Electrical grid simulation in the COLMENA framework

Pablo de Juan Vela [1]
[1]eRoots, Barcelona, Spain

November 5, 2024

Chapter 1

Initial Documentation

1.1 Introduction

ANDES Curent [1] is a python package that can be used to model, simulate, and analyze power systems. The package uses predefined models that simulate different elements of an electrical grid. Additionally, the package allows the user to define custom models which relative ease. ANDES stands out from other simulation tools for the use of a hybrid symbolic-numeric framework for modeling differential algebraic equations (DAEs). This project will present a test case for the COLMENA framework in the context of electrical grids using the simulation tools provided by ANDES. The goal of the project is to showcase the capabilities of COLMENA in different contexts. For this purpose we will simulate different electrical grids where the different devices will act as agents of the COLMENA colony. The agents will change their operating operating point dependent on the value of specific grid metrics and from the control from COLMENA. In this study, we will give an overview on how time domain simulation are done in electrical grids, then how we can adapt the COLMENA framework to power grids and finally we explain the results of simulation in preset test cases.

1.2 Modelling and Definitions

1.2.1 Electrical Grid Modelling

We define an electrical grid as a set of nodes called buses with devices attached to them, the nodes are interconnected with lines. These devices include generators, loads or others types of devices. We define each device as a set of state and algebraic variables coupled to their respective equations. This set of equations gives rise to a DAE for every specific model. Let $x_{mdl}, \dot{x}_{mdl} \in \mathbb{R}^n, x_{mdl} \in \mathbb{R}^m$ the state, the state derivatives and the algebraic variables for a given model respectively, x, y the whole's system variables and $f_{mdl}(x_{mdl}, y_{mdl}, x, y), g_{mdl}(x_{mdl}, y_{mdl}, x, y)$

the state and algebraic function for the model "mdl":

$$\dot{x}_{mdl} = f_{mdl}(x_{mdl}, y_{mdl}, x, y) \tag{1.1}$$
$$0 = g_{mdl}(x_{mdl}, y_{mdl}, x, y) \tag{1.2}$$

For every device that belongs to the same model the DAE definition is identical, only the dependencies in the functions f_{mdl}, g_{mdl} change. Combining all the DAEs from the individual devices defines a general DAE for the whole system. This DAE takes the following expression, $\forall t \in \mathcal{T}$:

$$\dot{x}(t) = f(x(t), y(t)) \tag{1.3}$$
$$0 = g(x(t), y(t)) \tag{1.4}$$

This set of expressions is what will define the dynamic behavior of the grid in a time domain simulation. However, before initializing we need to perform a power flow simulation. This consists in considering the grid only in steady state. In practical terms, this means dropping the state equations from the DAE and considering only the algebraic equations since in practice all the dynamic effects have already settled.

$$0 = g(x, y) \tag{1.5}$$

Once this system is solved the time domain simulation can start properly. This is also the case in ANDES where the Power Flow system.PFlow simulation needs to be run before the time domain simulation system.TDS. This step ensures the simulation starts with a feasible and steady state x and y. The time domain simulation will update the state variables iterative through a specific time steps $\{t_0, \ldots, t_T\}$.

On the modeling side, lines and buses are fundamental components of an electrical grid, and they play a critical role in grid simulation because they represent the network's physical and electrical structure and how power is transmitted and distributed throughout the system.

Buses represent the connection points in the network for different devices (like generators, loads...). The devices connected to bus will be influenced by the buses current (in volts). Buses are also the nodes of the grid in a topological sense. Additionally, they serve as reference points for power balance calculations (power consumed and produced must be equal to zero) as well as for setting the current balance equations through Kirchhoff's law (the sum of currents in a bus must be 0).

1.2.2 Agents & Models

The main objective of COLMENA in relation to the ANDES framework is developping a model in which the agents adapt their behavior in relation to the information they can access through the specific environmental communication. In the context of electrical grids, the different devices connected to the grid will act as the independent agents of the colony. We can have the following roles for agents in the simulation.

- Generators
- Converters
- Transformers
- Loads
- Switches
- Lines

These roles correspond to a different type of device in the grid. The agents of the colony have fixed roles inside the grid but can have different tasks depending on the information they gather through the *Edge* communication and the behavior that COLMENA gives assigns to them.

Given the context of the agents we can provide definitions for the agents in the context of electrical grids.

Definition 1 *A device d is a tuple $\{x, y, p\}$. Such that $x \in \mathbb{R}^n$ is the state variable, $y \in \mathbb{R}^m$ are the algebraic variables and $p \in \mathbb{R}^q$ the parameters.*

Definition 2 *An operating point $o(d)$ for a device p is a tuple $\{f(x, y), g(x, y)\}$. Such that $f(x, y) : \mathbb{R}^{n+m} \to \mathbb{R}^n$ is the state function, $g(x, y) : \mathbb{R}^{n+m} \to \mathbb{R}^m$ is the algebraic function.*

Definition 3 *An agent $\mathcal{A}$ is the tuple $\{d, \mathcal{R}(d)\}$ where the d is a device and $\mathcal{R}(d)$ is the set of operating points available to that type of device. When considering an agent in a fixed time t: there can only be one operating point active. Therefore, $\mathcal{A}(t) = \{d, o\}$ where o is an operating point is an agent at time t. This combination of device and operating point defines a DAE, where the state and algebraic functions are dependent on x, y but also on the current operating point. We can consider the set $\mathcal{R}(d)$ as the role given to that device.*

$$\dot{x} = f(x, y, r(t)) \tag{1.6}$$
$$0 = g(x, y, r(t)) \tag{1.7}$$
$$\forall t \in \mathcal{T} \tag{1.8}$$

This DAE describes completely the behavior of the agent in the simulation for every possible operating point of a given agent.

1.2.3 Environments

Environments are a key part of COLMENA's structure. They provide the context and tools to share data collected during the simulation. These environments are key to ensure the distributed nature of COLMENA.

Definition 4 *A environment $\mathcal{E}$ is a set of agents such that they share some type of information (to be specified further), $\mathcal{E} = \{d_1, ..., d_n\}$.*

We define environments in multiple contexts, most commonly by the geographical proximity of the agents to each other. Such as a set of agents belonging to the same region of the electrical grid. Additionally, we can consider dynamic environments where the environment set depends on the time step or on a given agent role's. For example, we can consider the environment $\mathcal{E}_{neighbour}(d) = \{d_i | d_i \text{ is a neighbour of } d\}$.

ANDES' structure allows us to define static environments with relative ease by giving the information before starting the simulation. We can also exploit the dependencies between the agents to define more complex environments in ANDES such as this model searching the neighbors of a specific bus (the bus in `IdxParam`) by first accessing the lines that are connected to the bus (`self.lines1` and `self.lines2`) and then searching the adjacent buses.

```
class Neighbourhood(Model, ModelData):
    def __init__(self, system, config):
        ModelData.__init__(self)
        self.bus = IdxParam(model='Bus')
        self.auxline = IdxParam(model='Line')
        self.auxbus = IdxParam(model='Bus')
        Model.__init__(self, system, config)
        self.lines1 = DeviceFinder(self.auxline, link = self.
            bus, idx_name ='bus1')
        self.lines2 = DeviceFinder(self.auxline, link = self.
            bus, idx_name ='bus2')
        self.bus1 = ExtParam(model='Line', src='bus1', indexer
            =self.lines2)
        self.bus2 = ExtParam(model='Line', src='bus2', indexer
            =self.lines1)
```

We can also use the agents roles' as a way to define an environment. In this case the environment $\mathcal{E}$ could be define in the following way:

$$\mathcal{E}_{generator} := \{a_1, ..., a_k | a_i \text{ is a generator } \forall i \in [k]\} \tag{1.9}$$

This environment would enable the agents with a generator role to exchange information over their specific1 *Edge*.

1.2.4 Metrics & KPIs

In the simulation we are proposing every device is connected to the grid through a bus. The bus and the device share the shame voltage. In most cases, the de-

vice needs to run at some specific interval for proper functioning. This is the case for example for nominal voltage, nominal power (for generators) or nominal current for lines. Functioning at, or close to those nominal values is important to not damage those components.

On a system wide context, the grid's frequency is also key for a good performance of the grid. The grid frequency needs to be as close to the nominal value (50 Hz) as possible and even slight mismatches can be problematic.

Additionally, we will define the different metrics in relations to their environment. This means that every metric is measured in a specific environment. This way we can define different types of environments for the metrics:

- Local: Measurements that can be done in the same device.
- Semi-Local: Measurements that can be done in the devices connected to the same bus.
- Regional: Measurements that can be done in the devices from the same given area.
- Categorical: Measurements that can be done in the devices of the same type.
- Adjacent: Measurements that can be done in the devices that are just one connection away.

These different types of environments define the available channels of communications for the agents. Moreover, the metrics' types are not mutually exclusive from the perspective of a single agent. For example for a synchronous generator its angular frequency is both local (as it is a state) and regional.

Let's now give a specific definition of a metric:

Definition 5 *Let $\mathcal{G}$ a grid with state x and $t \in \mathcal{T}$ a time step and $\mathcal{E} = \{d_1, ..., d_n\}$ a set of devices, then a metric m over the environment $\mathcal{E}$ is the function $m(x, t, \mathcal{E}) = \sum_{d_i \in \mathcal{E}} f(x_i, y_i)$.*

In the context of the environment defined in **??** one possible metric for the environment could be the mean frequency ω over the generators, where N is the cardinal of the environment set:

$$m_\omega(x, t, \mathcal{E}) = \sum_{g \in \mathcal{E}_{generators}} \frac{\omega_g}{N} \tag{1.10}$$

The different metrics can be used to build Key Performance Indicators (KPI) that describe the performance of parts of the grid or the whole grid. In the following section, we will explain different ways to build KPIs from existing

metrics. Given the numerous amounts of metrics the possible amount of KPIs that we can work with is very big, we will just explain a couple of examples.

Definition 6 *Let $\mathcal{G}$ a grid, $t \in \mathcal{T}$ a time step, and m a metric. Then a KPI κ is a function of the metric over different time steps*

$$\kappa(m, t) := \sum_{\tau \leq t} \kappa_t(m(x, \tau, \mathcal{E})) \tag{1.11}$$

If we use the previously defined metric we can define a new KPI κ_{gen} as the number of times ω was outside the admissible interval $I = [\omega_{min}, \omega_{max}]$ over a given time scale.

$$\kappa_{gen}(m_{gen}, t) := \sum_{\tau \in \{t-k,\ldots,t\}} Indicator_I(m(x, \tau, \mathcal{E})) \tag{1.12}$$

$$Indicator_I(x) = \begin{cases} 1 & \text{if } x \in I \\ 0 & \text{if } x \notin I \end{cases} \tag{1.13}$$

1.2.5 Role Definition

We have seen the different components and behaviors of the grid simulation we plan to run. Let's now give some example of actual roles and their different operation states.

Switches

Switches are controllable elements that control the connection state of a line between two buses. In this case the operating states are just 'Open' or 'Closed'. In the open state no current travels directly between the buses while in the Closed state the line works at normal operation. Additionally, switches can also be present connected to individual devices. In this case, triggering the switch just disconnects the device to the bus it is connected.

In the context of COLMENA, the switch can be operating closed if the current stays in an admissible range and open if not. This can be useful to protect the electrical components of the devices or even the integrity of the line itself.

Synchronous Generators

Generators are devices that inject power to the grid. A synchronous generator injects the power through a rotating part that spins synchronously to the grid's frequency. The synchronous generators in the simulation will be defined by two internal states: $\omega \in \mathbb{R}$ the angular velocity and $\theta \in \mathbb{R}$.

Converters

A Voltage Source Converter (VSC) is a device that converts a direct current to an Alternative Current (AC). They are commonly used to track the angle of the grid at a bus and inject (or absorb) active and reactive power from an energy source. They are usually paired with DC sources of energy as a way to connect them to the grid. The VSC can operate in two modes: 'Grid following'(GFL) and 'Grid forming'(GFM).

- Grid Following: In the GFL mode, the converter tracks the grid's frequency and injects a controlled power, acting as an ideal current source.
- Grid Forming: In the GFM mode, the converter creates its own frequency and imposes a voltage differential acting as an ideal voltage source.

In terms of control, both GFM and GFL have a reference value with a control. The control of the devices aims to get as close to the reference value as possible. For the GFL mode the reference value usually refers to the power injected, while for the GFM its the frequency or voltage. This GFM behavior is quite close to one of a typical synchronous generator.

Distributed generation with Batteries

The modelling of distributed generation in ANDES usually combines the generation considered as a DC current source and a battery that is connected to a AC-DC converter. The converter that is paired to this ensemble can control both the active and reactive power that is injected to the grid and that is stored in the battery (if present). The control of the setup is defined by the parameters $\gamma_p, \gamma_q \in [0, 1]$. These parameters define which proportion of the active and reactive power respectively and generated by the setup is injected to the grid. We can therefore define different operating points for the distributed generation depending on the power injected. The power that is not injected is then saved by the battery. This can be expressed as these three different operating points:

- $\gamma_p = 1$ all of the power generated is injected to the grid.
- $\gamma_p = 0$ all of the power generated is stored in the battery.
- $\gamma_p = 0.5$ half of the power generated is injected to the grid and half is stored in the battery.

Loads

A load is an electrical model that is connected to a bus and consumes a given amount of active and reactive($P(MW), Q(Mvar)$ respectively). In the context of COLMENA, the agents with a load role can have varying values of P and Q depending on their operating points, the control assigned by COLMENA and the info available to then through their environments. These adaptable response can be very useful to the grid in order to adapt to drops in generation or line faults.

1.3 Applying COLMENA to electrical grids

Models' equations in ANDES define the behavior of the devices.These equations govern the evolution of internal states as well as the internal states of the devices. We will perform changes in these equations in order to model the different roles.

1.3.1 Method 1: Equation Modification

Background

As explained earlier, a system of differential algebraic equations (DAE) defines the behavior of a given model. In this case, we want to combine two already existing behaviors into a single model. We will consider these two behaviors, A and B, as different roles of the same model. This can be done by combining both sets of equations into a single DAE:

$$\dot{x} = u_a f_a(x, y) + u_b f_b(x, y) \tag{1.14}$$
$$0 = u_a g_a(x, y) + u_b g_b(x, y) \tag{1.15}$$
$$u_a, u_b \in \{0, 1\}, \quad u_a + u_b = 1 \tag{1.16}$$

Where the original DAE for the behaviors A and B, and the discrete variables u_a and u_b are as follows:

$$\dot{x} = u_a f_a(x, y) + u_b f_b(x, y) \tag{1.17}$$
$$0 = u_a g_a(x, y) + u_b g_b(x, y) \tag{1.18}$$

$$\dot{x} = u_a f_a(x, y) + u_b f_b(x, y) \tag{1.19}$$
$$0 = u_a g_a(x, y) + u_b g_b(x, y) \tag{1.20}$$

$$u_a(t) = \begin{cases} 1 & \text{if behavior is A} \\ 0 & \text{if else} \end{cases} \qquad u_b(t) = \begin{cases} 1 & \text{if behavior is B} \\ 0 & \text{if else} \end{cases}$$

It is important to note that this equation transformation is only been possible since the state and algebraic variables are identical. This is usually not a problem since the different roles are usually behaviors of the same model but one could consider combining DAE with different states or variables. In this case, combining the different roles is harder and would necessitate of using coupling functions $f_{a,B}(x_a, y_a)$ and $f_{b,A}(x_b, y_b)$ that define how the states for the role a evolve when the role B is activated and vice versa.

For example, if we consider that both states are independent, which means

that $\frac{\partial x_a}{\partial t \partial x_b}(t, x_a, y_a) = 0$ and $\frac{\partial x_b}{\partial t \partial x_a}(t, x_b, y_b) = 0$. Then the resulting DAE is:

$$\dot{x_a} = u_a f_{a,A}(x_a, y_a) + u_b f_{a,B}(x_a, y_a) \tag{1.21}$$
$$\dot{x_b} = u_a f_{b,A}(x_a, y_a) + u_b f_{b,B}(x_a, y_a) \tag{1.22}$$
$$0 = g_a(x_a, y_a) \tag{1.23}$$
$$0 = g_b(x_b, y_b) \tag{1.24}$$

Implementation & Examples

Let us now give an overview on how to combine different equations to define a new model with two operating points in ANDES and what it means for the simulation. For this, we will be working with the model 'GENROU' provided by ANDES that represents a synchronous generator. From this initial model we want to define a new model that includes 2 different operating points. For this we will define a new discrete variable inside the model to define the select for the specific operating point as well as setting up the necessary new equations and parameters in the model.

For this we define a new class class GENROU_bimode in the proper ANDES folder. The new class inherits all the attributes from the original generator model. In this case, we define the limiter as a discrete variable that depends on the voltage of the bus that the generator is connected to. If said voltage belongs to the interval $[v_{min}, v_{max}]$ then the device will work at the operating point 'A' otherwise it will work at the operating point 'B'.In this configuration the agent only changes its configuration depending on information that it has direct access to. This can be an advantage to ensure that the reaction is dependent only on the agent's environment.

Alternatively, we can consider u_a, u_b as parameters that can be changed by the COLMENA's algorithm internal logic. The changes will be the result of some changes in the KPI. In this case, it makes more sense to change the parameter during each time step. We will explore how to do this in a later section.

```
class GENROU_bimode(GENROU):
    def __init__(self, system, config):
        super().__init__(system, config)
        self.vlimiter = Limiter(u=self.v, lower=vmin,
            upper=vmax,
        enable=True)
        ua = 'vlimiter_zu + vlimiter_zl'
        ub = 'vlimiter_zi'

        #Alternatively, If we want ua ub as parameters
        self.ua = NumParam(info = 'discrete parameter')
        self.ub = NumParam(info = 'discrete parameter')
```

Once we have defined the discrete markers. We proceed to redefine the DAE's equations. We first define new parameters attributes in the class: for each parameter we define 2 new parameters for the operating points A and B by appending 'A' or 'B' respectively to the original name. Then we redefine the equation by modifying the attributes v_str, e_str of the variables and including both operating points. v_str defines the initialization equation an v_str the state and algebraic equations. By using the symbolic nature of ANDES, combining these equations consists of just combining different strings in a specific manner and making sure all used variables actually belonging to the model. The complete code looks like:

```
#WE REDEFINE THE PARAMETERS ATRIBUTES
    for name in gen_mode1._all_params().keys():
        if name in exceptions:
            continue
        var1 = getattr(gen_mode1, name)
        var2 = getattr(gen_mode2, name)
        name_a = name + "a"
        name_b = name + "b"
        var1.name = name_a
        var2.name = name_b

        delattr(self, name)
        setattr(self, name_a, var1)
        setattr(self, name_b, var2)

#WE CHANGE THE EQUATION'S EXPRESSION
for i, var in enumerate(self._states_and_ext().values()):
        name = var.name
        e_eq1 = var.e_str
        e_eq2 = var.e_str
        v_eq1 = var.v_str
        v_eq2 = var.v_str
        for j, param in enumerate(all_params.values()):
            name = param.name
            pattern_a = re.escape(name) + r'(?!' + re.
                escape("a") + r')'
            pattern_b = re.escape(name) + r'(?!' + re.
                escape("b") + r')'
            append_letter_a =
                create_append_letter_function("a")
            append_letter_b =
                create_append_letter_function("b")
            e_eq1 = re.sub(pattern_a, append_letter_a,
                e_eq1)
            e_eq2 = re.sub(pattern_b, append_letter_b,
                e_eq2)
            v_eq1 = re.sub(pattern_a, append_letter_a,
```

```
        v_eq1)
    v_eq2 = re.sub(pattern_b, append_letter_b,
        v_eq2)

v_out = "(v_zl-+-v_zu)"
v_in = "v_zi"
var.e_str = v_out + "*" + e_eq1 + "-+-" +  v_in +
    "*" + e_eq2
var.v_str = v_out + "*" + v_eq1 + "-+-" +  v_in +
    "*" + v_eq2
```

It is important to remark that with this method the parameters for the different operating states are set during the definition of the grid before the simulation. This means that this method could be less flexible than the ones presented next.

Case Study 1

For this case study we use the grid defined in Kundur two area system [2]. this grid consists of 10 buses divided in 2 areas. Each area includes two synchronous generators. Additionally, this simulation includes a line failure at a specific time step (Line 8, at $t = 2s$) The objective of this case study is to showcase the change of behavior of the agent when a specific condition is respected.

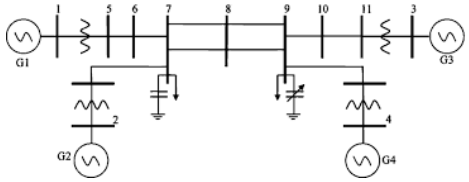

Figure 1.1: Kundur grid scheme. [2]

This grid contains several devices including synchronous generators. To test this approach we define a new generator model called "GENROU bimode". This model has two different operating states that differ in the value of the parameter M (the inertia of the rotor). The operating condition depends on ω the angular velocity of the generator's rotor. The angular velocity of the generator is the local measurement that the agent is adapting it's reacting to. We could imagine however an agent reacting to other measurements. Additionally, the derivative of ω depends on M which will make the difference in the operating point very visible. Here is a summary of the simulation parameters and their interactions:

Operating State	Operating Condition	Parameter M value
A	$\omega \in [1-\varepsilon, 1+\varepsilon]$	12
B	$\omega \notin [1-\varepsilon, 1+\varepsilon]$	120

Table 1.1: Bimodal generator operating state summary.

$$\dot{\omega} = \frac{f(x,y)}{M} \tag{1.25}$$

In this specific case we substitute one of the initial generators by a bimode generator. What we expect to see is the generator's angular velocity ω having 2 distinct behaviors. We run a time domain simulation to confirm our hypothesis. The simulation is run for over 20 seconds. The result conform to our hypothesis.

Additionally, we can clearly see 2 discontinuities in the derivative of ω. First at $t = 2s$ when the line fault is triggered and then at around $t = 3s$ when ω goes above the admissible interval ($\omega \geq \omega_{max} = 1.003 p.u.$).

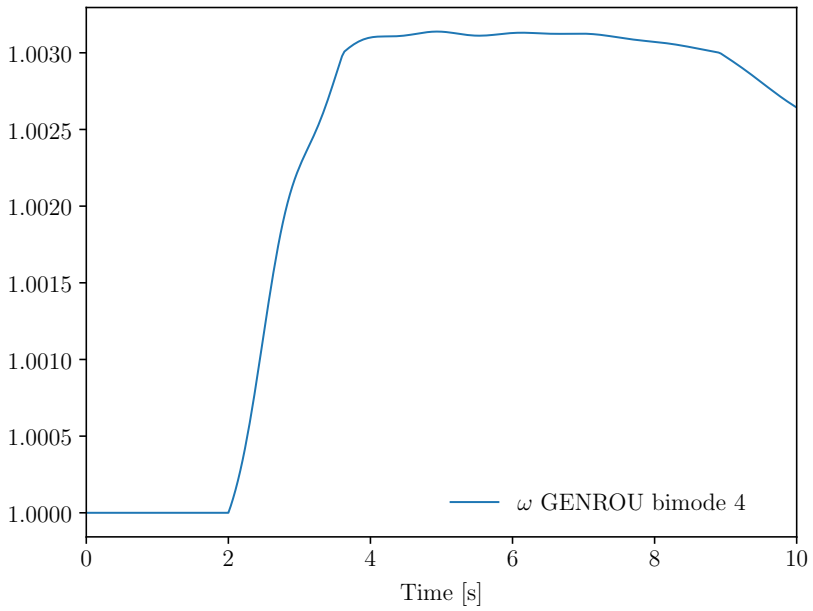

Figure 1.2: Time series of the distributed generator's frequency.

1.3.2 Method 2: Time Domain solver change

Another method to implement the different operating points for every agent is to directly change the parameters of the agent if necessary during each discrete

time step of the solving algorithm. The algorithm follows these steps: it first checks the operating conditions and then performs the necessary changes. In pseudo-code the algorithm looks as follows:

Algorithm 1 TDS COLMENA

```
Initialize grid states x to x_0 through Power Flow results.
t ← 0
while t ≤ t_end do
    for agent in the set of Agents do
        if Operating Condition A(agent) is respected at time t then
            mdl.param_M ← mdl.param_M_a
        else
            mdl.param_M ← mdl.param_M_b
        end if
    end for
    Perform Trapezoid Step
    Compute x_{t+1}, y_{t+1}
    t ← t + h
end while
Finalize the result
```

In this case, the implementation is more flexible and we can imagine different changes or operating conditions that can be checked during each time step. However, the constraint of only basing the decisions on the context depends on how you check the condition and not directly on the model definition as before.

Case Study 2

For this case study, we consider the 14 bus grid IEEE-14 [3]. We consider a unique distributed generator which control the active and reactive power being injected to the grid. Here the agent with the role of the generator will have two operating points: one where he has an active power reference of P_0 and another with $-P_0$. The generator will change its operating state at $t = 10s$. This is represented by the change in the parameter γ_p from 0.1 to -0.1. The simulation runs for 20s.

Operating State	Operating Condition	Parameter $gamma_p$ value
A	$t \leq 10$	0.1
B	$t \geq 10$	-0.1

Table 1.2: Distributed generator state summary.

As expected, we observe the change in operating time in the agent at $t = 10s$. The change in frequency is what we expect from the change in power output.

We also note that the initial frequency was of $60Hz$ which is the standard in the USA instead of the $50Hz$ that are used in Europe.

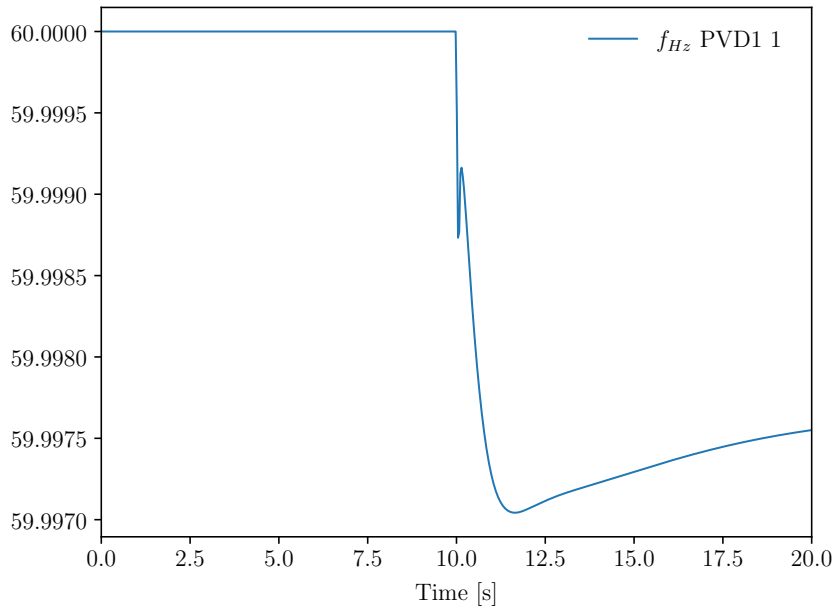

Figure 1.3: Time series of the distributed generator's frequency.

1.3.3 Future Simulations

The objective is to eventually run the simulation in larger grids with a larger number of agents. For this we can use other test grids such as the IEEE 118 bus case [4]. This test grid is composed by 118 buses, 54 generators and 99 Loads. Additionally, some of the generators are synchronous condensers who essentially are synchronous motors whose shaft is not connected to any mechanical load. Their main function is to provide reactive power compensation. Moreover, it uses the same models as the ones used in the previous tests cases so transferring the already developed models is feasible.

1.3.4 COLMENA Integration & Future Work

We have seen two approaches that are useful in knowing how to simulate electrical grids with agents with multiple operating points (Case Study 1) and how to change parameters or other values during the simulation itself (Case Study 2). We propose a mixed solution that is inspired from the previous works but also tries to integrate itself into the existing COLMENA framework. The AN-

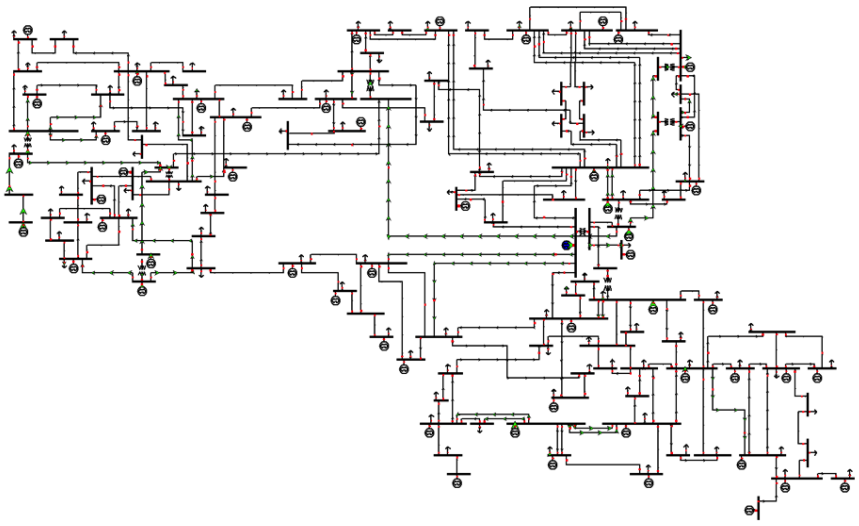

Figure 1.4: IEEE118 grid scheme. [4]

DES package as well as other tools simulate the grid up to a certain point while COLMENA is looking to simulate on a continuous time. For this we propose a combination of the frameworks in which COLMENA will call ANDES at irregular intervals of time and ANDES will run the dynamic simulation until a new steady-state point is reached.

For this, we define that agents can be available or locked. We consider that after a change of operating state the agent becomes active. This can be thought as being locked to the given operating point for a certain amount of time. After this time t_{lock}, the agent becomes available again.

The proposed algorithm integrates COLMENA into the simulation. COLMENA selects the operating points for the different agents in the necessary step. It's also important to remark that there is a power flow simulation that is performed at the very beginning. This step ensures the grid starts the simulation from a feasible and stable state. Specifically it performs a steady state simulation that aims to find a simulation such that the device's states are constant, that is $f(x, y) = 0$.

Performing the power flow simulation of a grid is considerably faster than doing a time domain simulation. Therefore, we could also consider a variation on the previous algorithm in which instead of computing the updated state in every new time step we compute the steady state directly and use it as. This simplification could be a possible alternative if COLMENA calls the simulation with enough time between each iteration so that we can consider that the grid

Algorithm 2 TDS COLMENA-ANDES integration

```
Initialize grid states x to x_0 through Power Flow results.
t ← 0
while not Stop do
    while No Agent is available do
        Run ANDES Normally
        t ← t + h
    end while
    Go back to COLMENA
    for Agent in Agents that are available do
        Update Agent's operating point
    end for
    t ← t + h
end while
Finalize the result
```

achieves the steady state. Otherwise, the classic time domain simulation is more precise since it takes into account the dynamics of the electrical devices.

Bibliography

[1] H. Cui, "Andes, model references," 2022. Accessed: Aug. 13, 2024. [Online]. Available: https://docs.andes.app/en/latest/modelref.html.

[2] Q. Huang, "Rlgc repository," 2022. Accessed: Aug. 13, 2024. [Online]. Available: https://github.com/RLGC-Project/RLGC.

[3] U. of Washington, "Power systems test case archive," 2022. Accessed: Aug. 13, 2024. [Online]. Available: https://labs.ece.uw.edu/pstca/.

[4] U. of Washington, "Power systems test case archive," 2022. Accessed: Aug. 13, 2024. [Online]. Available: https://labs.ece.uw.edu/pstca/pf118/pg_tca118bus.htm.