, S$ at time $t = 1, \ldots, T$.
- Let $L = [l_{t,i}] \in \mathbb{R}^{T \times S}$, where $l_{i,t}$ is the storage level of the storage facility $i = 1, \ldots, S$ at time $t = 1, \ldots, T$.
- Let $PW \in \mathbb{R}^P$ be the vector which contains the maximum capacity of each production facility, or the maximum storage capacity in the case of storage facilities.
- Let $WP \in \mathbb{R}^{T \times (W_{Pumps}+W_{Tanks})}$ be the matrix with the amount of water pumped by each water pump and tank at each point in time.
- Let $WCH = [wch_{t,i}] \in \mathbb{R}^{T \times W_{Tanks}}$, where $wch_{i,t}$ is the amount of water used to fill the water tank $i = 1, \ldots, W_{Tanks}$ at time $t = 1, \ldots, T$.
- Let $WL = [wl_{t,i}] \in \mathbb{R}^{T \times W_{Tanks}}$, where $wl_{i,t}$ is the water level of the water tank $i = 1, \ldots, W_{Tanks}$ at time $t = 1, \ldots, T$.

We also define the following variables to improve the probability that a feasible solution exists:

- Let $CU \in \mathbb{R}^T$ be the vector which represents the curtailment at each point in time $t = 1, \ldots, T$.
- Let $LL \in \mathbb{R}^T$ be the vector which represents the lost load at each point in time $t = 1, \ldots, T$.

3.3. Model Objective Function and Constraints

Now that we know what parameters are given and the variables that we want to optimise, we can define the objective function that we want to minimise. This function adds up the cost of electricity production, investments in electricity generating facilities and investments in storage facilities over the whole period of time from $t = 1$ to $t = T$. The investments in generators and storage facilities are annualised. The factor $\frac{8760}{T}$ computes the length of the optimisation period in years to adjust the computed total cost of electricity to yearly total cost.

$$f(G, LL, PW) := \underbrace{\frac{8760}{T} \cdot \mathbb{1}_T^T \cdot (G \cdot mc + LL_{cost} \cdot LL)}_{\text{Yearly total cost of electricity production}} + \underbrace{(IC \cdot PW)}_{\text{Yearly investment costs}} . \tag{1}$$

The first set of constraints that the model has to fulfil are the energy balance constraints. That means, that the electricity production from dispatchable and non-dispatchable generators as well as from discharging the storage facilities together with the lost load must be equal to the energy demand to pump water plus the electricity demand from other electricity consuming facilities plus the electricity used for charging storage facilities plus curtailment at every point in time.

$$G \cdot \mathbb{1}_P + LL = WP \cdot WEC + OD \cdot \mathbb{1}_E + CH \cdot \mathbb{1}_S + CU. \tag{2}$$

We also need to constraint the generation from non-dispatchable sources to their availability.

$$g_{t,i} = PW_i avail_{t,i} \quad \text{for all} \quad i = N_{DISP} + S + 1, \ldots, P \quad \text{and} \quad t = 1, \ldots, T. \tag{3}$$

The second set of constraints is given by the maximum generation limits of the dispatchable generators: at every point in time the electricity generation form each dispatchable generator must be less than or equal to the maximum capacity of that dispatchable generator.

$$g_{t,i} \leq PW_i \quad \text{for all} \quad i = 1, \ldots, N_{DISP} \quad \text{and} \quad t = 1, \ldots, T. \tag{4}$$

Third, we have the maximum storage constraints, which limit the charge of each storage facility at every point in time to the maximum possible charge level of that facility.

$$CH_{t,i} \leq PW_{i+N_{DISP}} \quad \text{for all} \quad i = 1, \ldots, S \quad \text{and} \quad t = 1, \ldots, T. \tag{5}$$

Fourth, we need to constraint the storage facility levels to their maximum capacity. That means that the storage level of each storage facility can never surpass its maximum capacity.

$$L_{t,i} \leq PW_{i+N_{DISP}} \quad \text{for all} \quad i = 1, \ldots, S \quad \text{and} \quad t = 1, \ldots, T. \tag{6}$$

Fifth, the difference between the storage levels at two consecutive points in time is equal to the difference between the effective charge and the effective discharge[13] at the first of the two points in time. The initial storage level must also be equal to 0 for all electricity storage facilities.

$$L_{t+1,i} = L_{t,i} + \mathit{eff}_{i+N_{DISP}} CH_{t,i} - \frac{g_{t,i+N_{DISP}}}{\mathit{eff}_{i+N_{DISP}}} \quad \text{for all} \quad i = 1, \ldots, S, \quad t = 1, \ldots, T-1. \tag{7}$$

$$L_{1,i} = 0 \quad \text{for all} \quad i = 1, \ldots, S. \tag{8}$$

Sixth, we add constraints for the area covered by the generation facilities.

$$PW_i SW_i \leq MSW_i \quad \text{for all} \quad i = 1, \ldots, P. \tag{9}$$

Seventh, the field must be irrigated with at least DEW effective units of water volume. Here, we assume that the time resolution is hourly, and that $\frac{T}{24} \in \mathbb{N}$. Thus, the amount of water pumped by the water pumps and the tanks minus the water used to fill the water tanks times the irrigation efficiency must be greater than or equal to the daily effective water need of the field each day in the optimisation period.

$$\sum_{t=24n}^{24(n+1)-1} WE_t \cdot \left(WP \cdot \mathbb{1}_{W_{Pumps}+W_{Tanks}} - WCH \cdot \mathbb{1}_{W_{Tanks}} \right)_t \geq DEW \quad \text{for all} \quad n = 1, \ldots, \frac{T}{24}. \tag{10}$$

Eight, as for the electricity storage facilities, the water tanks must fulfil the maximum storage, maximum capacity, level difference and initial level constraints. For the level difference constraint, we assume that the water stored in water tanks evaporates according to the water efficiency factor at the corresponding point in time.

$$WCH_{t,i} \leq WTC_i \quad \text{for all} \quad i = 1, \ldots, W_{Tanks} \quad \text{and} \quad t = 1, \ldots, T. \tag{11}$$

$$WL_{t,i} \leq WTC_i \quad \text{for all} \quad i = 1, \ldots, W_{Tanks} \quad \text{and} \quad t = 1, \ldots, T. \tag{12}$$

13. For a storage facility to contribute $g_{t,i+N_{DISP}}$ units of energy, its storage level must fall $\frac{g_{t,i+N_{DISP}}}{\mathit{eff}_{i+N_{DISP}}}$ units of energy.

$$WL_{t+1,i} = WE_t \cdot WL_{t,i} + WCH_{t,i} - WP_{t,i+W_{Pumps}} \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}, \quad t = 1, \ldots, T, \tag{13}$$

$$WL_{1,i} = 0 \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}. \tag{14}$$

Finally, we require all variables to be non-negative[14]:

$$G, CH, L, PW, WP, WCH, WL, CU, LL \geq 0.$$

The model presented above does not consider the option to buy electricity from the (national) grid or sell it to it explicitly. However, one could implicitly allow for buying, selling or both of them by defining the buying node as a dispatchable generator with marginal price equal to the buying price of electricity from the grid. For selling, another dispatchable generator can be defined, with marginal price equal to the price paid for injecting power into the grid, but with a negative sign. This can help to model a farm that is connected to other farms (micro-grid) or to the national grid. Nevertheless, this approach would be suited to neither account for fluctuations in the maximum capacity that the national grid can deliver to the farm on the buy side nor to account for fluctuations in the priced paid for injecting power to grid on the sell side. Thus, an extension of the model with new variables and constraints might be better suited to account for a connection to the (national) grid. Our model does not consider investments in water tanks either. But adapting the model to also take that possibility into account is not difficult.

It is possible to account for facilities that are already installed. As discussed above, the investment cost of already installed facilities can be set to 0, and the value of the corresponding variable in the vector PW fixed to the current installed capacity of the corresponding installed facility with a new constraint[15]. Alternatively, the investment cost can also be set to the opportunity cost of the facility, while the corresponding entry in the variable vector PW is only constrained to be less than or equal than the current installed capacity. In this last method, the model can find an optimal capacity below the one currently installed, which would mean, that it might make sense to (partially) sell the current installed facility.

3.4. Final Model

Using the objective function and the constraints defined above, we can summarise the optimisation model as follows:

14. Inequality signs are considered element-wise when comparing vectors and matrices.

15. We refrain from adding this feature to the mathematical model above, because it would further increase its complexity and reduce readability. Nevertheless, this feature in implemented in the Julia model that we coded

$$\min \quad \frac{8760}{T} \cdot \mathbb{1}_T^T \cdot (G \cdot mc + LL_{cost} \cdot LL) + (IC \cdot PW)$$

$$\text{s.t.} \quad WP \cdot WEC + OD \cdot \mathbb{1}_E + CH \cdot \mathbb{1}_S + CU = G \cdot \mathbb{1}_P + LL$$

$$g_{t,i} = PW_i avail_{t,i} \quad \text{for all} \quad i = N_{DISP} + S + 1, \ldots, P, \; t = 1, \ldots, T$$

$$g_{t,i} \leq PW_i \quad \text{for all} \quad i = 1, \ldots, N_{DISP}, \; t = 1, \ldots, T$$

$$CH_{t,i} \leq PW_{i+N_{DISP}} \quad \text{for all} \quad i = 1, \ldots, S, \; t = 1, \ldots, T$$

$$L_{t,i} \leq PW_{i+N_{DISP}} \quad \text{for all} \quad i = 1, \ldots, S, \; t = 1, \ldots, T$$

$$L_{t,i} + \mathit{eff}_{i+N_{DISP}} CH_{t,i} - \frac{g_{t,i+N_{DISP}}}{\mathit{eff}_{i+N_{DISP}}} = L_{t+1,i} \quad \text{for all} \quad i = 1, \ldots, S, \; t = 1, \ldots, T-1$$

$$L_{1,i} = 0 \quad \text{for all} \quad i = 1, \ldots, S$$

$$PW_i SW_i \leq MSW_i \quad \text{for all} \quad i = 1, \ldots, P$$

$$\sum_{t=24n}^{24(n+1)-1} WE_t \cdot \left(WP \cdot \mathbb{1}_{W_{Pumps}+W_{Tanks}} - WCH \cdot \mathbb{1}_{W_{Tanks}}\right)_t \geq DEW \quad \text{for all} \quad n = 1, \ldots, \frac{T}{24}$$

$$WCH_{t,i} \leq WTC_i \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}, \; t = 1, \ldots, T$$

$$WL_{t,i} \leq WTC_i \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}, \; t = 1, \ldots, T$$

$$WE_t \cdot WL_{t,i} + WCH_{t,i} - WP_{t,i+W_{Pumps}} = WL_{t+1,i} \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}, \; t = 1, \ldots, T$$

$$WL_{1,i} = 0 \quad \text{for all} \quad i = 1, \ldots, W_{Tanks}$$

$$G, CH, L, PW, WP, WCH, WL, CU, LL \geq 0$$

The code of the model implementation in Julia using the JuMP package[16] is attached in appendix B. The code is structured in three files, and well documented with comments. Alternatively, the Julia files as well as the data files used for the specific case study described in this paper can be found in the GitLab repository at https://git.tu-berlin.de/fcomanjon/energy-modelling.

16. Dunning, Huchette, and Lubin 2017

4. Data and Scenario Assumptions

In this section, we define the specific parameter values and assumptions that we used to apply the general model defined in section 3 to our specific case study.

Our main goal for this paper is to develop a model to optimise irrigation to minimise the total cost of electricity and investments in electricity producing facilities. The estimation of the parameters used in the model is out of the scope of this project. Our goal in this and the following section is to produce an example result that we can analyse. To apply the model in the real world, we do not recommend using any of the parameter values defined here.

4.1. Electricity Supply Side Parameters

We apply the model from section 3 to our target grid depicted in figure 1. As a consequence, we assume that we currently have a diesel generator already installed, as well as a water pump and a water tank. The model will optimise the size of the PV installation and the maximum capacity of the electricity storage facility, but we will only consider a single PV facility and a single storage facility. That implies $N_{DISP} := 1$, $N_{NDISP} := 1$ and $S := 1$. Hence, $P := 3$.Moreover, we consider an optimisation period of one year with hourly resolution, i.e., $T := 8760$ hours.

To compute the marginal cost of electricity production, we assume that the efficiency of the diesel generator is $eff_1 := 0.3$, the cost of $1kWh$ of diesel fuel is $fc_1 := \frac{29.01}{9.96}$ DZD[17] and variable cost $vc_1 := 0$. At the moment of writing this term paper, Algeria does not have a CO_2 emissions certificate market. Hence, neither companies nor individuals have a maximum CO_2 emission limit per year. They do not have to pay for their emissions either, regardless of how much they emit. Thus, we assume that $P_{CO_2} := 0$, and it is not necessary to define the emission factor of the diesel generator. Furthermore, we assume a lost load cost of 14000 DZD.

For the investment cost vector, we will assume no opportunity costs for the diesel generator, i.e. $IC_1 := 0$. For the (potential) PV installation, we assume a lifetime of 25 years, a cost per kW of maximum capacity of 10^5 DZD[18] and a fixed maintenance cost of 960 DZD per kW of maximum capacity, as they were assumed in Bekrit and Mesli 2022. We assume a fixed interest rate of 8%, which is the lending interest rate of the Algeria Bank according to the World Bank[19].

17. That is the price of 1 litre of diesel in Algeria, as of 29_{th} August 2022, times the lower calorific value of diesel according to the Merkblatt zur Ermittlung des Gesamtenergieverbrauchs (Stand 30.11.2020).
18. Source: https://youchoz.com/installation-panneau-solaire-algerie/
19. https://data.worldbank.org/indicator/FR.INR.LEND?locations=DZ

According to International Renewable Energy Agency 2017, the investment costs for a Li-ion battery lie between 200 USD/kWh and 840 USD/kWh. We take the average, i.e., 73143.20 DZD/kWh[20], assume a lifetime of 10 years, and no maintenance costs.

To compute PV and wind. Our naive approach is to set the availability of PV to the solar energy vector from Visual Crossing Corporation 2021 divided by its maximum, i.e., normalised. The same approach is implemented for wind with the wind speed data from Visual Crossing Corporation 2021, although not used in our case study.

In our case study the total surface available for installing PV modules is 2.7 ha. Because we only consider installing a single PV facility, we set $MSW_3 := 27000$ m^2. Moreover, we assume that PV panels can be installed which have a maximum capacity of 220 kW/m^2, i.e., $SW_3 := \frac{1}{220}$. For the diesel generator and the electricity storage facility, we assume no surface constraints.

4.2. Electricity Demand Side Parameters

On the demand side, we assume that only a single water pump and a water tank are part of the grid. That means $W_{Pumps} := 1$, $W_{Tanks} := 1$, $E := 0$, i.e., $D := 2$. Thus, the matrix of electricity demand from other sources is considered to be the zero-matrix. We also assume a daily effective water need of 43.5 m^3/ha[21], which for a 174.5 ha farm means $DEW := 7590.75$ m^3.

For the energy cost to pump 1 unit of water volume, we assume that the water pump needs 619 Wh/m^3 and that the water tank works with gravity, i.e., it needs no electricity to irrigate as in Bekrit and Mesli 2022. Thus, $WEC_1 := 0.619$ kWh/m^3 and $WEC_2 := 0$. Finally, we assume that the water tank has a maximum capacity of 250 m^3.

Finally, we define the irrigation efficiency vector for our case study using the data from Visual Crossing Corporation 2021 and the methodology[22], as well as the assumptions to apply the Penman-Monteith model, described in Mittapalli 2015 and López-Urrea et al. 2006.

$$WE_t := 1 - \frac{ET_t}{\max_{t=1,\dots,T} ET_t}, \quad ET_t := ET_o^t - prep_t, \qquad \text{for all} \qquad t = 1, \dots, T,$$

where ET_o^t is the evapotranspiration and $prep_t$ the precipitation in mm at time t.

20. Exchange rates as of 1$_{st}$ September 2022.
21. This is the average amount of water irrigated using the drip irrigation method as in Bekrit and Mesli 2022. Although this number already accounts for evaporation losses, we ignore that fact.
22. See appendix A.

5. Results and Discussion

In this section, we present and discuss the results[23] of the optimisation model defined in section 3 after running it with the parameter values defined in section 4.

With an objective value of 8.01×10^7 DZD, the model reaches its optimal solution with the parameters described in section 4. This objective value can be split in two: the yearly cost of electricity production, which amounts to 3262.17 DZD, and the annualised cost of investments, which amounts to 8.01×10^7 DZD.

To reach this objective value, the model returns an optimal investment as described in table 1. We do not consider the 7 kW capacity of the diesel generator as in investment, because it is already installed. Furthermore, the model has not optimised this facility's capacity.

Facility	Capacity	Investment Cost in DZD
PV	3932.90 kW	4.06×10^7
Electricity battery	3624.9 kWh	3.95×10^7

Table 1: Target Capacity by Facility.

Figure 5 in appendix C shows the hourly supply and demand by source. On average, PV produces 807.86 kWh of electricity per hour. The average hourly electricity generation from the diesel generation is as low as 0.04 kWh, highlighting how the investment in PV and battery facilities reduces the need to burn diesel to generate electricity. The diesel generator becomes a marginal electricity producing facility. Hence, the significantly low yearly cost of electricity mentioned above.

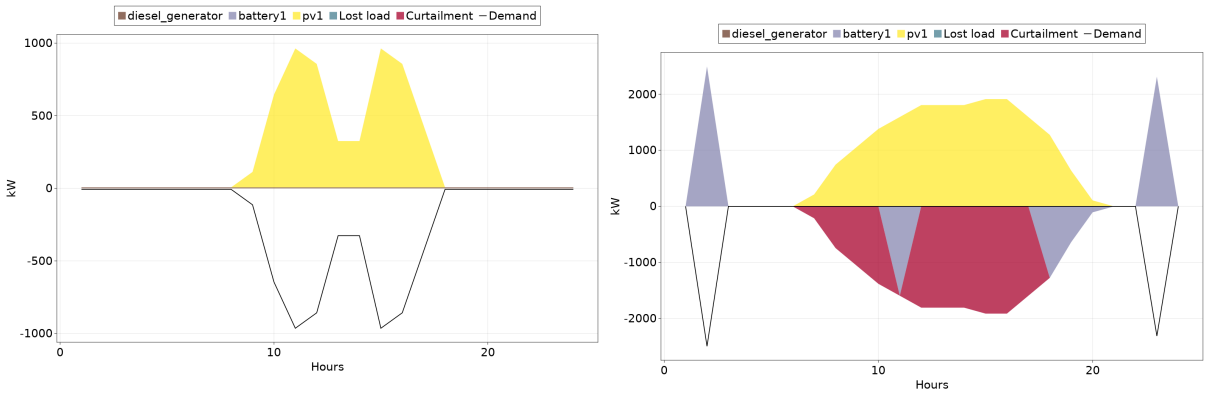
The maximum hourly curtailment amounts to 3932.90 kWh and the average hourly curtailment to 192.78 kWh. There is no lost load at any point in time. This is mostly achieved by using the battery to store electricity that is deployed when irrigation is most efficient or by pumping water to the water tank. Figures 6 and 7 in appendix C show the average daily level of the electricity and water storage facilities respectively.

In total, 3×10^6 m^3 of water are used to irrigate the field, that is 7.91% more than the total amount of water needed by the field. This results shows that the model can further be improved to reduce the water consumption, in case water is a scarce resource in the farm location. Setting the minimum daily irrigation constraint in equation 10 stricter to equality is an option. However,

23. All values in this section are rounded to the nearest 2 decimal places to improve readability.

we discarded it because the model becomes infeasible with the parameters we chose in section 4.

Figures 3a and 3b show the hourly electricity generation and demand on two example days: 1st January and 1st August. Here, we can clearly see the effect of the irrigation efficiency parameter. While in figure 3a demand perfectly matches the electricity production from PV, in figure 3b the PV production is partially used to charge the battery and mostly curtailed. The battery is then discharged early and late in the day, when the irrigation efficiency is the highest.



(a) 1st January

(b) 1st August.

Figure 3: Hourly Generation and Demand by Source on Two Different Days.

In this section, we have shown the results of the model when using the parameters defined in section 4. We have seen how an investment in PV and battery can reduce the variable cost of electricity to an almost negligible level. While the initial investment in PV and battery remains high, its advantages are proven: very low cost of electricity, low environmental impact and independence from the unreliable national grid can all be achieved. We acknowledge, however, that our model delivered an optimal solution using already known data about availability and other weather parameters. In a real-world application of the model, the user must take this into account, for example by expanding the model to consider different future unknown scenarios, in order to be better prepared to the fluctuations that affect non-dispatchable electricity generators.

6. Conclusion

We modelled a plant irrigation system based on solar energy. Our model both takes into account the availability of solar energy and it also optimises the timing to irrigate the plants, considering the evapotranspiration. There were two different alternatives to satisfy the electricity demand in absence of solar energy: storing energy through batteries and/or storing water through water tanks. However, the diesel generator is still present in the system, in case no other source is available. Diverse assumptions were required to run the model: some of them are strong due to the absence of data. Nonetheless, we managed to implement the model on Julia and run it. The results obtained prove that, in the given context of a small farm taking place in Algeria, an irrigation system based on solar energy is possible and and valuable.

Finally, it is worth to mention that our model can handle other general setup. However, we strongly recommend to use more detailed data, in order to improve the its reliability and applicability to reality.

Appendices

A. Evapotranspiration Model

Another important problem that required an in-depth literature review was the phenomena of water evaporation during the irrigation process.

According to Mittapalli 2015, evapotranspiration is the combination of soil evaporation and crop transpiration. In this paper, we consider the soil evapotranspiration to be equal to water-to-harvest evaporation, that is the amount of water that needs to be pumped in excess of the plane demand to water the crop sufficiently. This decision arises from two different considerations: first, the model used to compute the hourly soil evapotranspiration rate, that is the FAO-56 Penman-Monteith, assumes the presence of a hypothetical grass with a crop height of 0.12m. Second, the BELAIDOUNI Farm cultivates trees of olives, citrus fruits and grapes; meaning that, besides the harvest, grass is present in the cultivated area. Those two characteristics allow us to consider the soil transpiration as the grass one, and therefore what is left out of the equation is the soil evaporation, that we indeed assume to be equal to the water-to-harvest evaporation. However, both components must be taken into consideration when computing the necessary water in excess compared to the plane demand.

The methodology, as the assumptions used to apply the Penman-Monteith model, followed Mittapalli 2015 and López-Urrea et al. 2006.

The equation used to calculate ET_o, that is the evapotranspiration for each period, is:

$$ET_o := \frac{0.408\Delta\,(R_n - G) + \gamma \frac{900}{T+273} u_2\, VPD}{\Delta + \gamma\,(1 + 0.34u_2)} \in \mathbb{R},$$

where:

ET_o	$\equiv$ daily reference evapotranspiration in mm/d,
Δ	$\equiv$ slope vapour pressure curve in kPa/^{o}C,
R_n	$\equiv$ net radiation at the crop surface in MJ m^{-2}d^{-1},
G	$\equiv$ soil heat flux density in MJ m^{-2}d^{-1},
γ	$\equiv$ psychometric constant in kPa/^{o}C,
T	$\equiv$ air temperature at altitude z in ^{o}C,
u_2	$\equiv$ wind speed at altitude z in m/s,
$VPD := (e_s - e_a)$	$\equiv$ vapour pressure deficit in kPa.

To compute the different parameters, the used formulas are:

- $\Delta := \frac{4098\left(0.6108\exp\left(\frac{17.27T}{T+237.3}\right)\right)}{(T+237.3)^2}$.
- $R_n := (R_{ns} - R_{nl})$, where R_{ns} is the net solar radiation and R_{nl} is the net long wave radiation.

  $R_{ns} := (1-\alpha)R_s$, where R_s is the solar radiation and α is the albedo. We assume that $\alpha = 0.23$ for our case study.

  $R_{nl} := \sigma\left((T+273.16)^4\right)\left(0.34 - 0.14\sqrt{e_a}\right)\left(1.35\frac{R_s}{R_{so}} - 0.35\right)$, where R_{so} is the clear sky solar radiation and σ is the Stefan-Boltzmann constant.

  $R_{so} := \left(0.75 + 2\times10^{-5}z\right)R_a$, where R_a is the extraterrestrial solar radiation. To derive its value we exploit Gairaa and Bakelli 2013, who estimate R_{so} as $R_{so} = \frac{R_s}{K}$, where K is the clearness index, equal, on average, to approximately 0.67 according to Gairaa and Bakelli 2013 taking place in Algeria.
- The soil heat flux density G can be assumed to be approximately $G = 0.1R_n$, according to the research of Zahira, Abderrahmane, and Mederbal 2009 taking place at approximately 150km from the farm in question.
- $\gamma := 0.665\times10^{-3}P$, where P is the atmospheric pressure. P can be computed as $P = 101.3\left(\frac{293-0.0065z}{293}\right)^{5.26}$, where z is the altitude, which we assume to be $z = 2$ metres.
- $e_s(T) := 0.6108\exp\left(\frac{17.27T}{T+237.3}\right)$, and $e_a := e_s(T)\frac{RH_{hr}}{100}$, where RH_{hr} is the average hourly relative humidity.

In figure 4, we show the irrigation efficiency parameter computed for our case study. We have

aggregated the values by averaging them over each day for a better visualisation.

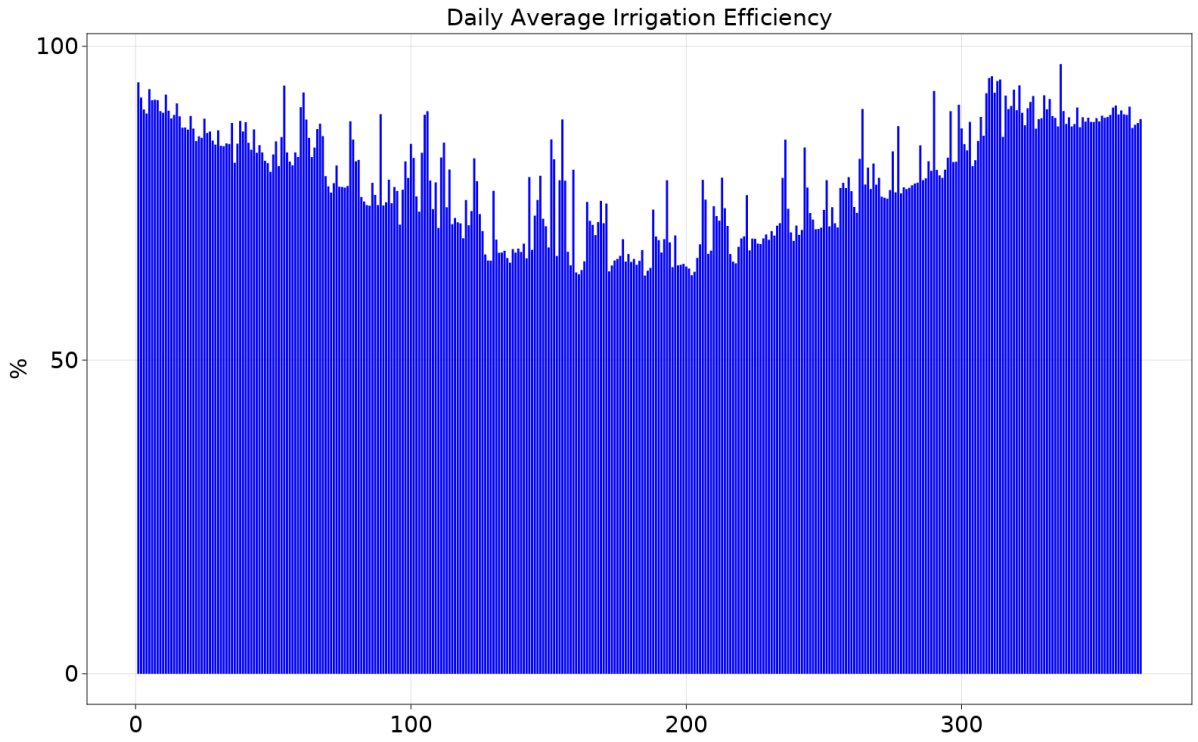


Figure 4: Daily Average Irrigation Efficiency.

Here, we see that the average water efficiency drops during the first half of the year and later rises until the end of year, with few exceptions. This is in line with our expectations, as temperatures rise during the first half of the year, peak in summer and then start to fall again until the end of the year. The opposite holds for precipitations, although this factor has a negligible effect in our case study, because the farm is located in a very dry area of the globe. The average irrigation efficiency over the whole optimisation timeframe is 79.1%.

B. Julia Model Implementation

Investment_Model.jl

```
# This is Julia program written by
# - Francisco Jose Manjon Cabeza Garcia,
# - Alessandro James Carenini,
# - Francesca Cioccarelli
# - and Gaspard Robert
# to implement the investment optimisation model described in our term paper for the energy modelling
# course at TU Berlin in the summer semester 2022. The code is mostly based on the code provided by
# the course lecturers in exercise session 5, in the Julia file called 05-invest01.jl.
#
# This optimisation model returns the optimal investments in dispatchable, storage and indispatchable
# electricity production facilities, as well as the optimal irrigation strategy for a farm to reduce the
# total cost of investments plus electricity production costs.

using JuMP # building models.
using JuMP.Containers # use JuMP containers for parameters.
using GLPK # solver for the JuMP model.
using CSV # readin of CSV files.
using DataFrames # data tables.
using CairoMakie # results plotting.

# ------------------------------------------------------------------------------------------------------------
# User parameters                                                                                            |
# ------------------------------------------------------------------------------------------------------------
# Define here the path to the folder containing the model parameters in csv files.
data_path = joinpath("Data")


# ------------------------------------------------------------------------------------------------------------
# Data imports                                                                                               |
# ------------------------------------------------------------------------------------------------------------

# DataFrame with data for temperature, precipitation, humidity, solar energy, solar radiation, etc.
# for each time step. This data file can be obtained at https://www.visualcrossing.com/weather-history/
# The model needs the data from the following columns (create a Weather_data.csv file with those columns and
# column names, if you are not using the data from VisualCrossing):
# - temp:               (float) temperature in degrees Celsius.
# - precip:             (float) precipitation in mm.
# - windspeed:          (float) wind speed in km/h.
# - solarradiation:     (float) solar radiation.
# - solarenergy:        (float) solar energy.
df_weather_time = CSV.read(joinpath(data_path, "Weather_data.csv"), DataFrame)

# DataFrame with parameters for supply side technologies: dispatchable, storage and indispatchable facilities.
# The data frame contains the following columns:
# - technology:         (string) name of the technology. PV and wind facility's names must start with "pv" or
#                       "wind", so that the program can identify them as such and use the correct availability
#                       parameter in the computations.
# - dispatchable:       (bool) if the facility is dispatchable or not.
# - renewable:          (bool) if the facility produces renewable energy or not.
# - variable_cost:      (float) variable cost of electricity production per unit of capacity.
# - fixed_cost:         (float) fixed cost of the facility per unit of capacity.
# - eff:                (float) efficiency factor.
# - inv_power:          (float) investment cost per unit of capacity for electricity producing facilities. Set
#                       this parameter to 0 for storage facilites.
# - inv_storage:        (float) investment cost per unit of capacity for storage facilities. Set this parameter
#                       to 0 for non-storage facilites.
# - lifetime:           (float) lifetime of the facility.
# - emission_factor:    (float) emission factor of the facility.
# - fuel_cost:          (float) fuel cost per unit of produced electricity.
# - surface_covered:    (float) surface covered by the facility per unit of capacity.
# - max_surface:        (float) maximum surface constraint of the facility.
# - g_max:              (float) if set to a number strictly greater than 0, the facility is considered to be
#                       installed, and its capacity is not optimised by the model, i.e., is assumed to be fixed.
df_supply_tech = CSV.read(joinpath(data_path, "Supply_technology.csv"), DataFrame)

# DataFrame with the parameters of the water side technologies: water pumps and water tanks. The DataFrame
# contains the following columns:
# - technology:         (string) name of the technology.
# - storage:            (bool) if the facility is a water tank or not.
```

```
# - energy_cost:        (float) cost of pumping one unit of water to the field in units of electricity.
# - max_capacity:       (float) maximum capacity if the facility is a water tank.
df_water_tech = CSV.read(joinpath(data_path, "Water_technology.csv"), DataFrame)

# DataFrame with the hourly electricity demand of each of the other demand sources as columns.
df_other_demand = CSV.read(joinpath(data_path, "Other_demand_data.csv"), DataFrame)

# DataFrame with other parameters. The DataFrame contains the following rows:
# - CO2_price:          (float) CO2 price.
# - Lost_load_cost:     (float) lost load cost.
# - Interest_rate:      (float) interest rate.
# - Daily_effective_water (float) daily effective water need of the field.
# - evtr_alpha:         (float) alpha parameter for the evapotranspiration computation.
# - altitude:           (float) altitude parameter for the evapotranspiration computation.
df_other_parameters = CSV.read(joinpath(data_path, "Other_parameters.csv"), DataFrame)

# Save the CO2 price to a variable.
co2_price = float.(df_other_parameters[df_other_parameters.Parameter .== "CO2_price", "Value"])[1]
# Save the lost load cost to a variable.
lost_load_cost = float.(df_other_parameters[df_other_parameters.Parameter .== "Lost_load_cost", "Value"])[1]
# Save the interest rate to a variable.
interest_rate = float.(df_other_parameters[df_other_parameters.Parameter .== "Interest_rate", "Value"])[1]
# Save the daily effective irrigation needed by the field to a variable.
DEW = float.(df_other_parameters[df_other_parameters.Parameter .== "Daily_effective_water", "Value"])[1]
# Save the alpha parameter for the evapotranspiration computation.
alpha = float.(df_other_parameters[df_other_parameters.Parameter .== "evtr_alpha", "Value"])[1]
# Save the altitude parameter for the evapotranspiration computation.
altitude = float.(df_other_parameters[df_other_parameters.Parameter .== "altitude", "Value"])[1]

# ------------------------------------------------------------------------------------------------------------------
# Model parameter definitions                                                                                      |
# ------------------------------------------------------------------------------------------------------------------

# Number of time steps.
T = Vector{Int64}(1:nrow(df_weather_time))
# Set of electricity production facilities: dispatchable, storage and indispatchable facilities.
P = string.(df_supply_tech.technology)
# Set of already installed facilities for which the installed capacity will not be optimised, i.e., is fixed.
INSTP = string.(df_supply_tech[df_supply_tech.g_max .> 0, "technology"])
# Set of dispatchable electricity production facilities.
DISP = string.(df_supply_tech[df_supply_tech.dispatchable .== 1, "technology"])
# Set of non-dispatchable electricity production facilities.
# Contrary to the definition in the methodology, electricity storage facilities are contained in this set.
NDISP = string.(df_supply_tech[df_supply_tech.dispatchable .== 0, "technology"])
# Set of electricity storage facilities.
S = string.(df_supply_tech[df_supply_tech.inv_storage .> 0, "technology"])

# Get availability from the weather data DataFrame for wind and solar.
avail_types = Vector{String}(["pv", "wind"])
avail = DenseAxisArray{Float64}(undef, T, avail_types)
for i in T
    avail[i, "wind"] = coalesce(df_weather_time.windspeed[i],0)
    avail[i, "pv"] = coalesce(df_weather_time.solarenergy[i],0)
end
# Normalise the values for availability.
avail[:, "wind"] = avail[:, "wind"] ./ maximum(avail[:, "wind"])
avail[:, "pv"] = avail[:, "pv"] ./ maximum(avail[:, "pv"])

# Compute the marginal cost of electricity production of each dispatchable facility.
mc = map(eachrow(filter(x-> x.dispatchable == 1, df_supply_tech))) do row
    fc = row.fuel_cost
    ef = row.emission_factor
    eff = row.eff
    vc = row.variable_cost

    return fc/eff + co2_price*ef/eff + vc
end

mc = DenseAxisArray(mc, DISP)

# Save the efficiency factor of each electricity producing facility.
eff = DenseAxisArray(df_supply_tech.eff, P)

# Define two methods to compute the annualised investment cost per unit
# of capacity of electricity producing facilities.
annuity(i,lifetime) = i * ((1 + i)^lifetime) / (((1 + i)^lifetime) - 1)
```

```
calc_inv_cost(oc, i, lt) = oc * annuity(i, lt)

# Compute the annualised investment cost per unit of capacity of electricity producing facilities.
invest_cost = map(
    row-> row.g_max > 0 ? row.fixed_cost : calc_inv_cost(maximum([row.inv_power, row.inv_storage]),
                                                         interest_rate,
                                                         row.lifetime) + row.fixed_cost,
    eachrow(df_supply_tech)
)
invest_cost = DenseAxisArray(invest_cost, P)

# Maximum capacity of installed facilities.
inst_g_max = map(
    row -> row.g_max,
    eachrow(filter(x-> x.g_max > 0, df_supply_tech))
)
inst_g_max = DenseAxisArray(inst_g_max, INSTP)

# Surface covered per unit of capacity.
SW = DenseAxisArray(df_supply_tech.surface_covered, P)
# Maximum surface available for each electricity producing facility.
MSW = DenseAxisArray(df_supply_tech.max_surface, P)


# Set of water facilities.
W = string.(df_water_tech.technology)
# Set of water tanks.
WTa = string.(df_water_tech[df_water_tech.storage .== 1, "technology"])
# Set of water pumps.
WPu = string.(df_water_tech[df_water_tech.storage .== 0, "technology"])

# Cost of pumping one unit of water volume in units of electricity.
WEC = DenseAxisArray(df_water_tech.energy_cost, W)
# Water tank's capacities.
WTC = map(
    row -> row.max_capacity,
    eachrow(filter(x-> x.storage > 0, df_water_tech))
)
WTC = DenseAxisArray(WTC, WTa)

# Convert wind speed from km/h to m/s.
df_weather_time[!, :windspeed] = df_weather_time[!, :windspeed] ./ 3.6

# Delta: slope vapour pressure curve.
df_weather_time[!, :delta] =  map(
    row -> 4098 * 0.6108
           * exp(17.27*row.temp/(row.temp+273.3))
           / (row.temp+273.3)^2,
    eachrow(df_weather_time)
)

# R_{ns}: net solar radiation.
df_weather_time[!, :Rns] = map(
    row -> (1 - alpha) * row.solarradiation,
    eachrow(df_weather_time)
)

df_weather_time[!,:ea] = map(
    row -> 0.6108*exp(17.27*row.temp/(row.temp+237.3)) # e^0(T)
           * row.humidity/100,
    eachrow(df_weather_time)
)

# R_{nl}: net long wave radiation.
df_weather_time[!, :Rnl] = map(
    row -> 4.903 * 10^(-9)
            *(row.temp + 273.16)^4
            * (0.34 - 0.14 * sqrt(row.ea) * (1.35 * 0.67 - 0.35)),
    eachrow(df_weather_time)
)

# R_n: net radiation at the crop surface.
df_weather_time[!, :Rn] = map(
    row -> row.Rns - row.Rnl,
    eachrow(df_weather_time)
)
```

```
# G: soil heat flux density.
df_weather_time[!, :G] = map(
    row -> row.Rn * 0.1,
    eachrow(df_weather_time)
)

# Gamma: psychometric constant.
gamma = 0.665*10^(-3)* 101.3*((293 - 0.0065*altitude)/293)^(5.26)

# Hourly reference evapotranspiration in mm.
evapotranspiration = map(
    row-> (0.408*row.delta*(row.Rn - row.G)
           + gamma * 900/(row.temp+273) * row.windspeed * (0.6108*exp(17.27*row.temp/(row.temp+237.3))
           - row.ea))
    / (row.delta + gamma * (1 + 0.34*row.windspeed)),
    eachrow(df_weather_time)
)
minevotrans = minimum(evapotranspiration)
println("Min. evapotranspiration: $minevotrans.")

# Shift evapotranspiration figures to get all non-negative numbers.
if sum(evapotranspiration .< 0) > 0
    WE = evapotranspiration .+ abs.(minimum(evapotranspiration))
end
# Substract the precipitation in mm from the evapotranspiration.
WE = WE - df_weather_time.precip
# Normalise the irrigation efficiency.
WE = WE ./ maximum(WE)
# Compute the irrigation efficiency as the inverse of the evapotranspiration.
WE = 1 .- WE

# Add up the demand from all other demand sources hourly.
OD = sum(eachcol(df_other_demand[:, Cols(x -> startswith(x, "load"))]))

# ------------------------------------------------------------------------------------------------------------
# Model definitions                                                                                          |
# ------------------------------------------------------------------------------------------------------------

# Define a new JuMP model.
m = Model(GLPK.Optimizer)

# Define the model variables.
@variables m begin
    # Generation from dispatchable facilities (dispatchable generators AND storages).
    # Contrary to the definition in the methodology, G is not defined for non-dispatchable facilities.
    G[DISP, T] >= 0
    # Energy used for charging electricity storages.
    CH[S, T] >= 0
    # Electricity storage levels.
    L[S, T] >= 0
    # Installed maximum capacities.
    PW[P] >= 0
    # Water pumped by all water facilities.
    WP[W, T] >= 0
    # Water used for filling water tanks.
    WCH[WTa, T] >= 0
    # Water tank levels.
    WL[WTa, T] >= 0
    #
    # Variables to improve the probability of finding a feasible solution.
    # Curtailement.
    CU[T] >= 0
    # Lost load.
    LL[T] >= 0
end

# Expresion for the electricity production from non-dispatchable facilities.
@expression(m, feedin[p=NDISP, t=T],
    PW[p] * (startswith(p, "pv") ?  avail[t,"pv"] : avail[t,"wind"])
)

# Expression for the hourly electricity demand from water pumps and water tanks.
@expression(m, elec_demand_irrigation[t=T],
    sum(WP[w,t] * WEC[w] for w in W)
)
```

```
# Expression with the yearly electricity cost without investments.
@expression(m, yearly_elec_cost,
    8760/length(T) * (
        sum(mc[p] * G[p,t] for p in DISP , t in T)
        + sum(lost_load_cost * LL[t] for t in T)
    )
)

# Expression with the yearly investment cost.
@expression(m, yearly_investment_cost,
    sum(invest_cost[p] * PW[p] for p in P)
)

# Expression with the total water irrigated to the field over the optimisation period.
@expression(m, total_water_irrigated,
    sum(WE[t] * (sum(WP[w,t] for w in W)
        - sum(WCH[wta, t] for wta in WTa))
        for t in T)
)

# Define the objective function of the model.
@objective(m, Min,
    8760/length(T) * (
        sum(mc[p] * G[p,t] for p in DISP , t in T)
        + sum(lost_load_cost * LL[t] for t in T)
    )
    + sum(invest_cost[p] * PW[p] for p in P)
)

# Energy balance constraints.
@constraint(m,
    EnergyBalance[t=T],
    sum(G[p,t] for p in DISP)
    + sum(feedin[nd, t] for nd in NDISP)
    + LL[t]
    ==
    elec_demand_irrigation[t]
    + OD[t]
    + sum(CH[s,t] for s in S)
    + CU[t]
)

# Maximum generation constraints for dispatchable generators
@constraint(m, MaxGeneration[p=DISP,t=T], G[p,t] <= PW[p])
# Fixed maximum generation from already installed facilities.
@constraint(m, InstalledGeneration[inst=INSTP], PW[inst] == inst_g_max[inst])

# Maximum electricity storage constraints for charging.
@constraint(m, MaxCharge[s=S,t=T], CH[s,t] <= PW[s])
# Maximum electricity storage constraints for storage levels.
@constraint(m, MaxStorage[s=S,t=T], L[s,t] <= PW[s])

# Define a function that gives us the next hour of the set T
next_hour(t) = t == T[end] ? T[1] : T[findfirst(isequal(t), T) + 1]
# Electricity storage facility level difference constraints.
@constraint(m, StorageBalance[s=S, t=T],
    L[s,next_hour(t)]
    ==
    L[s,t]
    + eff[s] * CH[s,t]
    - G[s,t]/eff[s]
)

# Electricity storage initial level constraints.
@constraint(m, StartingStorageLevel[s=S],
    L[s,1] == 0
)

# Contraints for area covered by generation facilities.
@constraint(m, AreaCovered[p=P],
    PW[p] * SW[p] <= MSW[p]
)

# Constaints for minimum irrigation.
Days = Vector{Int64}(1:nrow(df_weather_time)/24)
```

```
get_day_hours(d) = Vector{Int64}(24*(d-1)+1 : 24*d)
@constraint(m, IrrigationBalance[d=Days],
    sum(WE[t] * (sum(WP[w,t] for w in W)
                 - sum(WCH[wta, t] for wta in WTa))
        for t in get_day_hours(d))
    >=
    DEW
)

# Maximum water storage constaints for charging.
@constraint(m, MaxWaterCharge[wta=WTa,t=T],
    WCH[wta,t] <= WTC[wta]
)
# Maximum water storage constraints for storage levels.
@constraint(m, MaxWaterStorage[wta=WTa,t=T],
    WL[wta,t] <= WTC[wta]
)
# Water storage facility level difference constraints.
@constraint(m, WaterStorageBalance[wta=WTa,t=T],
    WL[wta,next_hour(t)]
    ==
    WE[t] * WL[wta,t] + WCH[wta,t] - WP[wta,t]
)

# Water storage initial level constraints.
@constraint(m, StartingWaterStorageLevel[wta=WTa],
    WL[wta,1] == 0
)

# ------------------------------------------------------------------------------------------------------------------------
# Optimise and print summary of the optmisation process and result.                                                      |
# ------------------------------------------------------------------------------------------------------------------------

optimize!(m)
println(solution_summary(m))

# ------------------------------------------------------------------------------------------------------------------------
# Compute and print some statistical figures.                                                                            |
# ------------------------------------------------------------------------------------------------------------------------

include(joinpath("Statistical_Figures.jl"))
Compute_Model_Statistics()

# ------------------------------------------------------------------------------------------------------------------------
# Plot results                                                                                                           |
# ------------------------------------------------------------------------------------------------------------------------

include(joinpath("Plot.jl"))
fig = plot_result();
```

Plot.jl

```
function plot_result()

    # --------------------------------------------------------------------------------------------------------------------
    # Get the values of the optimal variables.                                                                           |
    # --------------------------------------------------------------------------------------------------------------------

    generation = value.(m[:G])
    storage_gen = value.(m[:CH])
    curtailment = value.(m[:CU])
    storage_level = value.(m[:L])
    lost_load = value.(m[:LL])

    power = value.(m[:PW])[vcat(DISP, NDISP)]
    #storage = value.(m[:PW])[S]

    generation_combined = vcat(generation, value.(m[:feedin]), lost_load.data')
    cum_generation = vcat(zeros(1, length(T)), cumsum(generation_combined, dims=1))

    cum_charge = vcat(
        zeros(1, length(T)),
        -cumsum(vcat(storage_gen.data, curtailment.data'), dims=1)
```

```
)

water_storage_level = value.(m[:WL])

# ------------------------------------------------------------------------------------------------------
# Plot hourly electricity generation and demand by source.                                             |
# ------------------------------------------------------------------------------------------------------

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "kW", xlabel="Hours")
colors = ["#754937", "#8e8eb5", "#ffeb3b", "#518696"]
labels = vcat(string.(P), "Lost load")

legend_items = []
legend_items_labels = String[]
for row in 1:size(cum_generation, 1)-1
    b = band!(
        ax,
        1:length(T),
        cum_generation[row,:],
        cum_generation[row+1,:],
        color=(colors[row], 0.8),
        linewidth=0
    )
    push!(legend_items, b)
    push!(legend_items_labels, labels[row])
end

colors2 = ["#8e8eb5", "#ae123a"]
for row in 1:size(cum_charge, 1)-1
    b = band!(
        ax,
        1:length(T),
        cum_charge[row,:],
        cum_charge[row+1,:],
        color=(colors2[row], 0.8),
        linewidth=0
    )
    row == size(cum_charge, 1)-1 && push!(legend_items, b)
end

push!(legend_items_labels, "Curtailment")

#hlines!(ax, 0, color=:black, linewidth=2)

demand = value.(m[:elec_demand_irrigation]) .+ OD

p_demand = lines!(
    ax,
    -demand.data,
    color=:black,
    label="Demand",
    linewidth=2
)
push!(legend_items, p_demand)
push!(legend_items_labels, "Demand")

Legend(
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Hourly_generation_and_demand_by_source.png", fig)

# ------------------------------------------------------------------------------------------------------
# Plot mean day hourly electricity generation and demand by source.                                    |
# ------------------------------------------------------------------------------------------------------

meanDay_generation_combined = Matrix{Float64}(undef, size(generation_combined, 1), 24)
    for row in 1:size(meanDay_generation_combined, 1)
    for hour in 1:24
        hourIndices = [hour+(i-1)*24 for i in 1:Int(length(T)/24)]
        meanDay_generation_combined[row, hour] = 1/(length(T)/24) * sum(generation_combined[row, hourIndices])
```

```
    end
end
meanDay_cum_generation = vcat(zeros(1, 24), cumsum(meanDay_generation_combined, dims=1))

meanDay_storage_gen = Matrix{Float64}(undef, size(storage_gen.data, 1), 24)
for row in 1:size(meanDay_storage_gen, 1)
    for hour in 1:24
        hourIndices = [hour+(i-1)*24 for i in 1:Int(length(T)/24)]
        meanDay_storage_gen[row, hour] = 1/(length(T)/24) * sum(storage_gen.data[row, hourIndices])
    end
end
meandDay_curtailment = Vector{Float64}(undef, 24)
for hour in 1:24
    hourIndices = [hour+(i-1)*24 for i in 1:Int(length(T)/24)]
    meandDay_curtailment[hour] = 1/(length(T)/24) * sum(curtailment.data[hourIndices])
end
meanDay_cum_charge = vcat(
    zeros(1, 24),
    -cumsum(vcat(meanDay_storage_gen, meandDay_curtailment'), dims=1)
)

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "kW", xlabel="Hours")
colors = ["#754937", "#8e8eb5", "#ffeb3b", "#518696"]
labels = vcat(string.(P), "Lost load")

legend_items = []
legend_items_labels = String[]
for row in 1:size(meanDay_cum_generation, 1)-1
    b = band!(
        ax,
        1:24,
        meanDay_cum_generation[row,:],
        meanDay_cum_generation[row+1,:],
        color=(colors[row], 0.8),
        linewidth=0
    )
    push!(legend_items, b)
    push!(legend_items_labels, labels[row])
end

colors2 = ["#8e8eb5", "#ae123a"]
for row in 1:size(meanDay_cum_charge, 1)-1
    b = band!(
        ax,
        1:24,
        meanDay_cum_charge[row,:],
        meanDay_cum_charge[row+1,:],
        color=(colors2[row], 0.8),
        linewidth=0
    )
    row == size(meanDay_cum_charge, 1)-1 && push!(legend_items, b)
end

push!(legend_items_labels, "Curtailment")

hlines!(ax, 0, color=:black, linewidth=2)

demand = value.(m[:elec_demand_irrigation]) .+ OD
    meanDay_demand = Vector{Float64}(undef, 24)
for hour in 1:24
    hourIndices = [hour+(i-1)*24 for i in 1:Int(length(T)/24)]
    meanDay_demand[hour] = 1/(length(T)/24) * sum(demand.data[hourIndices])
end

p_demand = lines!(
    ax,
    -meanDay_demand,
    color=:black,
    label="Demand",
    linewidth=2
)
push!(legend_items, p_demand)
push!(legend_items_labels, "Demand")

Legend(
```

```
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Mean_day_hourly_generation_and_demand_by_source.png", fig)

# ------------------------------------------------------------------------------------------------------------
# Plot hourly electricity generation and demand by source for two example days.                              |
# ------------------------------------------------------------------------------------------------------------
daysoftheyear = [1, 213]
for day in daysoftheyear
    dayhours = 1+(day-1)*24:day*24

    fig = Figure(resolution = (1600, 1000), fontsize = 30)
    ax = Axis(fig[2,1], ylabel = "kW", xlabel="Hours")
    colors = ["#754937", "#8e8eb5", "#ffeb3b", "#518696"]
    labels = vcat(string.(P), "Lost load")

    legend_items = []
    legend_items_labels = String[]
    for row in 1:size(cum_generation, 1)-1
        b = band!(
            ax,
            1:24,
            cum_generation[row, dayhours],
            cum_generation[row+1, dayhours],
            color=(colors[row], 0.8),
            linewidth=0
        )
        push!(legend_items, b)
        push!(legend_items_labels, labels[row])
    end

    colors2 = ["#8e8eb5", "#ae123a"]
    for row in 1:size(cum_charge, 1)-1
        b = band!(
            ax,
            1:24,
            cum_charge[row,dayhours],
            cum_charge[row+1,dayhours],
            color=(colors2[row], 0.8),
            linewidth=0
        )
        row == size(cum_charge, 1)-1 && push!(legend_items, b)
    end

    push!(legend_items_labels, "Curtailment")

    #hlines!(ax, 0, color=:black, linewidth=2)

    demand = value.(m[:elec_demand_irrigation]) .+ OD

    p_demand = lines!(
        ax,
        -demand.data[dayhours],
        color=:black,
        label="Demand",
        linewidth=2
    )
    push!(legend_items, p_demand)
    push!(legend_items_labels, "Demand")

    Legend(
        fig[1,1],
        legend_items,
        legend_items_labels,
        orientation = :horizontal,
        nbanks = 1
    )

    save("Results/Day_of_Year_$day"*"_hourly_generation_and_demand_by_source.png", fig)
end
```

```
# ------------------------------------------------------------------------------------------------------
# Plot hourly electricity storage levels.                                                              |
# ------------------------------------------------------------------------------------------------------

legend_items = []
legend_items_labels = String[]

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "kWh", title="Storage Levels")#, height=120)
for (i,s) in enumerate(S)
    storage_lines = lines!(ax, storage_level[s,:].data, color=colors2[i])
    push!(legend_items, storage_lines)
    push!(legend_items_labels, S[i])
end

Legend(
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Hourly_electricity_storage_levels.png", fig)

# ------------------------------------------------------------------------------------------------------
# Plot daily electricity storage levels.                                                               |
# ------------------------------------------------------------------------------------------------------

dailyStorage_level = DenseAxisArray{Float64}(undef, S, 1:Int(length(T)/24))
for s in S
    for i in 1:Int(length(T)/24)
        dailyStorage_level[s, i] = 1/24 * sum(storage_level[s, 24*(i-1)+1 : 24*i])
    end
end

legend_items = []
legend_items_labels = String[]

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "kWh", title="Daily Average Storage Levels")#, height=120)
for (i,s) in enumerate(S)
    storage_lines = lines!(ax, dailyStorage_level[s,:].data, color=colors2[i])
    push!(legend_items, storage_lines)
    push!(legend_items_labels, S[i])
end

Legend(
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Daily_average_electricity_storage_levels.png", fig)

# ------------------------------------------------------------------------------------------------------
# Plot target capacity by facility.                                                                    |
# ------------------------------------------------------------------------------------------------------

fig = Figure(resolution = (1600, 1000), fontsize = 30)
power_ticks = axes(power, 1)
ax = Axis(
    fig[1,1],
    title="Capacity",
    xticks=(1:length(power_ticks), power_ticks),
    ylabel="kW",
    xticklabelrotation=pi/4
)

barplot!(
    ax,
    power.data,
    color=["#754937", "#8e8eb5", "#ffeb3b"]#, "#518696"]
)
```

```
save("Results/Target_capacity_by_facility.png", fig)

# ------------------------------------------------------------------------------------------------------------
# Plot hourly water storage levels.                                                                          |
# ------------------------------------------------------------------------------------------------------------

legend_items = []
legend_items_labels = String[]

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "m^3", title="Water Storage Levels")#, height=120)
for (i,wta) in enumerate(WTa)
    storage_lines = lines!(ax, water_storage_level[wta,:].data, color=:blue)
    push!(legend_items, storage_lines)
    push!(legend_items_labels, WTa[i])
end

Legend(
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Hourly_water_storage_levels.png", fig)

# ------------------------------------------------------------------------------------------------------------
# Plot daily water storage levels.                                                                           |
# ------------------------------------------------------------------------------------------------------------

dailyWater_storage_level = DenseAxisArray{Float64}(undef, WTa, 1:Int(length(T)/24))
for wta in WTa
    for i in 1:Int(length(T)/24)
        dailyWater_storage_level[wta, i] = 1/24 * sum(water_storage_level[wta, 24*(i-1)+1 : 24*i])
    end
end

legend_items = []
legend_items_labels = String[]

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[2,1], ylabel = "m^3", title="Daily Average Water Storage Levels")#, height=120)
for (i,wta) in enumerate(WTa)
    storage_lines = lines!(ax, dailyWater_storage_level[wta,:].data, color=:blue)
    push!(legend_items, storage_lines)
    push!(legend_items_labels, WTa[i])
end

Legend(
    fig[1,1],
    legend_items,
    legend_items_labels,
    orientation = :horizontal,
    nbanks = 1
)

save("Results/Daily_average_water_storage_levels.png", fig)

# ------------------------------------------------------------------------------------------------------------
# Plot hourly water efficiency.                                                                              |
# ------------------------------------------------------------------------------------------------------------

fig = Figure(resolution = (1600, 1000), fontsize = 30)
ax = Axis(fig[1,1], ylabel = "%", title="Irrigation Efficiency")
barplot!(ax, WE .* 100, color=:blue)

save("Results/Hourly_irrigation_efficiency.png", fig)

# ------------------------------------------------------------------------------------------------------------
# Plot daily water efficiency.                                                                               |
# ------------------------------------------------------------------------------------------------------------

dailyWE = zeros(Int(length(T)/24))
for i in 1:Int(length(T)/24)
```

```
        dailyWE[i] = 1/24 * sum(WE[24*(i-1)+1 : 24*i])
    end

    fig = Figure(resolution = (1600, 1000), fontsize=30)
    ax = Axis(fig[1,1], ylabel = "%", title="Daily Average Irrigation Efficiency")
    barplot!(ax, dailyWE .* 100, color=:blue)

    save("Results/Daily_average_irrigation_efficiency.png", fig)

    return


end
```

Statistical_Figures.jl

```
function Compute_Model_Statistics()
    pv_facilites = map(x -> startswith(x, "pv") ? x : "", NDISP)
    if sum(pv_facilites .!= "") > 0
        avg_pv_generation = 1/length(T) * sum(value.(m[:feedin])[pv_facilites, :])
        println("Average generation from PV: $avg_pv_generation kWh.")
    end

    wind_facilites = map(x -> startswith(x, "wind") ? x : "", NDISP)
    if sum(wind_facilites .!= "") > 0
        avg_wind_generation = 1/length(T) * sum(value.(m[:feedin])[wind_facilites, :])
        println("Average hourly generation from wind: $avg_wind_generation kWh.")
    end

    avg_disp_non_bat_generation = 1/length(T) * (sum(value.(m[:G])[DISP,:]) - sum(value.(m[:G])[S,:]))
    println("Avg. hourly generation from dispatchable and non-battery facilities: $avg_disp_non_bat_generation kWh.")

    avg_lost_load = 1/length(T) * sum(value.(m[:LL]))
    max_lost_load = maximum(value.(m[:LL]))
    println("Average hourly lost load: $avg_lost_load kWh. Maximum lost load: $max_lost_load kWh.")

    avg_curtailment = 1/length(T) * sum(value.(m[:CU]))
    max_curtailment = maximum(value.(m[:CU]))
    println("Average hourly curtailment: $avg_curtailment kWh. Maximum curtailment: $max_curtailment kWh.")

    avg_irrigation_eff = 1/length(T) * sum(WE) * 100
    println("Average irrigation efficiency: $avg_irrigation_eff %.")

    yearly_elec_cost = value.(m[:yearly_elec_cost])
    yearly_investment_cost = value.(m[:yearly_investment_cost])

    println("Total yearly electricity cost: $yearly_elec_cost DZD.")
    println("Total yearly investment cost (computed as annuity): $yearly_investment_cost DZD.")

    total_irrigation = value.(m[:total_water_irrigated])
    total_irrigation_text = "Total water irrigated: $total_irrigation m^3."
    irrigated_to_target_ratio = total_irrigation / (length(T)/24 * DEW)
    irr_ratio_text = "$irrigated_to_target_ratio times the effective water needed."
    println(total_irrigation_text*irr_ratio_text)
end
```

C. Further Results Figures

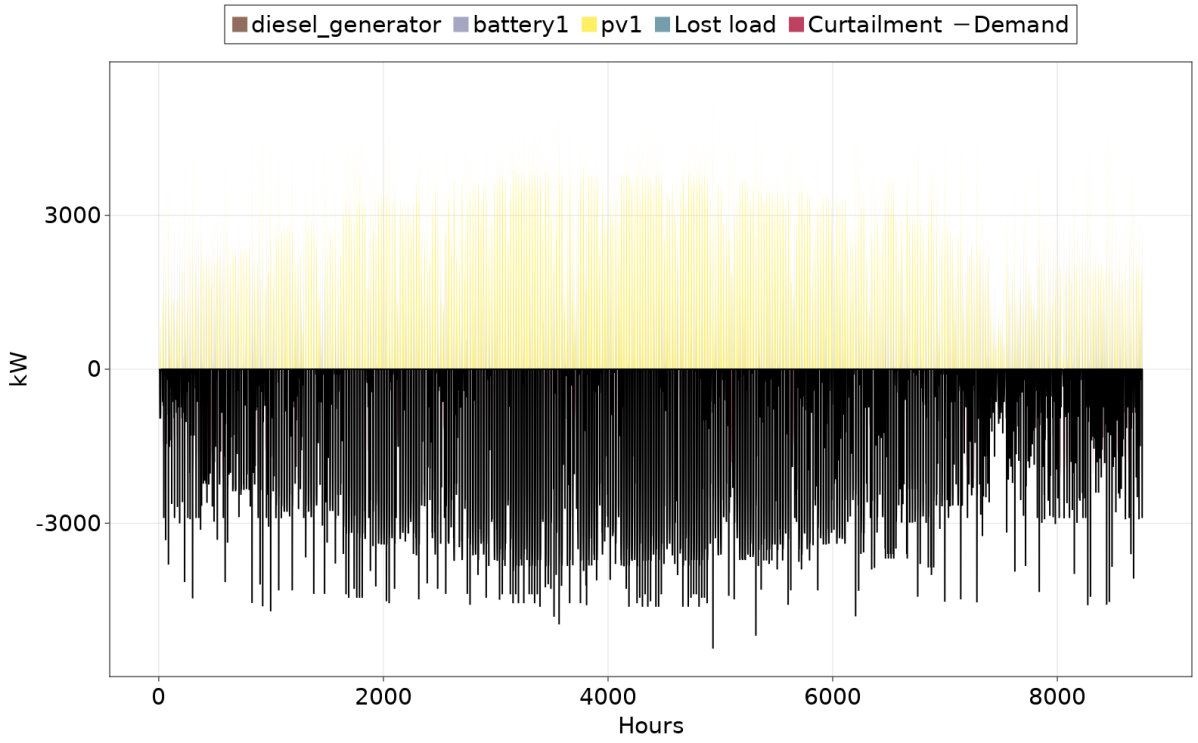


Figure 5: Hourly Generation and Demand by Source.

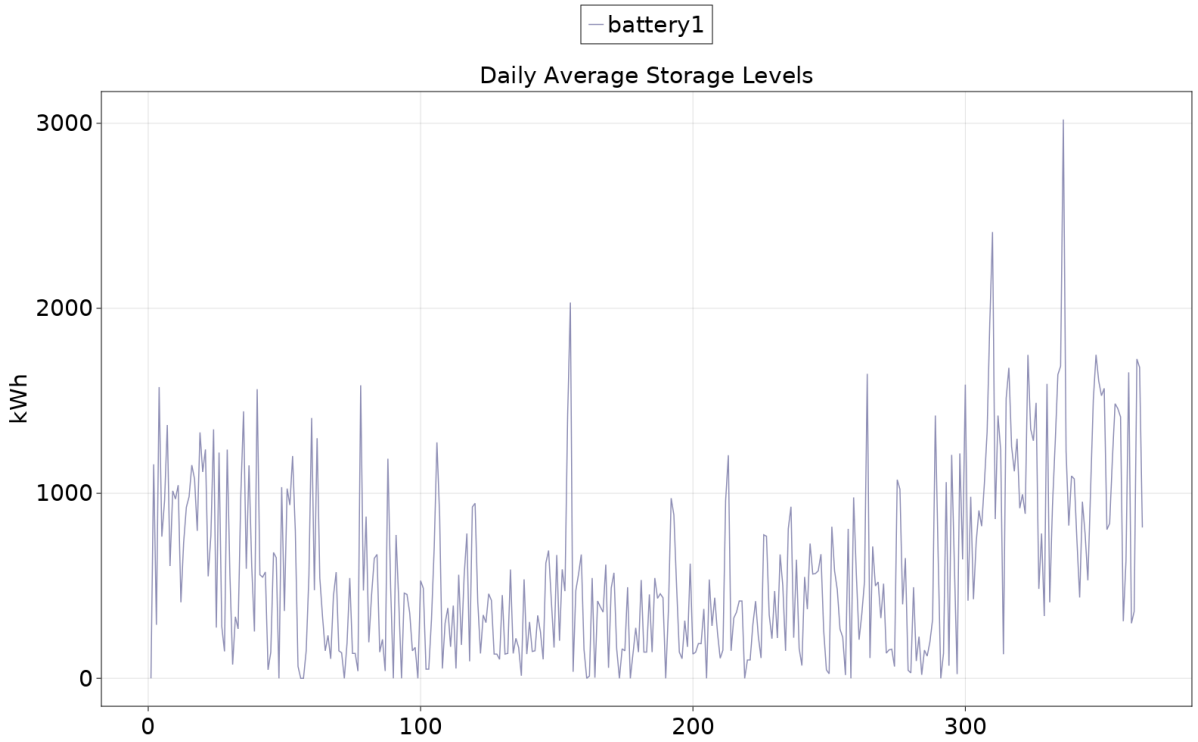


Figure 6: Daily Average Electricity Storage Levels.

Figure 8 shows the hourly generation and demand averaged over all the days in the optimisation period, i.e., it shows the average day. We can clearly see the peak in electricity production from PV during the daylight hours, which is mostly used to charge the battery or curtailed. Electricity is mostly demanded during the daylight hours, but we can also identify the use of the battery in

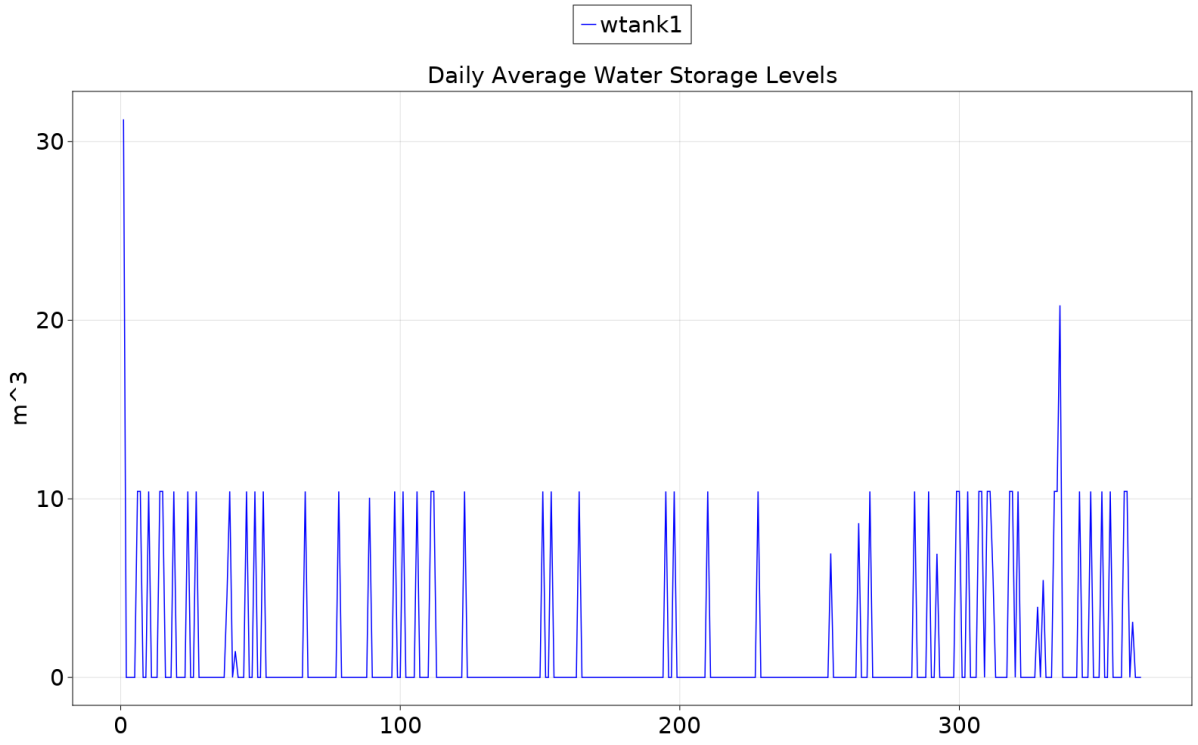


Figure 7: Daily Average Water Storage Levels.

the very early and very late hours of the day to satisfy some demand.

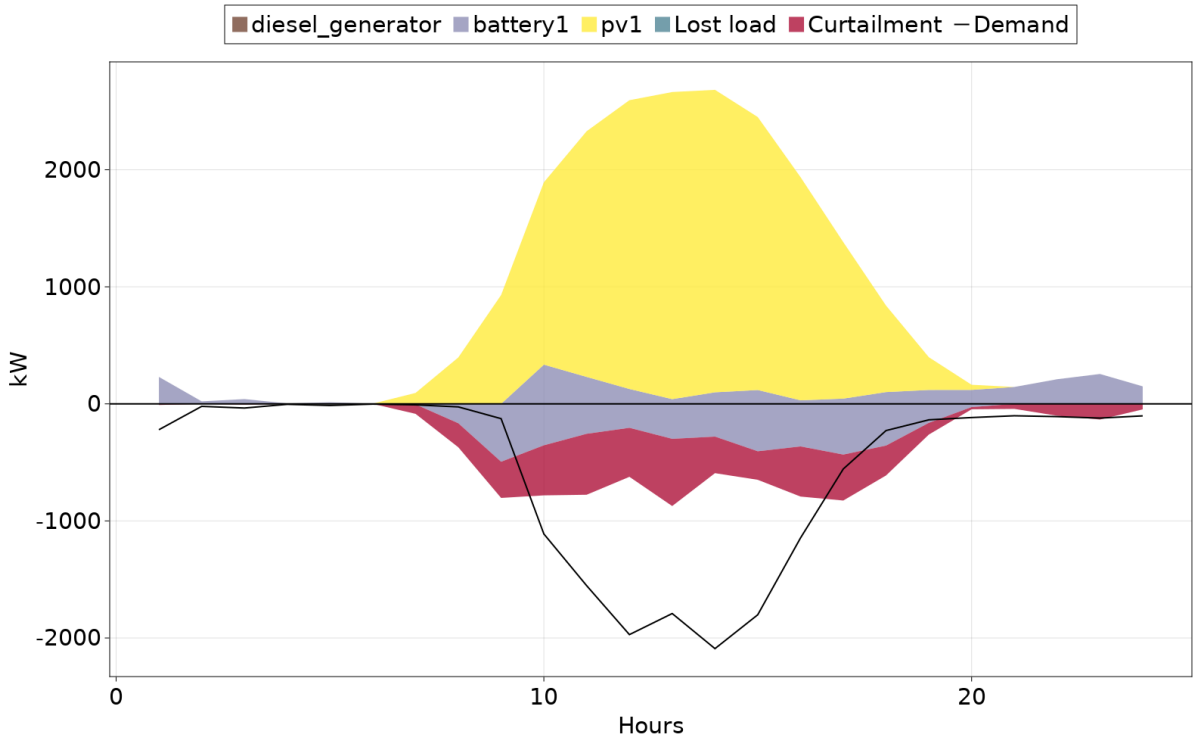


Figure 8: Mean Day Hourly Generation and Demand by Source.

References

Bekrit, Leila and Mesli, Zohra. 2022. "Study and sizing of a solar power station for an autoconsumption in the agricultural field."

Dunning, Iain, Huchette, Joey, and Lubin, Miles. 2017. "JuMP: A Modeling Language for Mathematical Optimization." SIAM Review 59 (2): 295–320.

Gairaa, Kacem and Bakelli, Yahia. 2013. "Solar Energy Potential Assessment in the Algerian South Area: Case of Ghardaïa Region." Jayanta Deb Mondol Volume 2013.

Giri, Nimay Chandra, Rana, Debaraj, Mish, Siba Prasad, and Pani, Bibhuti Bhushan. 2020. "Efficacy of Solar Powered Water Pumps for Rural Farmers in Odisha; India." PalArch's Journal of Archaeology of Egypt / Egyptology 17 (9): 2215–2224.

Global Agriculture Information Network. 2021. "The Algerian Agricultural Roadmap 2020–2024." PalArch's Journal of Archaeology of Egypt / Egyptology, number No. AG2021-0006.

International Renewable Energy Agency. 2017. "Electricity storage and renewables: Costs and markets to 2030."

López-Urrea, R., Santa Olalla, F. Martín de, Fabeiro, C., and Moratalla, A. 2006. "An evaluation of two hourly reference evapotranspiration equations for semiarid conditions." Agricultural Water Management 86 (3): 277–282.

Mittapalli, Giridhar. 2015. "Estimation of Hourly Reference Evapotranspiration." International Journal of Scientific and Engineering Research Volume 6: 1681.

US Department of Agriculture. 2020. "Algeria - Agricultural Sector." Privacy Shield Frameworks.

Visual Crossing Corporation. 2021. "Visual Crossing Weather". https://www.visualcrossing.com/weather-history/Algiers%2CAlgeria.

Zahira, Souidi, Abderrahmane, Hamimed, and Mederbal, Khalladi. 2009. "Mapping Latent Heat Flux in the Western Forest Covered Regions of Algeria Using Remote Sensing Data and a Spatialized Model." Remote Sensing.