Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DESARROLLO DE SOFTWARE PARA EL DISEÑO DE SISTEMAS DE AUTOCONSUMO FOTOVOLTAICO

Estudiante *Casado, Andueza, Gonzalo*
Director *Gómez-Cornejo, Barrena, Julen*
Departamento Ingeniería Eléctrica
Curso académico *2021-2022*

Bilbao, 18, septiembre, 2022

Resumen

El presente trabajo se centra en el desarrollo de un software que se emplea para diseñar sistemas de autoconsumo fotovoltaico.

Existen diferentes metodologías para el desarrollo de estos sistemas y la aplicada se basa en la utilización de los paquetes Oemof basados en el lenguaje de programación Python, que resultan interesantes porque son gratis y de libre acceso. Se han empleado funciones de estos paquetes para determinar tanto la demanda como la generación de los edificios instalados.

Su desarrollo ha permitido obtener un programa con el que el usuario puede diseñar su propio sistema de autoconsumo, indicar los valores que quiere dar a las diferentes variables que lo forman y obtener los correspondientes resultados de su simulación.

Los estudios realizados indican que en determinadas ocasiones es conveniente la introducción de sistemas de almacenamiento para optimizar el funcionamiento y rentabilizar los flujos de energía que se dan. También permiten observar si los sistemas de generación son suficientes para cubrir la demanda en cada caso y si la interacción con la red es muy alta.

Abstract

The present work focuses on the development of a software that is used to design photovoltaic self-consumption systems.

There are different methodologies for the development of these systems and the applied one is based on the use of Oemof packages based on the Python programming language, which are interesting because they are free and freely accessible. Functions of these packages have been used to determine both the demand and the generation of the installed buildings.

Its development has made it possible to obtain a program with which the user can design their own self-consumption system, indicate the values they want to give to the different variables that make it up, and obtain the corresponding simulation results.

The studies carried out indicate that on certain occasions it is advisable to introduce storage systems to optimize operation and make the energy flows that occur profitable. They also allow observing if the generation systems are sufficient to cover the demand in each case and if the interaction with the network is very high.

Laburpena

Lan honen ardatza autokontsumoko sistema fotovoltaikoak diseinatzeko erabiltzen den softwarea garatzea da.

Sistema horiek garatzeko hainbat metodologia daude, eta aplikatua Python programazio-lengoaian oinarritutako Oemof paketeen erabileran oinarritzen da; horiek interesgarriak dira, doakoak eta sarbide librekoak direlako. Pakete horien funtzioak erabili ditu instalatutako eraikinen eskaria eta sorkuntza zehazteko.

Haren garapenak aukera eman du erabiltzaileak bere autokontsumo-sistema diseinatzeko programa bat lortzeko, hura osatzen duten aldagaiei eman nahi dizkien balioak adierazteko eta simulazioaren emaitzak lortzeko.

Egindako azterketen arabera, batzuetan komenigarria da biltegiratze-sistemak sartzea, funtzionamendua optimizatzeko eta gertatzen diren energia-fluxuak errentagarri bihurtzeko. Halaber, aukera ematen dute ikusteko sortze-sistemak nahikoak diren kasu bakoitzeko eskaria betetzeko, eta sarearekiko interakzioa oso handia den.

ÍNDICE

LISTA DE TABLAS, FIGURAS Y ACRÓNIMOS

Lista de tablas

Lista de figuras

Lista de acrónimos

1.INTRODUCCIÓN

En este TFM se ha programado un software para la generación de sistemas de autoconsumo fotovoltaico y se han analizado las diferentes situaciones que ofrece para comprobar sus funciones prácticas.

La memoria se ha iniciado con una descripción del contexto para poder ubicar este trabajo. Seguidamente, se han explicado los objetivos del proyecto y el alcance, es decir, lo que se busca conseguir y los diferentes requisitos o características para llegar a cumplir esos objetivos.

A continuación, se han determinado los beneficios que se obtienen para comprobar si todo lo recogido es factible. Se han analizado las diferentes condiciones favorables, tanto económicas como sociales, medioambientales y técnicas.

Después, se ha introducido un apartado en el que se explican algunos conceptos asociados al autoconsumo fotovoltaico para poder comprender a posterior las diferentes configuraciones que existen.

En el siguiente apartado, se ha realizado un análisis de los softwares existentes para realizar funciones similares a las buscadas, para conocer lo que ofrecen y determinar aspectos positivos y negativos de ellos y aplicarlos a la hora de definir el programa.

El siguiente paso ha consistido en analizar los paquetes del programa Oemof que se han empleado, entre los que se encuentran oemof.solph, Demandlib y Feedinlib. Se han creado las funciones necesarias de cada uno de los paquetes. Esto facilita lo siguiente, que corresponde con el diseño del programa final.

Una vez diseñado, se ha realizado un estudio de los resultados y se han comparado diferentes situaciones, haciendo una valoración de los resultados de cada una de ellas.

Para terminar, se han analizado los gastos de ejecución y se han realizado las conclusiones observadas al finalizar el trabajo.

2.CONTEXTO

En la actualidad, uno de los mayores problemas mundiales es la crisis energética. Esta se debe a que los recursos naturales que se emplean para producir energía están disminuyendo, ya que sus existencias en el planeta no son infinitas y se están agotando progresivamente, mientras que la demanda está aumentando.

Además, el empleo de estos recursos naturales también supone un problema en lo respectivo a la contaminación global. Los procesos de obtención y de explotación de estos recursos suponen un grave perjuicio para la atmósfera terrestre por los gases contaminantes que emiten.

Debido a ello, la dirección que están tomando los órganos gubernamentales y las diferentes administraciones es la de sustituir parte de estos recursos naturales por recursos renovables, ya que estos son fuentes limpias y no se agotan. No producen gases de efecto invernadero, aunque alguno de su procesos sí contaminan, pero su efecto es menor.

Entre estas fuentes de energía se encuentran algunas como la energía solar, la eólica, la bioenergía o la energía geotérmica. Su empleo reduce la dependencia energética que se tiene de los combustibles fósiles, los cuáles además tienen la tendencia de aumentar su coste, mientras que con las renovables sucede el caso opuesto.

La Unión Europea es uno de los órganos que quiere ser pionero en este proceso. Se ha comprometido a reducir las emisiones de CO_2 en un 40% para 2030 [1]. Sus principales objetivos son priorizar la eficiencia energética, ser el líder mundial en lo referido a las energías renovables y poder realizar una oferta justa a los consumidores. Para ello, pretende emplear 177.000 millones de euros al año de inversión pública y privada a partir del 2021 [1].

Aquí es donde surge un nuevo concepto para tratar de aplicar todo lo mencionado, las comunidades energéticas. Se trata de organizaciones donde sus componentes planifican e implementan

medidas para producir, consumir y comercializar energías renovables. Estas comunidades emplean todos los recursos con los que cuentan con el objetivo de ser más autónomos y reducir su dependencia de las compañías eléctricas. La participación de la ciudadanía hace que se refuerce su posición en el sistema y garantiza su acceso a la energía [2]. Al ser capaces de producir energía y venderla, además de consumirla, obtienen un notable beneficio económico y pueden emplear herramientas de gestión de los precios energéticos para reducir los costes.

La actividad principal que se lleva a cabo en estas comunidades es el autoconsumo eléctrico. Este proceso implica que las personas que participan en él, los autoconsumidores, sean capaces de generar energía, consumirla ellos mismos, almacenarla y vender la parte sobrante a la red. Estas actividades pueden realizarse entre personas de un mismo edificio o incluso entre diferentes edificios, lo que se denomina autoconsumo colectivo, que es el tipo conveniente para la creación de una comunidad energética [3].

Otras de las actividades que desempeñan estos sistemas son los servicios de agregación, que se encargan de asegurar el mantenimiento de la frecuencia y calidad del suministro eléctrico, y

los servicios de recarga energética, como los puntos de recarga para los vehículos eléctricos, y servicios de aplicaciones inteligentes.

En España existían normativas muy restrictivas para el desarrollo de estos sistemas, como por ejemplo la imposibilidad de obtener compensación para los autoconsumidores por los excedentes de energía, por lo que se está trabajando en modificar esas restricciones y se están implementando medidas que favorecen su creación. Uno de los primeros pasos se realizó en el Real Decreto Ley 15/2018, de 5 de octubre de 2018 [4] , donde se recogen algunas de estas condiciones, como lo son la simplificación de los trámites administrativos, la introducción de mecanismos de compensación o la eliminación del denominado "impuesto al sol", que suponía un coste extra para el autoconsumo. En los posteriores años se han implementado programas de ayudas y subvenciones que facilitan también la decisión de optar por estos sistemas.

Dentro de toda esta situación, para tratar de optimizar y de obtener los mejores resultados posibles, resulta conveniente el diseño de softwares que permitan configurar estos sistemas y faciliten su desarrollo. Por ello en este trabajo se trabaja en esta dirección para tratar de obtener mejores soluciones para el diseño y la optimización de los sistemas de autoconsumo mediante el empleo de softwares.

3.OBJETIVOS Y ALCANCE

En este apartado se han explicado los objetivos y el alcance del trabajo.

3.1. Objetivos

El objetivo principal ha consistido en el desarrollo de un programa que permita la modelización de un sistema de autoconsumo fotovoltaico de forma que pueda realizarse su optimización y obtención de resultados.

A partir de estos resultados se ha tratado de analizar el diseño y obtener conclusiones a cerca de su funcionamiento y posibles mejoras que se puedan introducir. Se ha buscado también analizar las diferentes situaciones que se dan en estas prácticas al realizar su conexión a la red o no.

También se ha pretendido presentar las diferentes condiciones que implica la instalación de sistemas de almacenamiento mediante baterías, valorando en cada situación las capacidades requeridas o la posible ineficacia de su instalación.

3.2. Alcance

Para poder cumplir los objetivos especificados se ha empleado el programa Python, con una de sus evoluciones más recientes, los paquetes de Oemof, que se han empleado para modelar y analizar sistemas energéticos.

Para poder analizar las diferentes condiciones especificadas, se ha creado un programa con un formato de menú, que permite la introducción en él de los elementos que el usuario especifique, permitiendo analizar diferentes situaciones.

Se ha realizado por tanto un programa en el que el usuario es un sujeto activo y debe indicar variables a la hora de generar el modelo, es decir, el usuario tiene la posibilidad de crear su propio sistema. Se ofrece en él la posibilidad también de seleccionar si se desea introducir o no baterías, y se puede indicar sus capacidades energéticas.

4. BENEFICIOS DEL PROYECTO

A continuación, se muestran los diferentes beneficios que se han obtenido con el desarrollo de este trabajo.

4.1. Beneficios económicos

Una de las principales ventajas de este proyecto es que todos los softwares y programas que se han utilizado son de libre uso y gratuitos. Como se ha mostrado en el apartado 6, la mayoría de los softwares existentes tienen un precio elevado, teniendo que pagar una cuota mensual o anual para su utilización. Por lo tanto, la realización de este procedimiento mediante Python y los paquetes de Oemof supone un gran ahorro al conseguir los mismos objetivos.

En cuanto a la utilización de este programa por parte de una comunidad, esto supone un importante beneficio económico para ella, ya que se puede emplear las funcionalidades que se ofrecen para realizar cálculo de cada situación y analizar la situación óptima. Su correcta implementación implica una reducción de costes del suministro energético y una limitación de la dependencia energética, teniendo mayor libertad respecto a los precios de las empresas eléctricas.

Además, se fomenta la creación de empleo y el desarrollo de negocios locales. Según datos de la Unión europea, el enfoque hacia el autoconsumo puede generar un aumento del 1% del PIB y crear 900.000 empleos nuevos [1].

4.2. Beneficios sociales y medioambientales

La aplicación del programa desarrollado conlleva una mejora de las condiciones de vida en zonas rurales y urbanas. Se obtiene una mayor cohesión social entre todos los integrantes que forman las comunidades [2]. También se aumenta el valor de lo local, promoviendo que se realicen inversiones en esas comunidades, limitando el poder de las grandes empresas eléctricas.

En cuanto a las condiciones medioambientales, por una parte, supone una reducción de la energía desaprovechada. Al estar todos los elementos cooperando, si uno genera más de lo necesario puede aportarlo a otro de los componentes y viceversa, evitando la pérdida de esa energía. El programa permite analizar cada situación y diseñar el modelo adecuado para un correcto balance de la energía generada y la consumida.

A su vez, la tecnología empleada es la fotovoltaica, que es una de las energías renovables, denominadas energías limpias, que reducen las emisiones y el impacto negativo en el clima, siendo esto de gran valor para el futuro del planeta.

4.3. Beneficios técnicos

En lo relacionado a la parte técnica, el código generado permite crear un sistema con el número de edificios que se desee, no se limita a uno o dos edificios. Se ha dado la posibilidad de definir todas las características y parámetros, como son las fechas del estudio, la orientación de los paneles, la ubicación o el número de módulos fotovoltaicos.

Permite diseñar un sistema de almacenamiento mediante baterías adaptado a cada comunidad. Se pueden instalar baterías en unos edificios y en otros no, y a su vez, es posible disponer de una batería para todo el sistema. Esto da una gran capacidad y variedad de análisis. Se puede también cambiar las capacidades de dichas baterías y comprobar qué tipo son necesarias en función de la energía generada y consumida.

5. DESCRIPCIÓN DEL AUTOCONSUMO FOTOVOLTAICO

En esta sección se realiza una descripción del concepto de autoconsumo fotovoltaico para poder conocer mejor a posterior su funcionamiento y los diferentes factores que afectan a su desarrollo.

Existen diferentes tipos de autoconsumo como el autoconsumo solar térmico, el autoconsumo minieólico o el de biomasa, pero este trabajo se ha centrado en el caso de autoconsumo fotovoltaico, que es el más empleado y el más desarrollado en la actualidad.

5.1.Definición

Se entiende por autoconsumo fotovoltaico a la producción de energía eléctrica a través de paneles solares para satisfacer las necesidades energéticas de una instalación [5]. Las instalaciones requeridas, además de los paneles, tienen otros elementos como inversores, conectores, cables y en algunos casos baterías.

Se trata de uno de los sistemas en los que más se está avanzando en los últimos años y se está impulsando su introducción en el sistema energético [6] [5], ya que aporta beneficios tanto económicos como medioambientales. La disminución de los precios de los elementos que componen la instalación, a la vez que la mayor facilidad otorgada por las administraciones públicas en lo referido a los trámites administrativos ha supuesto una mayor accesibilidad a estos sistemas.

En los siguientes apartados se definen las características de estos sistemas, las cuales conviene analizar antes de diseñar un sistema para conocer las que mejor se adaptan a dicho sistema.

5.2. Modalidades de autoconsumo

Las modalidades de autoconsumo hacen referencia al concepto de la energía excedente, es decir, si además de aprovechar la energía obtenida para su consumo, se puede hacer otros usos si existe más de la necesitada.

Los dos tipos existentes son los siguientes:

5.4.1. Modalidad de suministro sin excedentes

Según lo definido por el Real Decreto-ley 15/2018 [4] , publicado en el BOE el 6 de octubre de 2018, se habla de suministro sin excedentes "cuando los dispositivos físicos instalados impidan la inyección alguna de energía excedentaria a la red de transporte o distribución." En este caso solo existe un tipo de sujeto que se trata del consumidor. La energía generada se emplea únicamente para su consumo.

5.4.2. Modalidad de suministro con excedentes

Según lo definido por el Real Decreto-ley 15/2018 mencionado previamente, se habla de suministro con excedentes "cuando las instalaciones de generación puedan, además de suministrar energía para autoconsumo, inyectar energía excedentaria en las redes de transporte y distribución". En este caso existen dos tipos de sujetos, el consumidor y el productor. Aquí es donde surge un nuevo término, el prosumidor, que corresponde al sujeto que ejerce ambas labores, produce energía y a su vez la consume.

A su vez, esta categoría se divide en dos subcategorías, que son las siguientes:

- Modalidad con excedentes acogida a compensación: Esta clase recoge aquellos casos de suministro en los que tanto el consumidor como el productor decidan acogerse a un mecanismo de compensación de dicho excedente de energía.
  Para poder realizarlo existen una serie de condiciones reguladas en el Real Decreto 244/2019 [7], entre las que se encuentran la necesidad de que la fuente de energía sea de origen renovable, la potencia total de las instalaciones no puede ser superior a 100 kW y una serie de condiciones contractuales entre consumidor y productor.
- Modalidad con excedentes no acogida a compensación: En esta modalidad se agrupan los diferentes sistemas que no cumplan los requisitos especificados para poder acogerse a compensación o aquellos sistemas que opten por no acogerse a la modalidad de compensación.

5.2. Tipos de autoconsumo

Existen dos tipos de autoconsumo, el autoconsumo individual y el autoconsumo colectivo, cada uno de ellos empleado para un tipo de sistema diferente.

5.2.1. Autoconsumo individual

Este tipo es aquel en el que hay un único consumidor que hace uso de la instalación generadora. Resulta útil para casos como casas alejadas del núcleo urbano y que no tengan suficientes conexiones cerca.

En este trabajo no se ha desarrollado este caso ya que se ha pretendido desarrollar un sistema en el que se aproveche las ventajas de tener diferentes consumidores y obtener los beneficios de trabajar conjuntamente por un beneficio común.

5.2.2. Autoconsumo colectivo

El autoconsumo colectivo es aquel que está compuesto por varios consumidores que se unen y que aprovechan conjuntamente las diferentes instalaciones generadoras. Es el caso que se desarrolla para el sistema trabajado.

Como viene definido en el Real Decreto 244/2019 [7], publicado en el BOE el 6 de abril de 2019,"se dice que un sujeto consumidor participa en un autoconsumo colectivo cuando pertenece a un grupo de varios consumidores que se alimentan, de forma acordada, de energía eléctrica que proveniente de instalaciones de producción próximas a las de consumo y asociadas a los mismos."

El autoconsumo colectivo permite acogerse a cualquiera de las modalidades de autoconsumo, cuando este se realice entre instalaciones próximas de red interior. Pero es importante tener en cuenta que todos los consumidores que estén unidos a la misma instalación de generación tienen que disponer de la misma modalidad y tener firmado un acuerdo que recoja los criterios con los que se reparte la energía.

5.3. Tipos de conexión de los sistemas

Los diferentes tipos de conexión de los sistemas de autoconsumo son los siguientes:

5.3.1. Instalación aislada

Como aparece publicado en el Real Decreto 244/2019 [7], publicado en el BOE el 6 de abril de 2019, una instalación aislada es "aquella en la que no existe en ningún momento capacidad física de conexión eléctrica con la red de transporte o distribución ni directa ni indirectamente a través de una instalación propia o ajena." Este tipo de instalaciones no están conectadas a la red eléctrica, por lo que su único objetivo consiste en dar energía al lugar o al edificio en el que están instaladas.

5.3.2. Instalación conectada a la red

Según el Real Decreto 244/2019, publicado en el BOE el 6 de abril [4] , una instalación conectada a la red es "aquella instalación de generación conectada en el interior de una red de un consumidor, que comparte infraestructuras de conexión a la red con un consumidor o que esté unida a este a través de una línea directa y que tenga o pueda tener, en algún momento, conexión eléctrica con la red de transporte o distribución." También se recoge en este Decreto que se considera instalación conectada a red a aquella que esté conectada directamente a las redes de transporte y distribución.

La ventaja con la que cuentan estos sistemas es que permiten excluir los módulos de almacenamiento, ya que en caso de tener un excedente de energía se puede vender a la red, mientras que si existe una falta de energía se puede comprar energía a la red y obtener la necesaria para el correcto desarrollo del sistema. Esto supone que permitan reducir el coste de la facturación mensual del consumidor [6] Sin embargo, requieren de la introducción de un equipo especial para poder adaptar la energía generada con la energía de la red eléctrica.

5.5. Almacenamiento mediante baterías

Uno de los principales problemas de la energía fotovoltaica es que no existe una generación constante, ya que se depende del factor de las horas de sol disponible. Estas horas son muy variables a lo largo del año debido a los diferentes fenómenos meteorológicos que tienen lugar. Además, las horas nocturnas también son un impedimento ya que no se cuenta con la energía solar.

Debido a ello, para garantizar que se dispone de energía eléctrica sin depender de estos factores, existe la posibilidad de instalar sistemas de almacenamiento mediante baterías, para acumular la energía en las horas que haya un excedente y poder emplearla en las horas que no se tiene generación.

Otra función que tienen estas baterías es la de ser cargadas en las horas que el precio de la electricidad sea más bajo, para luego ser descargadas cuando el precio sea más elevado, teniendo como consecuencia un beneficio económico.

Según un estudio denominado "Análisis del estado actual del almacenamiento detrás del contador en España"[8] , realizado por el Instituto para la Diversificación y Ahorro de la Energía (IDAE), las tecnologías más empleadas actualmente son las baterías electroquímicas, en especial las de tipo Li-ion. Estas baterías tienen densidades de potencia y energía elevada, además de un ciclo de vida largo, permitiendo entre 1.500 y 10.000 ciclos, y una excelente eficiencia, entre el 86 y el 96 % [9] Sin embargo, cuenta con algunas desventajas. Su coste es elevado respecto a otras tecnologías como la de plomo ácido. Otros inconvenientes son el riesgo de deflagración y la degradación.

Las características que hay que tener en cuenta a la hora de seleccionar un sistema de baterías son las siguientes:

- Capacidad de potencia (MW): Potencia máxima de descarga.
- Capacidad energética (MWh): Cantidad máxima de energía que puede almacenar.
- Densidad de potencia (W/l): Potencia instantánea por unidad de volumen.
- Densidad de energía (Wh/l): Energía que puede almacenar por unidad de volumen.
- Tiempo de descarga (h): Tiempo para descargar su capacidad de potencia.
- Vida útil: Número de ciclos realizables.
- Autodescarga (%):Carga que pierde por reacciones químicas sin utilización.
- Estado de carga (%): Nivel de carga disponible.
- Eficiencia (%): Ratio entre la carga y la descarga de la batería.
- Coste (€/kWh): Precio por unidad de energía almacenada.
- Madurez: Fase de evolución de la tecnología.

En el estudio mencionado se hace referencia también a los diferentes retos que tienen los sistemas de almacenamiento mediante baterías. Se recogen retos regulatorios, tecnológicos y económicos. Esto se debe a que actualmente es una tecnología que requiere de una evolución. Se necesitan dispositivos con mayor eficiencia y vida útil. Además, su coste actual es elevado, por

lo que una disminución del precio resultaría efectiva para dar una mayor accesibilidad a estos componentes.

6. ANÁLISIS DEL ESTADO DEL ARTE

En el presente apartado se describen las herramientas de análisis más destacadas para la realización de una solución para un sistema de autoconsumo. El objetivo es mostrar los diferentes softwares que se pueden emplear para diseñar un proyecto de autoconsumo, para observar las funcionalidades que ofrecen estos métodos, y a posterior realizar la solución con el método que se ha desarrollado en este trabajo. Las diferentes opciones explicadas se centran en el autoconsumo fotovoltaico, que es la solución que se ha empleado.

A continuación, se separan los softwares en dos grupos. Los primeros se tratan de aquellos destinados exclusivamente a cálculos de fotovoltaica y los segundos son aquellos destinados a la modelización de sistemas.

6.1. Softwares para cálculos fotovoltaicos

En este apartado se presentan aquellos softwares cuya función es diseñar sistemas fotovoltaicos y realizar los correspondientes cálculos. En la realización de este trabajo se ha optado por no emplearlos, ya que la mayoría tienen un precio elevado y uno de los objetivos principales es diseñar un programa sin gastos de software y de libre acceso. A pesar de ello, el estudio de las funciones que ofrecen es beneficioso para el proyecto ya que permite valorar su introducción en el programa.

6.1.1. Homer

Se trata de un software para la planificación de microrredes, que ofrece la posibilidad de realizar el análisis técnico y económico de optimización y sensibilidad de sistemas híbridos. Cuenta con 3 productos: Homer Grid, Homer Front y Homer Pro. [10]

Homer Grid se emplea para generación distribuida. Reduce los costos de energía y aumenta la resiliencia de las instalaciones conectadas a la red y las estaciones de carga de vehículos eléctricos. Homer Front se ocupa de los sistemas híbridos a gran escala, maximizando su rendimiento de almacenamiento. Por último, el Homer Pro se encarga de las microrredes independientes, por lo que resulta el más interesante para el actual trabajo, ya que explora las soluciones de menor costo para sistemas de energía remotos, microrredes y servicios públicos aislados.

El proceso que desarrolla comienza por intentar simular un sistema viable para todas las combinaciones posibles del equipo que se desee considerar, . A continuación, se procede a la optimización de los sistemas simulados, que se ordenan y se filtran según los criterios que se definan, para poder observar los mejores ajustes posibles. Principalmente se trata de un modelo de optimización económica, aunque también se puede emplear para minimizar el uso de combustible. Por último, existe una paso opcional, el análisis de sensibilidad, que permite modelar el impacto de la variables que están fuera de control, como la velocidad del viento o los precios de combustible, y observar cómo cambia el sistema óptimo con estas variaciones.

Para el empleo de este software para el diseño hay que seguir 5 pasos. Para comenzar, se debe generar un carpeta Homer y definir una serie de datos, que se realiza como se observa en la Figura 6.1. Después, se tiene que introducir el perfil de carga. Una vez definido, se procede a crear el sistema, añadiendo componentes como pueden ser generadores, placas fotovoltaicas, baterías, conversores...etc.

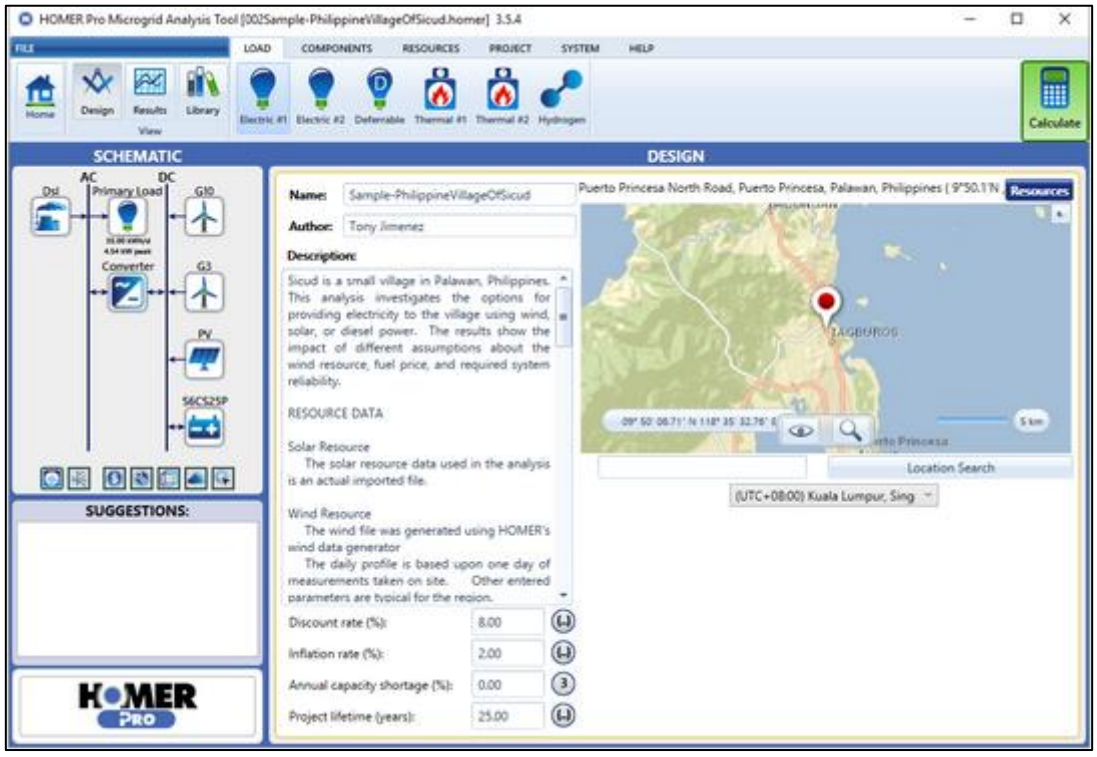

Figura 6.1. *Programa Homer [11]*

Para terminar, se definen las fuentes existentes, es decir, se tienen que aportar los datos de entrada de energía solar, eólica, temperaturas, combustibles, etc. Con todo definido, se realiza la simulación y la optimización.

Su precio es de 125 dólares por mes en su nivel base, 249 dólares al mes el nivel profesional y 379 dólares el nivel experto, añadiéndose ciertas funcionalidades al aumentar el nivel.

6.1.2. PV*SOL

Este software permite diseñar sistemas fotovoltaicos de autoconsumo con o sin excedentes, dando la posibilidad de introducir diferentes tarifas en el caso de existir excedentes. Ofrece la posibilidad de introducir sistemas de baterías, permitiendo definir diferentes estrategias de carga [12] Ofrece la posibilidad de diseñar desde un tejado con pocos módulos hasta parques solares con hasta 100.000 módulos, por lo que resulta interesante para el autoconsumo por esa posibilidad de representar sistemas pequeños.

Además del diseño simple del sistema, PV*SOL ofrece la opción de trabajar con el programa integrado de medición de fotografías PhotoPlan. Usando una foto y una dimensión de referencia, el techo respectivo con el sistema fotovoltaico potencial se puede mostrar de una manera

fotorrealista. También está disponible una herramienta de planificación gráfica 2D . Cuenta con una base de datos en línea en la que se incluyen 21.000 módulos fotovoltaicos, 5.100 inversores, 1900 sistemas de baterías y muchos otros productos, dando grandes posibilidades a la hora de la creación del sistema. Las tarifas eléctricas actuales están disponibles en su base de datos para el cálculo de la eficiencia económica. Dispone de los datos climáticos globales disponibles, proporcionando los datos más recientes del DWD (Deutscher Wetterdienst) para Alemania y más de 8000 ubicaciones adicionales. Las ubicaciones no incluidas se interpolan utilizando datos satelitales y estaciones de medición terrestres vecinas.

Con todo esto, permite crear informes completos de proyectos. El resumen detallado de los resultados contiene representaciones de los resultados de la simulación, los resultados de rentabilidad y un balance de energía tabular detallado con todas las ganancias y pérdidas que ocurren. El precio de la licencia que incluye 6 meses de mantenimiento de software es de 895 euros.

6.1.2.1.PV*SOL premium

El programa PV*SOL cuenta con una licencia premium, que destaca por su visualización 3D. Permite visualizar todos los tipos comunes de sistemas en 3D, ya sea integrados en el techo o montados en el techo, en techos pequeños en ángulo, grandes naves industriales o espacios abiertos.

Dispone de varias funcionalidades extra respecto a la licencia normal. Permite realizar el análisis del sombreado en función de objetos 3D, realizando una representación tridimensional de los elementos circundantes para determinar la reducción de rendimiento debido al sombreado. Posibilita también la creación de edificios y objetos de forma rápida y sencilla utilizando planos de planta y capturas de pantalla de mapas. Algunos de los detalles extra con los que cuenta son la importación de modelos 3D de diferentes formatos de archivo, la asignación de módulos en vista 3D, la interconexión polimórfica en combinación con optimizadores y la optimización de la interconexión de módulos.

El precio de la licencia premium es de 1295 euros que incluye 6 meses de mantenimiento de software.

6.1.3. PVGIS

Se trata de otro software que aporta datos respecto a la irradiación solar y el rendimiento energético. En este caso, no requiere de ningún tipo de instalación, ya que se trata de una herramienta online gratuita, por lo que esto supone una ventaja. Está desarrollada por la Unión Europea[13]

El PVGIS permite seleccionar un punto concreto para buscar datos útiles relativos a las instalaciones fotovoltaicas. Requiere de la introducción de determinados datos, como el punto geográfico exacto en el que se realiza la instalación, la potencia, y diferentes datos de los módulos como la orientación o la inclinación. Con ellos se obtiene la producción energética aproximada. Tiene varios inconvenientes para el diseño. La potencia que requiere introducir es la potencia

pico, por lo que hay que hacer cálculos antes. Además, estima las pérdidas por defecto y no deja introducir tipos de módulos fotovoltaicos.

Se concluye que es un software diferente a los anteriores, ya que cuenta con ventajas en cuanto a su facilidad de uso y su precio, pero cuenta con varias limitaciones, que lo hacen más que un software para calcular instalaciones fotovoltaicas, un software de evaluación de la generación eléctrica de dichos sistemas. Por ello no se ha empleado para el trabajo ya que sus funciones son insuficientes para su desarrollo.

6.1.4. PVSYST

PVSYST es una de las herramientas más empleadas en el mundo para dimensionar y estudiar instalaciones fotovoltaicas [14] Permite diseñar tanto plantas grandes como sistemas de autoconsumo con o sin conexión a la red.

Incluye el Meteonorm, con el que se obtienen los datos meteorológicos, que cuenta con un programa de interpolación para definir cualquier punto del planeta. Dispone de una herramienta para dimensionar el sistema. Corresponde a una herramienta visual, se muestran una serie de gráficos y hay que comprobar si el inversor y el número de módulos seleccionados son los adecuados para el proyecto. Al igual que el PV*SOL premium, dispone de una representación 3D para evaluar el sombreado, permitiendo diferentes funciones como manipulación y creación de objetos, simulación del punto de vista del sol o herramientas de identificación de orientación y validación de escena.

Mediante este software, el diseño del sistema requiere de 3 pasos. La selección de la potencia deseada o del área disponible, la elección del módulo fotovoltaico de una base de datos interna y la selección de un inversor de una base de datos interna. Estos pasos se observan en la Figura 6.2, donde también se aprecia que hay que indicar el número de módulos.

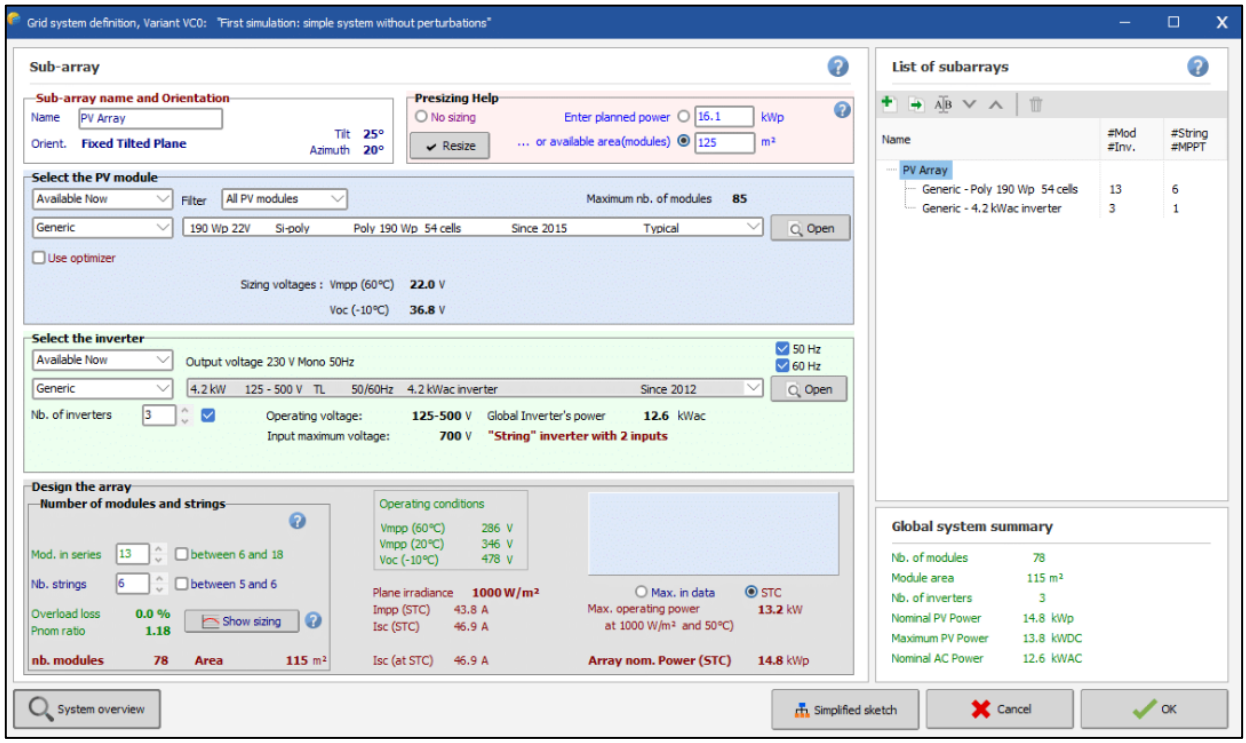

Figura 6.2. *Definición proyecto mediante PVSYST [14]*

Una vez realizado el diseño, la simulación calcula la distribución de energías a lo largo del año, y los principales resultados que se obtienen son:

- Producción total de energía (MW/h), esencial para evaluar la rentabilidad
- Índice de rendimiento, para evaluar la calidad del sistema
- Energía específica, que es un indicador de producción basado en la irradiación disponible

Con estos resultados, el software permite realizar una evaluación económica, obteniendo una evaluación de costes, una estrategia de precios, un análisis financiero avanzado y un análisis de rentabilidad. El precio del PVSYST es de 609 euros por la suscripción anual.

6.1.5. PV-DesignPro

Este software es una herramienta autónoma de análisis y diseño de sistemas fotovoltaicos. El periodo de simulación que emplea es de una hora, por lo que emplea archivos climáticos por hora. Cuenta con 3 versiones del programa. PV-DesignPro-S para sistemas independientes con almacenamiento de batería, PV-DesignPro-G" para sistemas conectados a la red sin almacenamiento de batería, y PV-DesignPro-" para sistemas de bombeo de agua [15]

6.2. Softwares para modelización de sistemas

En esta sección se muestran los softwares destinados a modelizar sistemas. Sus aplicaciones no se centran exclusivamente en el cálculo de instalaciones fotovoltaicas, pero mediante el

empleo de sus herramientas es posible la modelización de un sistema que realice las mismas funciones que los softwares anteriores.

6.2.1. MATLAB

MATLAB es una plataforma de programación y cálculo numérico empleada para analizar datos, desarrollar algoritmos y crear modelos. Permite realizar funciones como analizar datos, desarrollar algoritmos, crear de apps, visualizar gráficas y otras muchas funciones.

En este caso, esta herramienta no está destinada al diseño de sistemas fotovoltaicos, como si sucedía con el resto de los softwares explicados, pero sí que ofrece la posibilidad de realizar un sistema como el buscado mediante el empleo de sus funciones.

No se ha optado por este software, a pesar de que cumple las funciones buscadas, ya que la solución que se emplea a posterior es similar a este software, pero cuenta con paquetes destinados a la modelización de sistemas energéticos, por lo que facilita el diseño del programa.

6.2.2. OEMOF

OEMOF se trata de un marco abierto para el desarrollo de modelos de sistemas energéticos y la realización del análisis de dichos sistemas. Es una colección de paquetes gratuitos y de libre acceso que se usan mediante el programa Python.

Para la realización del trabajo se ha optado por emplear este software, ya que se trata de un marco en desarrollo que resulta muy interesante para diseñar este tipo de sistemas y ofrece las funcionalidades requeridas en este estudio. En el siguiente apartado se realiza una descripción más extensa de estos paquetes y de las diferentes bibliotecas con las que cuenta.

7. MÉTODO DE TRABAJO

Una vez analizadas las diferentes posibilidades existentes y las funciones que ofrecen, el programa que se ha empleado para la realización del presente trabajo es el Python, haciendo uso de unos paquetes específicos denominados Oemof (Open Energy Modeling Framework).

Python es un lenguaje de programación que se emplea para múltiples funciones. Sus estructuras de datos integradas de alto nivel y su escritura dinámica lo hacen interesante para el desarrollo de aplicaciones y para su uso como lenguaje de secuencias de comandos. El programa y sus bibliotecas principales están disponibles para todas las plataformas principales de forma gratuita. La sintaxis de este lenguaje es simple y fácil de aprender. La ausencia de un paso de compilación hace que el ciclo de edición, prueba y depuración sea muy rápido. Una de sus funcionalidades más interesantes es que admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código. Es aquí donde entran paquetes como el mencionado Oemof, que se ha trabajado con ellos

El primer paso para obtener un entorno en el que poder trabajar mediante Oemof ha consistido en instalar el programa Python. Es importante destacar que se debe comprobar si el ordenador tiene un sistema operativo de 32 o 64 bits, ya que en base a eso se debe descargar Python para una opción u otra.

Lo siguiente ha consistido en instalar los paquetes de Oemof, que se realiza en la ventana de comandos de Python introduciendo el siguiente comando:

- **pip install oemof**

Una vez instalados, ya se dispone del programa para poder comenzar a trabajar. Al instalar los paquetes de Python, se ha instalado también el programa Spyder que es el entorno en el que se ha trabajado ya que resulta más cómodo y fácil para trabajar.

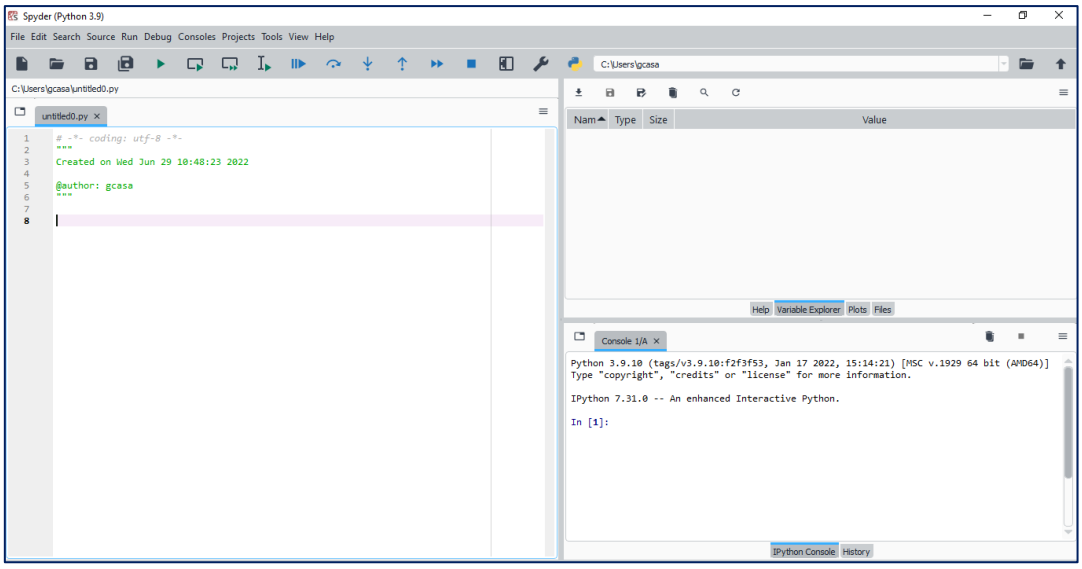

Figura 7.1. *Programa Spyder*

Como se observa en la Figura 7.1, se tienen 3 ventanas. En la parte izquierda se encuentra la ventana donde se escribe el código. En la parte derecha inferior se encuentra la consola, donde aparecen los resultados y donde se introducen las sentencias requeridas por el programa. También en ese punto se encuentra una ventana en la que se ve la historia de las ejecuciones realizadas. Y en la parte superior derecha está la ventana donde se ven las variables, los gráficos, así como la ventana de ayuda, que es muy útil ya que Oemof resulta difícil de comprender de inicio y es recomendable utilizar la ayuda para entenderlo mejor.

Oemof consiste en una colección de bibliotecas de Python, por lo que se puede considerar que se trata de una aplicación de Python para realizar modelos en el marco energético, lo cual lo hace una aplicación muy interesante ya que Python es un programa accesible para todo el mundo y al igual que oemof es gratuito. Su desarrollo comenzó en el año 2014, cuando se fundó el grupo de trabajo para su desarrollo, formado por investigadores del Centro de Sistemas de Energía Sostenible de Flensburg (ZNES) junto con el Instituto Reiner Lemoine (RLI) en Berlín.

Debido a que se desarrolló en un contexto académico, los estándares científicos (transparencia, repetibilidad, reproducibilidad y escrutinio) eran importantes. Los modelos de sistemas de energía a menudo no tienen un código fuente de acceso público, faltan datos disponibles de forma gratuita y están mal documentados. Esto crea barreras para el progreso científico en el modelado y análisis de sistemas energéticos. Además, el código del modelo del sistema de energía a menudo no se puede reutilizar debido a problemas legales y prácticos. El desarrollo de oemof aborda estos problemas al ofrecer un marco gratuito, abierto y claramente documentado. El enfoque de código abierto permite un desarrollo colaborativo del marco que ofrece varias ventajas:

- Sinergias: mediante el desarrollo colaborativo, se pueden utilizar sinergias entre los institutos participantes.
- Depuración: a través de la entrada de un grupo más grande de usuarios y desarrolladores, los errores se identifican y corrigen en una etapa anterior.
- Avance: la aplicación basada en oemof se beneficia del mayor desarrollo del marco.

Oemof trata de abordar los desafíos actuales y futuros en el modelado de sistemas de energía teniendo las siguientes características:

- Flexible y genérico: crear aplicaciones y adaptar los componentes a su alcance y propósito.
- Intersectorial: incluir y vincular el sector de la calefacción, la energía y la movilidad.
- Multirregional: conectar de forma flexible varias regiones.
- Flexibilidad temporal: elegir la resolución temporal que mejor se adapte a su aplicación.
- Modular: elegir entre varios paquetes de Python con interfaces bien definidas para el modelado y la optimización.
- Abierto: ayuda para agregar nuevas funciones.
- Transparente: encontrar la información que se necesita en su documentación.
- Impulsado por la comunidad: participar en el proceso de desarrollo.

Todo esto se logra con las diferentes bibliotecas desarrolladas en el programa, cada una especializada para una determinada tarea, pero todas comparten principios comunes y son parcialmente interoperables. Actualmente, los proyectos con lanzamientos estables son los siguientes:

- oemof-solph: Es un generador de modelos para modelado y optimización de sistemas de energía .
- oemof-thermal: Se trata de herramientas para modelar componentes de energía térmica (bombas de calor de compresión, plantas termosolares, acumuladores térmicos y colectores solares térmicos) como extensión de oemof-solph.
- Cycle Detection in Time Series (CyDeTS): Representa un algoritmo para detectar ciclos en series temporales junto con su respectiva profundidad de ciclo (DoC) y duración.
- demandlib: Permite generar perfiles de carga para la demanda de electricidad y calor.
- feedinlib: Calcula series temporales de alimentación para fuentes de energía renovable fluctuantes a partir de datos meteorológicos.
- Thermal Engineering Systems in Python (TESPy) : Es un conjunto de herramientas de simulación para plantas de ingeniería térmica como centrales eléctricas, sistemas de calefacción urbana o bombas de calor.
- windpowerlib: Calcula la potencia de salida de aerogeneradores eólicos.

Existen a su vez paquetes que se encuentran en sus principios de desarrollo. Actualmente se está desarrollando la biblioteca DHNx, cuyo objetivo es desarrollar modelos de optimización y simulación de sistemas de calefacción urbana.

Para el desarrollo de este trabajo se ha empleado la combinación de 3 de estas funcionalidades, todas ellas aplicadas exclusivamente a la parte eléctrica. La primera empleada ha sido Demandlib, con la que se ha desarrollado una función para determinar la demanda eléctrica de un edificio para un determinado periodo de tiempo. A continuación, se ha utilizado el paquete Feedinlib para desarrollar las funciones que permiten calcular los datos meteorológicos de un lugar, y emplear éstos para calcular la generación de una instalación fotovoltaica en ese lugar. Por último, se ha usado oemof-solph con el que, utilizando los datos obtenidos de los otros dos paquetes, se ha modelado el sistema y se ha procedido a su resolución.

Para comprender mejor el funcionamiento de estos paquetes, se dispone del código fuente de ellos en GitHub, donde se obtienen diferentes carpetas como se aprecia en la Figura 7.2 [18] .

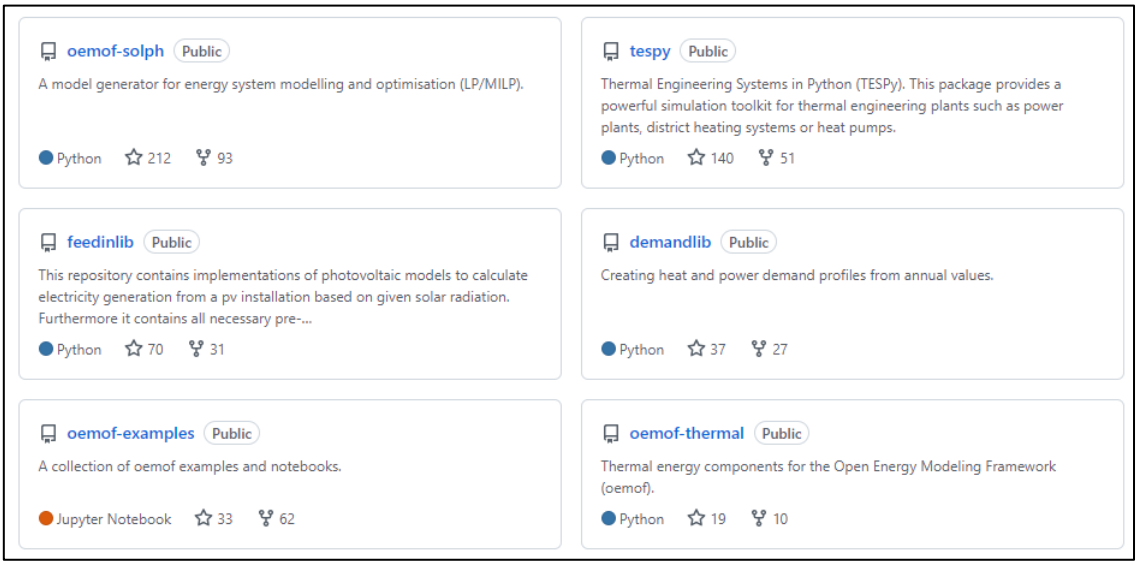

Figura 7.2. *Código fuente en GitHub [18]*

Esto supone una gran ayuda especialmente al empezar a trabajar con Oemof, ya que permite comprender mejor su funcionamiento. A su vez se dispone también de una carpeta de ejemplos, que permiten observar el desarrollo de un programa base y a partir de él desarrollar el programa que se desea. Se ha empleado especialmente uno de ellos para el trabajo, denominado *storageinvestment*, ya que se acerca a lo que se desea realizar, por lo que como se ha explicado tenerlo como base de trabajo resulta muy útil.

Durante el desarrollo del trabajo, tanto la carpeta Feedinlib como la Demandlib han sido muy importantes ya que en ellas se profundiza en estos módulos y resulta muy práctico para saber cómo emplear sus funciones.

7.1. Demandlib

El primero de los paquetes que se han empleado para el desarrollo de este trabajo es el módulo Demandlib. Su principal función es generar perfiles de carga para la demanda de electricidad y calor. Como se ha explicado previamente, en este caso el trabajo se ha centrado en la parte eléctrica, por lo que solo se ha empleado la función para generar perfiles de la demanda eléctrica. Estos paquetes son gratuitos y de libre instalación. Se obtienen mediante la ejecución en la ventana de comandos de la siguiente secuencia:

- **pip install demandlib**

Al ejecutarlo, se ha comprobado que existían funciones de las que no se disponía, es por ello por lo que resulta más interesante instalar la versión en desarrollo, con la cual se obtienen las últimas funcionalidades desarrolladas, que se consigue ejecutando lo siguiente:

- **pip install** https://github.com/oemof/feedinlib/archive/master.zip

La forma en la que se consiguen estos perfiles es escalando los perfiles BDEW (Bundesverband der Energie- und Wasserwirtschaft) a la demanda anual deseada. BDEW es la Asociación Federal de Gestión de la Energía y el Agua de Alemania, por lo que estos están basados en mediciones realizadas sobre el sector eléctrico alemán. Los diferentes tipos de perfiles que se obtienen dependiendo del sector son:

Tabla 7.1. *Tipos de perfiles de demanda eléctrica [19]*

Tipos	Descripción	Explicación
G0	Comercio general/negocios	Promedio de perfiles G1-G6
G1	Días laborables de 8 p.m. a 6 a.m.	Oficinas, colegios...
G2	Negocios con consumo predominante en horario nocturno	Clubes deportivos, restaurantes nocturnos...
G3	Negocios continuos	Cámaras frigoríficas, plantas de tratamiento de aguas residuales...
G4	Tiendas/peluquerías	
G5	Panaderías con horno	
G6	Operación en fin de semana	Cines...
G7	Estación transmisora de telefonía móvil	Perfiles de carga de banda continua
L0	Fincas	Promedio de perfiles L1-L2
L1	Fincas con ganadería lechera	
L2	Resto de fincas	
H0	Hogares	

En la Tabla 7.1 se han definido los diferentes tipos de perfiles eléctricos que se obtienen con estos paquetes. Cabe destacar que al trabajar con Python las letras que definen cada tipo deben redactarse mediante letras minúsculas para el correcto desarrollo del programa.

En el caso de este trabajo, se han empleado los 3 tipos que se han considerado más interesantes. El tipo H0 que corresponde a la demanda de los hogares, el G6 para los edificios con operación en fin de semana y el G0 para los que les corresponde una operación predominante entre semana.

Los paquetes de Demandlib están pensados para la obtención de un perfil anual, pero en la realización de este trabajo se han modificado una serie de parámetros para poder desarrollar estos perfiles en un periodo de tiempo cualquiera, pudiendo ser desde días hasta años, dando una mayor funcionalidad al programa. Para ello, se ha realizado una función llamada *diasHastaFecha*, cuyo código se adjunta en el anexo A, para determinar el número de días que conlleva la realización del estudio.

Con todo lo explicado previamente, se ha realizado una función denominada *Demanda*, cuya ejecución ha requerido la importación de los paquetes que se muestran en la figura 7.3:

```
import datetime
import numpy as np
from matplotlib import pyplot as plt
import demandlib.bdew as bdew
from funciones.DiasEntreFechas import diasHastaFecha
```

Figura 7.3. *Paquetes importados para la función Demanda*

Para la determinación del periodo mencionado previamente se han empleado tanto la función *diasHastaFecha* como la función *datetime*, que sirve para conseguir el formato de fecha a partir de los datos de día, mes y año.

```
def Demanda():
    #Para definir las vacaciones se toma como referencia el calendario
    #festivo del País Vasco
    holidays = {
        datetime.date(2022,1,1): "Año nuevo",
        datetime.date(2022,1,6): "Día de Reyes",
        datetime.date(2022,4,14): "Jueves Santo",
        datetime.date(2022,4,15): "Viernes Santo",
        datetime.date(2022,4,18): "Lunes de Pascua",
        datetime.date(2022,7, 25): "Santiago",
        datetime.date(2022,8,15): "Asunción de la Virgen",
        datetime.date(2022,9,6): "Día de Elcano",
        datetime.date(2022,10,12): "Fiesta Nacional de España",
        datetime.date(2022,11,1): "Todos los Santos",
        datetime.date(2022,12,6): "Día de la Constitución",
        datetime.date(2022,12,8): "Inmaculada",
        }
    Año1=int(input("Indique el año en el que se va a iniciar el estudio:"
                   " "))
    Año2=int(input("Indique el año en el que se va a finalizar el estudio:"
                   " "))
    Mes1=int(input("Indique el mes de inicio del estudio: "))
    Dia1=int(input("Indique el día de inicio del estudio: "))
    Mes2=int(input("Indique el mes de fin del estudio: "))
    Dia2=int(input("Indique el día de fin del estudio: "))
    dias=diasHastaFecha(Dia1,Mes1,Año1,Dia2,Mes2,Año2)
```

Figura 7.4. *Determinación del periodo de estudio de la demanda*

Se observa en la Figura 7.4 como también se han definido los días festivos del año 2022, que es el periodo en el que en el apartado 9 se ha realizado el ejemplo de desarrollo del programa. Se han tomado los días del calendario festivo del País Vasco, ya que se trata del lugar en el que se ha desarrollado el programa. Para el estudio de otro periodo se requeriría de la definición de los días festivos de ese periodo, aunque también existe la posibilidad de no definir los días vacacionales y ejecutar el programa sin esa información.

Para la definición del consumo anual de cada tipo se ha realizado lo siguiente:

```
demanda_por_sector = {
    "g0": 52.5*(dias/365),
    "h0": 3500*(dias/365),
    "g6": 52.5*(dias/365),
    }
```

Figura 7.5. *Determinación de la demanda eléctrica total*

En la Figura 7.5 se aprecia como se han definido las demandas en kWh para los 3 tipos. Los 3 casos se han multiplicado por el número de días del estudio y se han dividido entre los días del año, ya que como se ha comentado las funciones de Demandlib realizan el cálculo para una demanda anual. En este caso, se divide entre 365 siempre, ya que así está implementado en la función que se ejecuta a continuación. La posibilidad de encontrarse en un año bisiesto ya se ha tenido en cuenta a la hora de realizar la función *diasHastaFecha*.

En el caso del tipo H0, se ha tomado como referencia el consumo anual medio de electricidad de una vivienda, que es de aproximadamente 3500 kWh [20] ,cuyo valor final a posterior se multiplica por el número de viviendas que se indiquen al programa.

En el caso de los tipos G0 y G6, la referencia ha sido el dato del consumo medio de un edificio por metro cuadrado de superficie útil, que se trata aproximadamente de 52,5 kWh/ m^2 [21] , el cual se multiplica en el programa por la superficie útil del edificio correspondiente.

Para finalizar este proceso, una vez habiendo definido esta serie de datos, se ha procedido al cálculo de los perfiles. Para ello, se ha empleado la función *ElecSlp* del paquete *bdew* que se ha importado como se observa en la Figura 7.6.

```
# A continuación se obtienen los perfiles de demanda
dias=dias*24
perfiles = bdew.ElecSlp(Dia1,Mes1,Año1,Dia2,Mes2,Año2,dias,
                        holidays=holidays)

# Aquí se multiplica por la demanda anual seleccionada
elec_demand = perfiles.get_profile(demanda_por_sector)

DemandaElectricakWh=elec_demand.truediv(4)

# Para graficar, se emplea la función resample para obtener valores
# cada hora
elec_demand_resampled = elec_demand.resample("H").mean()
```

Figura 7.6. *Desarrollo de los perfiles de demanda*

Como se observa en la Figura 7.6, la función *ElecSlp* requiere una serie de parámetros que no requería la función original, como son las diferentes fechas entre las que se lleva a cabo el estudio. El parámetro *dias* se ha multiplicado por 24, que son las horas de un día, ya que esos son los datos que son necesarios que sean aportados en el programa.

La función *ElecSlp* aporta datos cada 15 minutos, ya que ese parámetro *dias* que se le ha aportado, se multiplica por 4 dentro de la función para obtener los datos con esa frecuencia. Por lo que las unidades en las que se han obtenido los datos son kW cada 15 minutos. Debido a ello, el valor final se obtiene en la variable *DemandaElectricakWh*, dividiendo todos los factores que se obtienen cada 15 minutos entre 4, obtenido los valores con esa frecuencia, pero en las unidades de kWh, con las que se ha trabajado a lo largo de todo el desarrollo.

Por último, se ha realizado la gráfica del perfil de la demanda en este periodo. Se han tomado datos cada hora para reducir el número de datos, que se obtienen de la variable *elec_demand_resampled*.

```
# Gráfico de la demanda
ax = elec_demand_resampled.plot()
ax.set_xlabel("Fecha")
ax.set_ylabel("Demanda eléctrica")
plt.show()

for key in demanda_por_sector:
    assert np.isclose(
    elec_demand[key].sum() / 4, demanda_por_sector[key]
)
return DemandaElectricakWh,Mes1,Dia1,Mes2,Dia2,Año1,Año2,dias
```

Figura 7.7. *Gráfico de los perfiles de demanda*

Una vez se ha realizado lo que se muestra en la Figura 7.7, se obtiene la gráfica deseada para los 3 tipos. Es importante destacar que al haber especificado la demanda total dependiendo del número de viviendas o del número de metros cuadrados útiles, las gráficas que se han obtenido pueden parecer incompletas o erróneas, al ser los datos del tipo H0 mucho mayores que los de los tipos G0 y G6. Esto se debe a que el valor del número de viviendas siempre es menor que el número de metros cuadrados útiles de un edificio, por lo que a posterior cuando se desarrolla el programa y se multiplica por esos factores se obtienen los datos completos.

Para terminar se le ha especificado a la función *Demanda* la instrucción *return*, en la que se han recogido los diferentes parámetros de esta función que interesa obtener a posterior para el programa final.

7.2. Feedinlib

El siguiente paquete que se ha aplicado al diseño del programa es el módulo Feedinlib. Se trata de una aplicación del grupo Oemof, pero trabaja de forma independiente. Está preparado para trabajar con ello actualmente, sin embargo, tiene mucho margen de mejora y modelos que pueden ser mejorados y sus funcionalidades se pueden explotar de una mejor manera.

Su principal función es calcular series temporales de alimentación de energía fotovoltaica y eólica. Es decir, a partir de una serie de datos que se le aportan en el código, calcula la potencia que se podría obtener con un sistema de los mencionados. Este trabajo se ha centrado exclusivamente en el apartado fotovoltaico, por lo que no se ha empleado la parte de Feedinlib dedicada a la energía eólica, aunque su aplicación es similar a la de la parte fotovoltaica. Estos paquetes son gratuitos y de libre instalación. Se obtienen mediante la ejecución en la ventana de comandos de la siguiente secuencia:

- **pip install feedinlib**

De la misma manera que en el caso anterior, al ejecutarlo se ha observado como había limitaciones provocadas por la ausencia de módulos instalados en las últimas actualizaciones, por lo que se ha instalado la versión en desarrollo que se consigue ejecutando lo siguiente:

- **pip install https://github.com/oemof/feedinlib/archive/master.zip**

La ejecución de estas funciones se divide en dos partes principales. La primera es obtener los datos meteorológicos a lo largo de un periodo de tiempo con el objetivo de obtener la irradiancia solar. Feedinlib proporciona interfaces para descargar datos meteorológicos de diferentes maneras. Los datos que se han empleado han sido los de *open_FRED*, los cuales han sido utilizados para calcular la producción fotovoltaica del sistema. Esas interfaces aportadas por Feedinlib están proporcionadas para *pvlib* y *windpowerlib*, que son las librerías para calcular generaciones fotovoltaicas y eólicas, pero como se ha explicado solo se ha empleado la parte correspondiente a *pvlib*. Además, también dispone de los parámetros técnicos de muchos módulos e inversores fotovoltaicos, así como de turbinas eólicas, que se utilizan para realizar los cálculos del sistema.

En segundo lugar, una vez obtenidos los datos meteorológicos, se ha procedido a realizar el cálculo de la generación fotovoltaica, obteniendo la potencia que se obtiene para los datos seleccionados.

Para la realización de estos 2 apartados, se han definido 3 funciones que se explican a continuación, para lo que se han importado los siguientes paquetes:

```
from feedinlib.powerplants import Photovoltaic
from feedinlib.open_FRED import Weather
from feedinlib.open_FRED import defaultdb
from shapely.geometry import Point
import warnings
warnings.filterwarnings("ignore")
from feedinlib.models import get_power_plant_data
import matplotlib.pyplot as plt
```

Figura 7.8. *Paquetes importados para las funciones de Feedinlib*

1. Selección de los módulos e inversores

Para la definición de los módulos e inversores fotovoltaicos que se han seleccionado para el sistema, primero se ha definido esta función que muestra los diferentes tipos que existen y después se ha optado por los que más se acercan a los parámetros esperados. Aquí ha aparecido una de las primeras limitaciones de Feedinlib, ya que a pesar de que hay un amplio número de tipos, los parámetros pueden no ser iguales a los de los componentes que se deseen instalar, por lo que se puede producir cierto error en los cálculos. Sería conveniente una mayor base de datos y una actualización de los diferentes tipos más frecuente.

La función que se ha creado para mostrar lo mencionado es *ModulosInversores*, cuyo código es el mostrado en la Figura 7.9:

```
#Con esta función se obtienen los diferentes módulos fotovoltaicos e
#inversores que permite el programa.
def ModulosInversores():
    #obtener los módulos
    module_df = get_power_plant_data(dataset='sandiamod')
    # imprimir los módulos
    module_df.iloc[:, 1:5]
    #obtener inversores y imprimirlos
    inverter_df = get_power_plant_data(dataset='cecinverter')
    inverter_df.iloc[:, 1:5]
    return module_df,inverter_df
```

Figura 7.9. *Obtención de los módulos e inversores fotovoltaicos*

Con la sentencia *get_power_plant_data* se obtienen los diferentes tipos. La sentencia que se realiza a continuación (*iloc*) sirve para imprimir un determinado número de elementos, en este caso son los que se encuentran entre el primero y el quinto, pero se puede seleccionar el rango deseado en cada caso.

En el caso de los módulos, el empleado para el sistema ha sido uno cuyo nombre es "Advent_Solar_Ventura_210___2008_". Los datos que se han obtenido se muestran en la Tabla 7.2. Por su parte, el inversor seleccionado ha sido el denominado "ABB__PVI_3_0_OUTD_S_US__240V_". Sus correspondientes datos se muestran en la Tabla 7.3.

Tabla 7.2. *Datos módulo fotovoltaico*

Index	Advent_Solar_Ventura_210__2008_	Index	Advent_Solar_Ventura_210__2008_
Vintage	2008	C3	-10.758
Area	1.646	A0	0.9067
Material	mc-Si	A1	0.09573
Cells_in_Series	60	A2	-0.0266
Parallel_Strings	1	A3	0.00343
Isco	8.34	A4	-0.0001794
Voco	35.31	B0	1
Impo	7.49	B1	-0.002438
Vmpo	27.61	B2	0.00031
Aisc	0.00077	B3	-1.246e-05
Aimp	-0.00015	B4	2.11e-07
C0	0.937	B5	-1.36e-09
C1	0.063	DTC	3
Bvoco	-0.133	FD	1
Mbvoc	0	A	-3.45
Bvmpo	-0.135	B	-0.077
Mbvmp	0	C4	0.972
N	1.495	C5	0.028
C2	0.0182	IXO	8.25
		IXXO	5.2

Tabla 7.3. *Datos inversor fotovoltaico*

Index	ABB__PVI_3_0_OUTD_S_US__240V_
Vac	240
Pso	16.8808
Paco	3000
Pdco	3121.67
Vdco	340
C0	-5.7019e-06
C1	-2.1e-05
C2	0.000582
C3	-0.000712
Pnt	0.1
Vdcmax	480
Idcmax	9.18138
Mppt_low	100
Mppt_high	480
CEC_Date	nan
CEC_Type	Utility Interactive

2. Cálculo de los datos meteorológicos

En este apartado la función que se ha creado tiene el nombre de *IrradianciaSistemaFotovoltaico*. Esta requiere de una serie de parámetros de entrada, que se deben aportar a posterior a la hora de llamar a la función, como lo son *Mes1*, *Dia1*, *Mes2*, *Dia2*, *Año1*, *Año2*, es decir, los días entre los que se desea realizar el estudio.

Para comenzar, se ha definido el sistema fotovoltaico, aportando los diferentes parámetros de la Figura 7.10.

```
def IrradianciaSistemaFotovoltaico(Mes1,Dia1,Mes2,Dia2,Año1,Año2):

    azimuth=int(input("Indica la orientación que van a tener los paneles"
                      " teniendo en cuenta lo siguiente:\n"
                     "-Norte=0\n"
                     "-Este=90\n"
                     "-Sur=180\n"
                     "-Oeste=270\n"
                     "IMPORTANTE: En caso de poder colocar los paneles en"
                     " cualquier dirección, hacerlo en la dirección "
                     "Sur=180, donde se conseguirá la mayor potencia.\n"
                     "\nSelección:"))

    system_data = {
        'module_name': 'Advent_Solar_Ventura_210___2008_',
        'inverter_name': 'ABB__PVI_3_0_OUTD_S_US__240V_' ,
        'azimuth': azimuth,
        'tilt': 30,
        'albedo': 0.2}
    placas=int(input("Indica la cantidad de módulos fotovoltaicos que se"
                     " van a instalar en este edificio: "))
    system_data['modules_per_string']= placas

    #con estos datos se genera el sistema
    pv_system = Photovoltaic(**system_data)
```

Figura 7.10. *Introducción de parámetros y generación de sistema fotovoltaico*

Los primeros datos que se han introducido son los nombres de los módulos e inversores seleccionados en el apartado anterior. A continuación, se han definido 3 datos de la geometría solar correspondiente a los paneles, dos de ellos se han establecido como constantes y otro cuyo valor se solicita que sea ingresado.

El *tilt* corresponde a la inclinación del panel, que para que sea óptima tiene que encontrarse entre 20º y 40º, por lo que se ha seleccionado la inclinación de 30º que es la óptima.

El *albedo* se trata de una forma de cuantificar la radiación que se refleja desde la superficie. Es un parámetro que se tiene que encontrar entre 0 y 1, en este caso se ha tomado el valor de 0,2 como aproximación para todo el planeta.

El *azimuth* o *acimut* se define como el ángulo que forma la proyección sobre el plano horizontal de la perpendicular a la superficie del módulo con el meridiano del lugar. El ángulo donde se obtiene una mayor potencia es el Sur, que en la librería Feedinlib implica darle un valor de 180 al *azimuth*, según lo establecido por el programa. De la misma manera, el valor 0 representa al Norte, 90 corresponde a la dirección Este y 270 al Oeste. El programa requiere que se indique el valor ya que para cada caso resulta diferente dependiendo de las posibilidades de la instalación.

En la siguiente figura se presenta una representación de la geometría solar, donde se muestran tanto el azimuth como el tilt o inclinación del panel. Cabe destacar que los valores del azimuth representados son diferentes a los requeridos por el programa, pero resulta útil para ver gráficamente lo explicado.

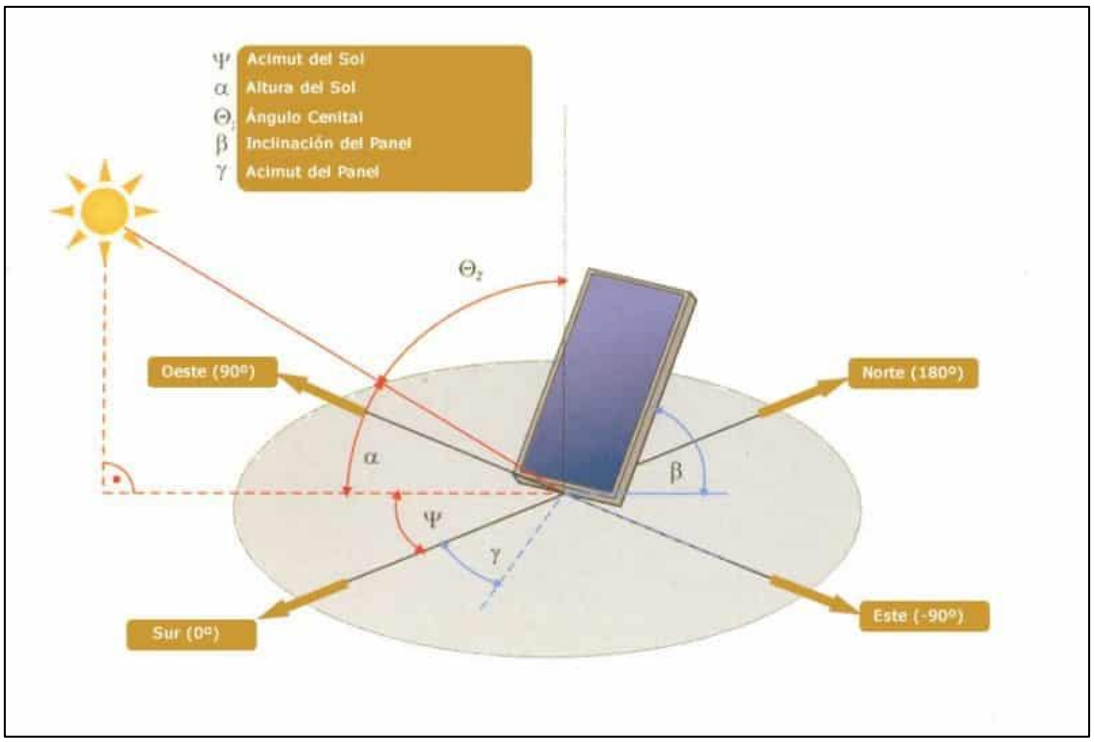

Figura 7.11. *Representación geometría solar [22]*

El último dato que queda por definir el sistema es el número de módulos fotovoltaicos, que se trata de un parámetro que se debe indicar para cada edificio. Con todos estos datos recogidos, se ha procedido a generar el sistema con la función *Photovoltaic*, que ha sido importada de las librerías de Feedinlib.

Una vez definido el sistema, los siguientes pasos han sido definir las variables que faltan para calcular los datos meteorológicos, que son la ubicación de instalación de los paneles y el periodo del estudio.

```
#aqui se da la ubicación del sistema y se obtienen los datos
latitud=float(input("Ingresa la latitud del punto de colocación de los"
                    " paneles fotovoltaicos: "))

longitud=float(input("Ingresa la longitud del punto de colocación de "
                     "los paneles fotovoltaicos: "))

lat = latitud
lon = longitud
location = Point(lon, lat)
```

Figura 7.12. *Localización del estudio*

Como se observa en la Figura 7.12, la manera para definir donde se debe localizar el estudio es mediante las coordenadas de latitud y longitud de ese punto, variables que el programa solicita que sean ingresadas. Se les ha dado a estas dos coordenadas el formato *Point* importado del paquete *shapely.geometry* y se ha obtenido la variable *location*.

```
        print("El año límite para el cálculo de la irradiancia es el 2018 por"
              " limitaciones del programa,"
              "por lo que se va a emplear este año o años previos,"
              "en caso de que la duración sea mayor a 1 año, para el cálculo"
              " de la irradiancia")
        print("-------------------------------------------------------------------"
              "-----------------------------")
        if Año2<=2018:
            inicio=f"{Año1}/{Mes1}/{Dia1}"
            final=f"{Año2}/{Mes2}/{Dia2}"

        elif Año2==2019 and Mes2==1 and Dia2==1:
            inicio=f"{Año1}/{Mes1}/{Dia1}"
            final=f"{Año2}/{Mes2}/{Dia2}"

        else:
            if Mes2==1 and Dia2==1:
                Año=Año2-Año1
                inicio=f"{2018-Año}/{Mes1}/{Dia1}"
                final=f"2019/{Mes2}/{Dia2}"
            else:
                Año=Año2-Año1
                inicio=f"{2018-Año}/{Mes1}/{Dia1}"
                final=f"2018/{Mes2}/{Dia2}"
```

Figura 7.13. *Definición del periodo de estudio*

Como se lee en la Figura 7.13, en este apartado se ha encontrado otra de las limitaciones de Feedinlib. El año límite para el cálculo de la irradiancia ha sido el año 2018. El año 2019 ha permitido la ejecución del programa, pero no ha aportado ningún dato, mientras que a partir del año 2020 no ha ejecutado el programa y ha devuelto un mensaje de error. Debido a ello, dado que como se ha explicado se quiere que el programa permita la ejecución en cualquier periodo de tiempo, se ha diseñado una secuencia que permita la ejecución del programa para años posteriores, suponiendo que las condiciones meteorológicas a lo largo del año son similares.

En caso de que el año final del estudio sea anterior a 2019, o en el caso de que el día final sea el 1/1/2019, las fechas que se introducen son las mismas que las que se han definido. La segunda de estas posibilidades se debe a que el cálculo se hace hasta el 1/1/2019 a las 00:00, por lo que hasta ese punto sí existen datos.

En el caso de que en un año posterior a 2019 el día final sea el 1 de enero, por lo que se ha explicado en la anterior condición, el último día del estudio es el 1/1/2019, y el año de inicio se obtiene de restar al 2018 la diferencia entre el año de inicio y el año de final. Lo mismo se ha aplicado para el caso de que el día final no sea el 1 de enero. El año inicial resulta de restar a 2018 la diferencia de los años de estudio, mientras que en este caso el año final es el 2018.

Se encuentran definidos todos los datos que requiere la función *Weather* ,que se ha importado de la librería *feedinlib.open_FRED*, con la que se obtienen los datos meteorológicos.

```
    open_FRED_weather_data = Weather(
        start=inicio, stop=final,
        locations=[location],
        variables="pvlib",
        **defaultdb())

    weather_df = open_FRED_weather_data.df(location=location, lib="pvlib")

    return pv_system, weather_df, lat, lon
```

Figura 7.14. *Cálculo de las condiciones meteorológicas del estudio*

Se comprueba en la Figura 7.14 que los parámetros que hay que introducir en la función *Weather* son las fechas de inicio y final, las coordenadas definidas en la variable *location*, y a parte se le ha indicado que trabaje con la librería pvlib. Para finalizar este apartado, se le ha dado un *return* a la función para que al ejecutarla devuelva los valores que después se emplean, como son las variables del sistema fotovoltaico, la de los datos meteorológicos, y la latitud y la longitud del punto de instalación.

3. Cálculo de la potencia del sistema

La tercera de las funciones de Feedinlib cierra el proceso realizado y consigue llegar a los datos de potencia que puede generar la instalación. Se ha credo la función *PotenciaenkW* para desarrollarlo, cuyos parámetros de entrada son los obtenidos de salida de la anterior función *IrradianciaSistemaFotovoltaico*.

```
def PotenciaEnkW(weather,latitud,longitud,sistema,a):

    feedin = sistema.feedin(
    weather=weather,
    location=(latitud, longitud))
    feedin=feedin/1000
    feedin=feedin.replace(-0.0001,0)

    ax1 = feedin.plot(title=f"Generación {a}")
    ax1.set_xlabel('Tiempo')
    ax1.set_ylabel('Potencia en kW')
    plt.show()

    return feedin
```

Figura 7.15. *Cálculo de la potencia generada*

En este caso, para ello se ha empleado la función *feedin*, que es la que realiza este cálculo. Los datos que se han obtenido tienen como unidades los vatios, por lo que se ha dividido ese valor entre 1000 para así obtener el valor en kilovatios, que son las unidades en las que se ha trabajado en el resto del trabajo. También se le ha indicado a la función que represente en una gráfica la potencia en función del tiempo de duración, para poder observar gráficamente a lo largo del tiempo.

7.3. Oemof.solph

El paquete oemof.solph es un generador de modelos para el modelado y optimización de sistemas energéticos. Está diseñado para para crear y resolver problemas de optimización lineal o lineal de enteros mixtos. Se trata de un paquete gratuito y de libre instalación, y que además permite la contribución de todo el mundo, es decir, existe la posibilidad de corregir errores o implementar mejoras por cualquier usuario. Su instalación se realiza de la siguiente manera:

- **pip install feedinlib**

Al igual que los casos anteriores, para disponer de las últimas funciones desarrolladas existe la posibilidad de instalar la versión en desarrollo que se consigue ejecutando lo siguiente:

- **pip install https://github.com/oemof/oemof-solph/archive/dev.zip**

Una vez realizada la instalación, para que trabajen correctamente las funciones del oemof.solph es necesario instalar un solver. Existen varios solvers comerciales y de código abierto, pero se ha instalado el recomendado por Oemof, que se trata del CBC (Coin-or branch and cut). Una vez realizados estos dos pasos, se ha comprobado que todo funciona correctamente y que el solver está instalado, obteniendo un mensaje como el que se observa en la Figura 7.16.

```
****************************
Solver installed with oemof:

cbc: working
glpk: not working
gurobi: not working
cplex: not working

****************************
oemof successfully installed.
****************************
```

Figura 7.16. *Comprobación de la correcta instalación*

La parte inicial para la generación de un problema implica establecer el sistema energético. Para ello hay que definir una variable de tipo *EnergySystem* que es el contenedor principal del modelo de sistema de energía. Para definir esto hay que establecer un índice de tipo *Datetime* para definir el rango de tiempo y su correspondiente incremento para el modelo.

La realización de esta parte principal se ha mostrado en el desarrollo del programa principal, dado que las variables temporales que se han aplicado son las definidas por el usuario en él. De la misma manera, también en él se ha profundizado sobre la parte final de este proceso, que es la que recoge la optimización, la solución del problema, y su correspondiente análisis.

Este apartado se ha centrado en los diferentes tipos de componentes que se pueden añadir a dicho sistema energético definido, los datos que requieren y la manera de introducirlos. Para crear un modelo de sistema energético, el oemof.solph dispone de componentes genéricos y específicos, que se explican a continuación.

Una vez definido el explicado elemento de tipo EnergySystem, se debe agregar todos los nodos necesarios para definir el sistema. Existen dos tipos de nodos: componentes y buses. Todo componente tiene la obligación de estar conectado a uno o más buses. La conexión realizada entre un bus y un componente viene definida por el flujo.

Todos los componentes de oemof.solph se pueden emplear para configurar un modelo de sistema de energía, pero hay que tener cuidado ya que no es posible conectar todos los componentes a todos los flujos, por lo que conviene leer la documentación de cada componente para conocer el uso y las restricciones.

Bus:

Todos los flujos que entran y salen de un bus están balanceados. Por lo tanto, una instancia de la clase Bus representa una cuadrícula o red sin pérdidas. La función creada para la definición de un bus es la de la Figura 7.17, denominada *Buses*.

```
#Representación de los buses.
def Buses(a):
    bus=solph.Bus(label=f"Electricidad_{a}")
    return bus
```

Figura 7.17. *Definición de buses*

El único parámetro que se le ha dado a la función *solph.Bus* es la etiqueta (*label*), que resulta muy útil a la hora de analizar los resultados para obtener una etiqueta diferente en cada bus. El parámetro *a* es un parámetro de entrada, que se le debe asignar a la función, y está representado con la función de obtener etiquetas diferentes para cada bus. Cada vez que se defina un bus, se le suma una unidad, de esa manera se consigue diferenciar las diferentes etiquetas por el número final. Se ha empleado una variable de tipo *f-String* para poder realizar esto.

Flow:

Esta clase debe usarse para conectar los buses con los componentes. Las instancias de tipo Flow se emplean normalmente en la definición de los componentes. La sentencia para crearlo es *solph.Flow()*. Un flujo básico se puede definir sin ningún parámetro. Puede estar limitado por límites superiores e inferiores (constantes o dependientes del tiempo) o por límites resumidos.

Componentes:

Los componentes se dividen en tres categorías. Componentes básicos (solph.network), componentes adicionales (solph.components) y componentes personalizados (solph.custom). La sección personalizada se creó para reducir la barrera de entrada de nuevos componentes, por lo que estos componentes corresponden a los que se encuentran en estado experimental.

Para la definición del trabajo se han empleado únicamente componentes básicos y componentes adicionales, por los que son los que se definen a continuación.

Componentes básicos:

Existen 3 tipos de componentes básicos. Se han utilizado todos ellos para la definición del problema, y son los siguientes:

Sink:

Un componente tipo Sink se usa para definir la demanda dentro de un modelo energético. Su instrucción es *solph.Sink*.

```
#Representación de la demanda del edificio, empleando la función Demanda de
#Demandlib1
def DemandaEdificio(bus,Demanda,a):
    DemandaEdificio=solph.Sink(label=f"Demanda_{a}",
    inputs={bus:
    solph.Flow(fix=Demanda, nominal_value=1,
    Fixed=True)})
    return DemandaEdificio
```

Figura 7.18. *Definición de la demanda de un edificio*

En la Figura 7.18 se representa la función *DemandaEdificio* para generar el componente de la demanda de un edificio. Tiene 3 parámetros de entrada: el bus al que se conecta, el parámetro *a* que al igual que en el caso de los buses sirve para definir la etiqueta y la función *Demanda*, que es la que se ha definido en el apartado 7.1. en la explicación de la librería Demandlib.

Como se comprueba, la conexión con el bus es de tipo input, es decir, el flujo entra en el elemento *DemandaEdificio*. El flujo viaja desde el bus seleccionado hasta el elemento tipo Sink. Este componente indica la demanda del edificio correspondiente, y el bus al que se conecta debe proporcionarle dicha demanda para que el sistema funcione correctamente, sino se completa este flujo, el edificio no dispone de la energía demandada.

La otra función de un componente tipo Sink es la de detectar excesos. Se ha realizado el mismo proceso anterior, pero para definir el exceso del sistema, como se representa en la función *Exceso* de la Figura 7.19.

```
def Exceso(bus_conexion,coste):
    exceso = solph.Sink(label="Exceso",
                        inputs={bus_conexion:
                                solph.Flow(variable_costs=coste)})
    return exceso
```

Figura 7.19. *Definición del exceso del sistema*

En este caso, no existe parámetro *a*, debido a que solo existe un componente *Exceso* para todo el sistema. Este se ha conectado al bus conexión, que es el bus al que se le unen todos los diferentes subsistemas, en el que se realiza el cálculo general, y en el caso de que en todo el sistema haya un excedente de energía, el flujo va de ese bus al componente *Exceso*, que representa la venta de energía, a la red.

Al flujo se le ha asignado una variable *coste*, que corresponde al precio de venta de energía a la red, ya que al sistema hay que indicarle el precio que se obtiene de vender energía a la red, para que determine si resulta conveniente o existen opciones mejores donde destinar la energía.

Source:

Un componente tipo Source o Fuente sirve para representar un sistema fotovoltaico, una planta de energía eólica o una importación de gas natural. En este caso se ha empleado para definir sistemas fotovoltaicos, como se ve en la función *PlacasSolares* de la Figura 7.20.

```
#Función para la generación mediante placas solares
def PlacasSolares(bus,Potencia,a):
    placas=solph.Source(label=f"PlacasSolares_{a}",
    outputs={bus: solph.Flow(fix=Potencia,
    nominal_value=1, Fixed=True)})
    return placas
```

Figura 7.20. *Definición de la generación fotovoltaica*

Los componentes de entrada en este caso también son la variable *a* y el bus al que se debe conectar el componente, pero en este caso a diferencia del anterior, la tercera variable de entrada es la potencia que se obtiene con los paneles solares que se definan en cada subsistema. Esta variable potencia es la definida como *feedin* en el apartado 7.2., en el desarrollo de la librería Feedinlib.

La conexión con el bus es de tipo output. En los componentes tipo Source el flujo viaja desde el propio componente hasta el bus al que se conecte. Se le ha indicado la potencia que es capaz de generar dicho elemento, y esa energía sale del componente para transferirse al bus al que se ha conectado.

La otra función de la que se dispone es la de representar el posible déficit de energía del sistema, para lo que se ha definido la función *Deficit* representada en la Figura 7.21, que actúa de la misma manera que la función *Exceso*, pero en la dirección contraria, es decir, actúa si en el sistema hay falta de energía y se necesita comprar energía a la red.

```
def Deficit(bus_conexion,coste):
    deficit=solph.Source(label="Deficit",outputs={bus_conexion:
    solph.Flow(variable_costs=coste)})
    return deficit
```

Figura 7.21. *Definición del déficit de energía del sistema*

Solo existe uno de estos componentes, que también se ha conectado al bus conexión del sistema, en este caso para solucionar los posibles problemas de ausencia de energía. También se le asigna al flujo la variable *coste*, pero en este caso debe representar el precio de compra de energía a la red.

Transformer

El tercero de los tipos de componentes básicos es el de tipo Transformer, que representa un nodo con múltiples flujos de entrada y salida, como una planta de energía, una línea de transporte o cualquier tipo de proceso de transformación como electrólisis, un dispositivo de enfriamiento o una bomba de calor.

En la Figura 7.22 se representa la función *Conexion*, donde se representa su aplicación en el presente trabajo.

```
#La siguiente función se emplea para conectar los elementos del sistema.
def Conexion(bus_in,bus_out,a,b):
    Conexion=solph.Transformer(label=f"trafo_{a}_{b}",
    inputs={bus_in: solph.Flow()}, outputs={bus_out: solph.Flow()},)
    return Conexion
```

Figura 7.22. *Definición del déficit de energía del sistema*

En este caso, la aplicación que se le ha dado a los componentes Transformer es la de representar la unión entre los buses de los diferentes subsistemas con el bus conexión del sistema completo, del que ya se ha hablado en las funciones *Exceso* y *Deficit*.

Para la correcta definición del sistema, para cada unión entre un bus del subsistema con el bus conexión se ha requerido de la implementación de dos componentes Transformer, uno que represente el flujo en la dirección desde el bus conexión del sistema hasta el bus del subsistema, y otro que vaya en la dirección contraria.

Se observa en la Figura 7.22, al contrario que en los componentes Sink y Source, que en este caso se va a tener entradas representadas bajo la variable inputs, y salidas representadas bajo la variable outputs. Podrían disponer un número cualquiera de entradas y salidas, pero en este sistema solo se han empleado Transformers con una entrada y una salida. Debido a ello, los parámetros de entrada de esta función son el bus de entrada y el bus de salida de cada componente.

Además, se han definido dos parámetros de entrada más para el etiquetado de cada transformador. El primero es la variable *a*, de la que ya se ha hablado previamente, y que representa el número de subsistema al que está conectado el bus conexión general. La segunda es la variable *b*, que tiene dos valores: "IN "y "OUT". Si toma el valor "IN", representa que el flujo va desde el bus conexión hasta el bus del subsistema número *a*., mientras que, si toma el valor OUT, supone que el flujo entra desde el subsistema número *a* hasta el bus conexión.

Componentes adicionales

Existen diferentes tipos de componentes adicionales como lo son *ExtractionTurbineCHP*, que se utiliza para representar una planta combinada flexible de calor y electricidad; o *GenericCHP*, que sirve para modelar diferentes tipos de plantas CHP (Combined Heat and Power), como turbinas de extracción de ciclo combinado, turbinas de contrapresión y cogeneración motorizada.

Pero el único componente adicional que se ha aplicado es el de tipo GenericStorage, que sirve para modelar un sistema de almacenamiento con sus características básicas. Se ha creado la función *Bateria*, que se muestra a continuación:

```python
def Bateria(bus,a,cap,Pcarga,Pdescarga):
    bateria = solph.GenericStorage(label=f"bateria_{a}",
    nominal_storage_capacity=cap,
    inputs={bus: solph.Flow(nominal_value=Pcarga,
    variable_costs=0.001)},
    outputs={bus: solph.Flow(nominal_value=Pdescarga,
    variable_costs=0.001)},
    loss_rate=0.00,
    initial_storage_level=0.5,
    balanced=False,
    inflow_conversion_factor=0.9, #eficiencia al cargar
    outflow_conversion_factor=0.9, #eficiencia al descargar
    min_storage_level=0.2, #Estado de carga (SoC mín)
    max_storage_level=0.8, #Estado de carga (SoC máx)
    )

    return bateria
```

Figura 7.23. *Definición de un sistema de almacenamiento*

Esta función cuenta con 5 parámetros de entrada. Los 2 primeros son el bus en el que se instala el almacenamiento y el número del bus correspondiente. Los 3 últimos se han empleado para asignar los valores de los diferentes parámetros de diseño de las baterías. Al establecer los valores de capacidad nominal y potencia de carga y descarga como parámetros de entrada, se tiene una gran capacidad de estudio, ya que se pueden variar y comprobar su influencia en cada situación.

Respecto a la definición del componente GenericStorage, lo primero que se ha realizado es indicarle la etiqueta correspondiente como en el resto de los componentes. Lo siguiente ha sido indicar la energía nominal de la que dispone dicho sistema de baterías. El valor indicado está en las unidades de kWh, y puede ser diferente para cada edificio.

A continuación, los elementos GenericStorage están diseñados para tener una entrada y una salida, ya que se requiere que las baterías aporten energía al bus al que están unidas en caso de necesitarla, y que las baterías almacenen el excedente del bus en caso de haberlo, por lo que se ha procedido a su definición. A la entrada se le ha asignado el bus al que se le conecta y la potencia de carga de la que disponen en kW. A la salida se le ha conectado al mismo bus, y se le ha indicado la potencia de descarga en las mismas unidades que la de carga.

A ambas se les ha dado un parámetro de costes que toma el valor de 0.001,ya que el flujo desde el bus hasta las baterías o en la dirección contraria no resulta costoso, al contrario que en el caso de comprar energía a la red o de venderla, por lo que al resolver el problema se da prioridad al almacenamiento en las baterías.

El problema con este tipo de solución radica en los costes de las baterías, que a pesar de que están teniendo una gran evolución, siguen resultando muy caras, por lo que puede resultar complicado su amortización dependiendo de cada caso. Se debe decidir a la hora de diseñar el sistema si se quiere optar por la opción de instalar un sistema de almacenamiento.

Los siguiente parámetros que se han indicado son el porcentaje de pérdidas y el nivel de almacenamiento inicial. Respecto a este último, existen 4 posibilidades de diseño que se le pueden dar los siguiente valores:

-*initial_storage_level*=None, *balanced*=True: Esto sirve para representar que el estado de carga en el paso de tiempo cero es el resultado de la optimización, y el estado de carga del último paso de tiempo es igual al paso de tiempo cero.

-*initial_storage_level*=0.5, *balanced*=True: El estado de carga en el paso de tiempo cero se fija en 0,5 (50 % cargado). El estado de carga en el último paso de tiempo también está limitado por 0,5.

-*initial_storage_level*=None, *balanced*=False: Tanto el estado de carga en el paso de tiempo cero como el último paso de tiempo son el resultado de la optimización y no están acoplados.

-*initial_storage_level*=0.5, *balanced*=False: El estado de carga en el paso de tiempo cero está limitado por un valor dado. El estado de carga del último paso de tiempo es el resultado de la optimización.

Se ha optado por la última de estas opciones, para tener en el estado inicial las baterías cargadas al 50% y en el estado final se tiene en ellas el resultado de la optimización.

Por último, se han definido las eficiencias de carga y descarga, así como los niveles mínimo y máximo de almacenamiento que se necesita que haya en estas baterías.

En el programa final se ha dado la posibilidad de aparte de instalar un sistema de almacenamiento en cada subsistema, poder instalar un almacenamiento para el sistema completo. Por lo que para ello se ha definido otra función denominada *BateriaGen*, que tiene un desarrollo similar al anterior.

Se le ha asignado a la variable de costes un valor algo superior para que el programa priorice primero el uso de las baterías de los subsistemas, y que la batería general se emplee solo en caso de requerir de mayor almacenamiento. Es importante remarcar que el precio de éstas es elevado, por lo que es importante valorar correctamente su instalación. El código que tiene la función *BateriaGen* por lo tanto es el mostrado en la Figura 7.24.

```
def BateriaGen(bus,cap,Pcarga,Pdescarga):
    bateria = solph.GenericStorage(label="BateriaGeneral",
    nominal_storage_capacity=cap,
    inputs={bus: solph.Flow(nominal_value=Pcarga,
    variable_costs=0.002)},
    outputs={bus: solph.Flow(nominal_value=Pdescarga,
    variable_costs=0.002)},
    loss_rate=0.00,
    initial_storage_level=0.5,
    balanced=False,
    inflow_conversion_factor=0.9, #eficiencia de carga
    outflow_conversion_factor=0.9, #eficiencia de descarga
    min_storage_level=0.2, #Soc min
    max_storage_level=0.8, #Soc máx
    )
    return bateria
```

Figura 7.24. *Definición del sistema de almacenamiento general*

Se observa como en este caso también los valores de los parámetros de las baterías se pueden variar cada vez que se ejecute el programa, teniendo una gran variedad de posibilidades para el estudio posterior.

8. DESARROLLO DE LA SOLUCIÓN

Esta sección describe el software desarrollado, el cual está en gran medida basado en el uso de las funciones de los 3 paquetes presentados en la anterior sección.

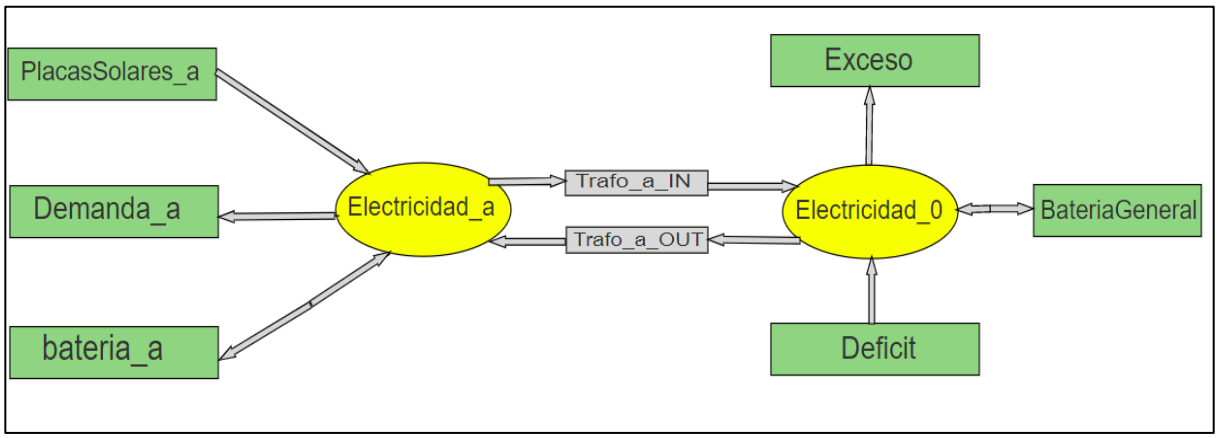

Figura 8.1. *Representación sistema desarrollado*

En la Figura 8.1 se muestra la representación gráfica del sistema final que se ha obtenido con la ejecución del trabajo, con los nombres sin tildes ya que no se emplean al nombrarlas. La parte izquierda, compuesta por los elementos que llevan en su nombre como índice la letra *a*, se trata de la representación de los edificios y sus componentes. En el gráfico se ha representado únicamente el sistema correspondiente a un edificio, pero el programa está diseñado para que el sistema tenga el número de edificios que el usuario seleccione, por lo que el sistema final tiene un número *a* ramas como la representada, ya que los componentes son los mismos para todos los edificios.

El bus *Electricidad_a* se trata del bus de cada edificio. A él entra la energía correspondiente a la generación fotovoltaica, representada como *PlacasSolares_a*, y la energía de la batería de ese edificio, en el caso de tenerla disponible, definida como *bateria_a*. De él sale energía para cubrir la demanda de dicho edificio, representada como *Demanda_a*, así como la energía excedente para almacenarla en las baterías, que también se define como *bateria_a*, ya que como se ha explicado previamente, las baterías son elementos de dos direcciones, pueden aportar energía o pueden recibirla.

Para conectar los buses de cada edificio al bus de conexión de todos los edificios, llamado *Electricidad_0*, se han instalado dos elementos. Uno es el encargado de transmitir la energía desde el edificio hasta el elemento de conexión, definido como *Trafo_a_IN*. El otro es el que se ocupa de hacerlo en la dirección contraria, desde el bus de conexión hacia el bus de dicho edificio, llamado *Trafo_a_OUT*.

La parte derecha corresponde a los elementos independientes del número de edificios, por lo que su nombre no lleva el índice *a*. Como se ha mencionado, uno de ellos es el bus de conexión de los edificios. Como el sistema se ha conectado a la red para tener la posibilidad de vender o comprar energía a la red, se han instalado dos elementos. El primero es el que representa la compra de energía a la red, en caso de no disponer de suficiente, cuyo nombre es *Deficit*. El

segundo es el que representa la venta de electricidad a la red, en caso de tener un excedente, denominado *Exceso*.

Por último, en el programa se ha dado también la posibilidad de instalar un sistema de baterías para el sistema completo, a parte de las posibles baterías de cada edificio, por lo que este elemento recibe el nombre de *BateriaGeneral*, también de dos direcciones como el resto de las baterías, como se observa en la Figura 8.1.

Para comenzar con la definición del programa final, primero se han importado los paquetes necesarios. En la Figura 8.2 se tiene los diferentes paquetes importados, donde aparecen todas las funciones definidas previamente, así como el resto de las funciones necesarias para su correcto funcionamiento.

```
from funciones.feedinlib1 import FeedIn
from funciones.demandlib1 import Demand
from funciones.Solph import FuncionesSolph
import pandas as pd
from oemof import solph
import logging
import matplotlib.pyplot as plt
import os
```

Figura 8.2. *Paquetes importados para el programa final*

El siguiente paso ha consistido en establecer el sistema energético, como se ha indicado en el apartado 7.3. El paso previo a ello ha requerido la llamada a la función *Demanda* desarrollada en el apartado 7.1., ya que como se comprueba en la Figura 8.3, varios parámetros de esta función como son la fecha de inicio del proyecto o los periodos de los de que dispone la solución, son necesarios para la definición del sistema.

```
Demanda15min,Mes1,Dia1,Mes2,Dia2,Año1,Año2,periodos=Demand.Demanda()

print("")
print("------------------------------------------------------------------------"
      "----------------------")
print("Una vez seleccionado el periodo en el que se va a realizar el estudio, "
      "se va a proceder a diseñar el sistema.\n")
inicio=f"{Año1}/{Mes1}/{Dia1}"


logging.info("Inicializar el sistema")
date_time_index=pd.date_range(inicio,periods=periodos*4, freq="15min")
#formato fecha mm/dd/yyyy
energysystem=solph.EnergySystem(timeindex=date_time_index)
```

Figura 8.3. *Generación del sistema energético*

Se ha creado la variable *energysystem*, donde se han almacenado los diferentes datos que se seleccionan en el programa para luego proceder a la solución y optimización del problema. Para ello se le ha dado un rango de tiempo establecido con la variable *date_time_index*, donde se ha definido el inicio del estudio, los periodos de los que dispone el mismo, y la frecuencia de datos con la que se ha trabajado, que en este caso ha sido de 15 minutos.

Una vez definido el sistema, se ha procedido a introducir los elementos que forman parte de él. Existen ciertos elementos que dependen de lo indicado al ejecutar el programa, pero a su vez existen otros que son independientes del sistema diseñado, como se ha observado en la Figura 8.1, por lo que estos son los primeros que se han definido. El primer paso ha sido definir la variable *a*, que se le ha dado el valor 0, ya que a la hora de definir los siguientes elementos y darles nombre, se les ha asignado el valor 0 al nombre para poder identificarlos a posterior.

Las variables definidas en la Figura 8.4 son el bus de conexión de todos los buses que contenga el sistema final, y las componentes de exceso y déficit.

```
#Se crean los buses de conexión y los componentes de exceso y déficit,
#que van a ser independientes del número de edificios que tenga el sistema
a=0
BusConexion=FuncionesSolph.Buses(a)
Excedente=FuncionesSolph.Exceso(BusConexion, data["precioexcedente"])
Deficit=FuncionesSolph.Deficit(BusConexion, data["preciocompra"])
energysystem.add(BusConexion,Excedente,Deficit)
```

Figura 8.4. *Definición elementos independientes del sistema*

El bus de conexión es muy importante, ya que a él se han conectado el resto de los buses que existan en el sistema, para luego realizar la solución del problema y el balance de la energía necesaria en cada uno de ellos. El exceso y el déficit son la conexión a la red del sistema. Esto supone que, en caso de tener un déficit de energía en el sistema, se obtiene energía de la red, pagando el correspondiente precio al que se encuentre. Lo mismo sucede en la dirección contraria, si hay exceso de energía, existe la posibilidad de venderla a la red por su precio correspondiente. Ambos componentes se han conectado exclusivamente al bus de conexión, el de exceso saliendo del bus de conexión al exceso, y el de déficit en la dirección contraria.

Para establecer los precios de estas variables, se ha llamado primero a un archivo Excel en el que se han definido los precios de ambas variables, denominado *Precios* como se ve en la Figura 8.5.

```
filename=os.path.join(os.getcwd(),"Precios.csv")
data=pd.read_csv(filename, sep=";")
```

Figura 8.5. *Llamada al archivo Excel con los precios de excedente y déficit*

La obtención de estos precios se ha realizado mediante la página web de E·SIOS(Sistema de Información del Operador del Sistema) [24] ,donde aparecen recogidos estos parámetros y se puede especificar el periodo deseado y la frecuencia de datos deseados. Para la posterior ejecución, se han tomado los datos cada 15 minutos del año 2021, que es el que se ha desarrollado en el caso que se estudia a continuación. Para terminar con este paso, se han añadido a la variable *energysystem* estos 3 elementos.

Lo siguiente ha consistido en dar la posibilidad al programa de que exista una batería general para todo el sistema, para que en caso de que el usuario lo desee se instale, a parte de las posibles baterías de cada edificio.

```
print("Existe la posibilidad de ubicar una batería para todo el sistema que "
      "almacene los posibles excedentes de todos los edificios.\n")
print("Seleccione 1 en caso de querer colocar esta batería, o seleccione 2 en"
      " caso de no querer disponer de ella.\n")
batgen=int(input("Selección: "))
print("-------------------------------------------------------------------------"
      "---------------------")

if batgen==1:
    print("Indique los valores de los siguientes parámetros de la batería:")
    cap=float(input("Capacidad nominal (kWh): "))
    Pcarga=float(input("Potencia de carga (kW): "))
    Pdescarga=float(input("Potencia de descarga (kW): "))
    BateriaGeneral=FuncionesSolph.BateriaGen(BusConexion,cap,Pcarga,Pdescarga)
    energysystem.add(BateriaGeneral)
```

Figura 8.6. *Definición batería sistema general*

En la Figura 8.6 se comprueba lo que el programa pide al ejecutarlo. El usuario debe pulsar el número 1 en caso de querer instalarla, y el número 2 en caso de no querer hacerlo. Si el número aportado al programa es el 1, se ejecuta la sentencia siguiente que supone la definición de la batería general y su introducción en el sistema. La conexión en este caso es también sólo con el bus de conexión, pero en este caso es de dos direcciones. La batería puede aportar energía al sistema en caso de requerirla, pero también el sistema puede almacenar energía en la batería en caso de existir un excedente. Se solicita además al usuario que indique los parámetros de capacidad nominal, potencia de carga y potencia de descarga, representados por las variables *cap*, *Pcarga* y *Pdescarga*.

Ya se encuentran definidos todos los elementos independientes del número de edificios de los que dispone el sistema, por lo que a continuación se ha procedido a definir los elementos correspondientes a cada edificio. Lo primero que se ha definido han sido una serie de variables para guardar todos los datos, con el objetivo de reducir el número de variables y simplificar el proceso.

```
op=1
a=1
Bus=dict()
Generacion=dict()
Consumo=dict()
TrafoIn=dict()
TrafoOut=dict()
Bateria=dict()
bat=dict()
DatosPotencia=pd.DataFrame()
```

Figura 8.7. *Variables conjuntas de los edificios*

Como se ve en la Figura 8.7, al factor *a* se le ha dado ahora un valor de 1, que es el que va indicado en el primer edificio que se seleccione. Este factor se incrementa en 1 cada vez que se define un edificio, como se ha mostrado posteriormente en el trabajo. Se ha generado una variable denominada *op* que es la que indica el tipo de edificio que se genera. El resto de las variables se han empleado para guardar los datos de cada edificio como se ha mencionado.

Una vez definidas estas variables, se ha comenzado a la definición de los edificios. Se ha explicado en el apartado 7.1. que existe la posibilidad de introducir 3 tipos de edificios al sistema, y como el número de edificios es el número que el usuario seleccione, es necesario ejecutar un bucle *while*, para que el programa se continúe ejecutando hasta que se le indique que termine de ejecutar ese bucle, cuando la definición de los edificios haya terminado. Esto se ha hecho como se observa en la Figura 8.8.

```
while op != 4: #Se ejecuta mientras op sea diferente de 4
    print('1.Añadir edificio residencial\n2.Añadir edificio con horario '
          'laboral?'
          '(colegios,edificios de oficinas...)\n3.Añadir edificio con horario '
          'predominante el fin de semana\n4.Selección de edificios terminada')
    #Muestra las opciones
    op = int(input('Ingresa una opcion: ')) # Usuario ingresa opcion
    print("------------------------------------------------------------------------"
          "-------------------------")
```

Figura 8.8. *Selección de los edificios*

El programa indica al usuario el tipo de edificios que puede seleccionar, como se ve en la imagen. Además, aporta la opción número 4 que se trata de la sentencia para que la definición de edificios finalice. Dentro del bucle *while,* una vez se haya definido el tipo de edificio, es el momento de definirlos. Una vez terminada esa definición, el programa vuelve a ejecutar el bucle.

La primera de las opciones corresponde a un edificio de viviendas, que corresponde con el tipo h0. Su desarrollo viene definido en la Figura 8.9.

```
    if op == 1:
        print('¿Desea instalar un sistema de almacenamiento en el edificio '
              'seleccionado?')
        print('Seleccione una de las siguientes opciones:')
        print('1->Edificio con almacenamiento')
        print('2->Edificio sin almacenamiento')
        bat[a]=int(input('Indica la opción seleccionada: '))
        n=int(input(
            "Indica el número de viviendas que contiene el edificio: "))

        #Obtener los datos de potencia de este sistema
        sistema,tiempo,latitud,longitud=FeedIn.IrradianciaSistemaFotovoltaico(
            Mes1,Dia1,Mes2,Dia2,Año1,Año2)
        Potencia=FeedIn.PotenciaEnkW(tiempo,latitud,longitud,sistema,a)
        PotenciaData=pd.DataFrame(Potencia)
        DatosPotencia[a]=PotenciaData

        #Ahora se va a crear el sistema
        Bus[a]=FuncionesSolph.Buses(a)
        Generacion[a]=FuncionesSolph.PlacasSolares(Bus[a],DatosPotencia[a],a)
        Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a], Demanda15min.h0*n,a)
        TrafoIn[a]=FuncionesSolph.Conexion(Bus[a], BusConexion,a,"IN")
        TrafoOut[a]=FuncionesSolph.Conexion(BusConexion, Bus[a],a,"OUT")
        energysystem.add(Bus[a],Generacion[a],Consumo[a],
                         TrafoIn[a],TrafoOut[a])

        if bat[a]==1:
            print("Indique los valores de los siguientes parámetros"
                  "de la batería:")
            cap=float(input("Capacidad nominal (kWh): "))
            Pcarga=float(input("Potencia de carga (kW): "))
            Pdescarga=float(input("Potencia de descarga (kW): "))
            Bateria[a]=FuncionesSolph.Bateria(Bus[a],a,cap,Pcarga,Pdescarga)
            energysystem.add(Bateria[a])

        a+=1
```

Figura 8.9. *Definición edificio residencial*

Lo primero que tiene lugar es la decisión de instalar baterías en dicho edificio. El programa pregunta al usuario por ello, debiendo indicar la opción 1 en caso de querer instalarlo y la opción 2 en caso de no querer hacerlo. Dicho parámetro se ha guardado en la variable *bat*, una de las definidas previo a la definición de los edificios, para después dependiendo de ese parámetro introducir el sistema de baterías o no hacerlo.

El siguiente paso ha consistido en obtener los datos de demanda del edificio, pero la demanda ya ha sido definida previamente, ya que existían ciertos parámetros de esa función como eran la fecha de inicio y los periodos que se han empleado ya. Además, la función *Demanda* no depende del tipo de edificio, al ejecutarla realiza el cálculo de la demanda para los 3 tipos de edificios, por lo que para introducirla es suficiente con indicarle cuál de las 3 columnas se quiere emplear.

Lo único que queda por definir de la demanda es el parámetro que se ha determinado para cuantificar el tamaño del edificio. En el caso del edificio residencial, como se ha explicado en el apartado 7.1., el parámetro por el que se multiplica es el número de viviendas de las que consta, por lo que lo que se ha hecho es indicarle al programa que pregunte al usuario por ello, guardando

ese valor en una variable que posteriormente se ha multiplicado por la demanda obtenida de la función *Demanda*.

El siguiente paso ha consistido en definir los datos de potencia. Primero se han calculado los datos meteorológicos del lugar donde se ubica mediante la función *IrradianciaSistemaFotovoltaico*. Los parámetros de entrada que representan la fecha de inicio y de fin del cálculo, se han definido al ejecutar la función *Demanda*. A la salida se han obtenido los datos de ubicación del sistema, así como los datos meteorológicos. Con esos datos se ha entrado en la función *PotenciaEnkW,* con la que se consigue la potencia en las unidades de kW que se ha generado con el sistema indicado. Para poder disponer de todos los datos de potencia de todos los edificios en una misma variable, lo cual facilita a posterior la resolución y optimización del problema, se ha convertido esos datos al tipo *DataFrame* y se han guardado en la variable *DatosPotencia*. Ya se tienen definidos los datos de demanda y potencia generada, por lo que lo que continúa ha sido definir los elementos del sistema.

Como se observa en la Figura 8.9, se ha comenzado por definir el bus del edificio, mostrado en el gráfico del sistema de la Figura 8.1 como *Electricidad_a*, que es el nombre que obtiene el bus, con el valor de *a* del que se disponga en ese momento en el nombre.

Después se han definido los elementos de demanda y generación. Al elemento de generación se le han introducido los datos de potencia que se han calculado previamente, así como el parámetro *a* para definir su nombre y el bus del edificio para realizar la conexión. Lo mismo se ha hecho con el elemento de demanda, pero aportándole los datos de la variable *Demanda15min*. Para señalar que se trata del caso 1, se ha llamado a la variable como *Demanda15min.h0*. De esta manera se ha obtenido únicamente la columna del tipo h0. Además, como se ha definido, se ha multiplicado los valores de esa columna por el número de viviendas de dicho edificio, para obtener el valor final. Para conectar estos elementos al bus de conexión del sistema se han definido los elementos *TrafoIn* y *TrafoOut* para realizar las conexiones en los dos sentidos. Una vez está todo definido, se han añadido estos elementos al sistema, añadiéndolos a la variable *energysystem*.

Queda para finalizar introducir en el sistema de baterías en caso de querer instalarlo. Para ello, si la variable en la que se ha guardado la selección es 1, se genera dicho elemento, se pregunta por las variables ya mencionadas de sus características y se introduce al sistema. En caso de no serlo, no se introduce el sistema de baterías. Las variables empleadas para los parámetros técnicos (*cap*, *Pcarga* y *Pdescarga*) son las mismas que para la batería general, y en el resto de los casos también se hace de la misma manera, ya que no es necesario almacenar esos valores por separado por lo que se actualiza el valor cada vez que se solicita y se introduce en la batería generada.

Por último, se ha incrementado en 1 el valor *a* que ya se ha explicado que se emplea para dar nombre a cada edificio, por lo que de esa manera se tiene una diferenciación en los nombres.

Para el caso de que se trate de un edificio con horario laboral o un edificio con horario predominante el fin de semana, la definición es similar entre ellas, por lo que se ha agrupado en una misma sentencia condicional, como aparece definido en la Figura 8.10.

```
    elif op == 2 or op==3:
        print('¿Desea instalar un sistema de almacenamiento en el edificio '
              'seleccionado?')
        print('Seleccione una de las siguientes opciones:')
        print('1->Edificio con almacenamiento')
        print('2->Edificio sin almacenamiento')
        bat[a]=int(input('Indica la opción seleccionada: '))
        m2=int(input("Indica los metros cuadrados útiles de los que dispone el"
                     " edificio: "))

        #Obtener los datos de potencia de este sistema
        sistema,tiempo,latitud,longitud=FeedIn.IrradianciaSistemaFotovoltaico(
            Mes1,Dia1,Mes2,Dia2,Año1,Año2)
        Potencia=FeedIn.PotenciaEnkW(tiempo,latitud,longitud,sistema,a)
        PotenciaData=pd.DataFrame(Potencia)
        DatosPotencia[a]=PotenciaData
        #Ahora se va a crear el sistema
        Bus[a]=FuncionesSolph.Buses(a)
        Generacion[a]=FuncionesSolph.PlacasSolares(Bus[a],DatosPotencia[a],a)
        if op==2:
            Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a],
                                                      Demanda15min.g0*m2,a)
        elif op==3:
            Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a],
                                                      Demanda15min.g6*m2,a)
        TrafoIn[a]=FuncionesSolph.Conexion(Bus[a], BusConexion,a,"IN")
        TrafoOut[a]=FuncionesSolph.Conexion(BusConexion, Bus[a],a,"OUT")
        energysystem.add(Bus[a],Generacion[a],Consumo[a],
                         TrafoIn[a],TrafoOut[a])
        if bat[a]==1:
            print("Indique los valores de los siguientes parámetros "
                  "de la batería:")
            cap=float(input("Capacidad nominal (kWh): "))
            Pcarga=float(input("Potencia de carga (kW): "))
            Pdescarga=float(input("Potencia de descarga (kW): "))
            Bateria[a]=FuncionesSolph.Bateria(Bus[a],a,cap,Pcarga,Pdescarga)
            energysystem.add(Bateria[a])

        a+=1
```

Figura 8.10. *Definición edificios laborales*

La definición de estos sistemas es muy parecida al caso anterior. Los únicos parámetros que han cambiado son los referidos a la demanda. En este caso, el parámetro que se ha empleado para cuantificar el tamaño del edificio son los metros cuadrados útiles, por lo que se ha preguntado por ello en caso de que las opciones sean la 2 o la 3.

En caso de que el edificio sea aquel cuya actividad se da en horario laboral, al introducir la demanda en la variable *Consumo*, se ha llamado a la columna g0 de la variable *Demanda* y se ha multiplicado por el valor de los metros cuadrados. En el caso de que se trate de uno con horario predominante el fin de semana, se ha realizado la misma multiplicación, pero por la columna g6.

Ya se tienen definidas las opciones con las que se generan los sistemas de los edificios. Solo queda por definir la última de las opciones, que es la número 4, que supone la finalización del diseño y el comienzo al cálculo de la solución. El bucle se deja de ejecutar y se pasa al siguiente paso. Esto se hace como se ve en la Figura 8.11, dando un mensaje al usuario de que ha finalizado la definición y comienza el programa a solucionar el problema.

```
    elif op == 4:
        print("-------------------------------------------------------------------"
              "-----------------------------")
        print('Diseño del sistema completado, se procede a realizar los '
              'cálculos')
        print("-------------------------------------------------------------------"
              "-----------------------------")
```

Figura 8.11. *Finalización del bucle de definición del sistema*

Se ha comenzado con la optimización y solución del sistema, como se aprecia en la Figura 8.12. Primero se ha resuelto el sistema con el solucionador instalado, que se trata del CBC. A continuación, se han solucionado tanto el *main*, que contiene la solución de cada elemento por separado, como el meta que ha solucionado el sistema como conjunto. Se han restaurado también los resultados como se indica en la siguiente imagen.

```
model = solph.Model(energysystem)
model.solve(solver="cbc", solve_kwargs={'tee': True})

energysystem.results["main"] = solph.processing.results(model)
energysystem.results["meta"] = solph.processing.meta_results(model)
energysystem.dump(dpath=None, filename=None)
energysystem = solph.EnergySystem()
energysystem.restore(dpath=None, filename=None)
```

Figura 8.12. *Solución del problema*

Una vez realizado estos pasos, se ha realizado la definición de los resultados que se quieren representar y las gráficas que se desean obtener. Para ello, primero se han definido una serie de variables en la Figura 8.13:

```
results = energysystem.results["main"]
i=1
e=0
results_baterias=dict()
custom_baterias=dict()
electricity_bus=dict()
```

Figura 8.13. *Variables para representar las soluciones*

Se ha guardado en la variable *results* los resultados del main, que son los resultados de cada elemento por separado. Además, se han inicializado las variables *e* y *i*, que a continuación se muestra para qué se emplean. El resto de las variables tienen como función el almacenamiento de los datos.

Para empezar, se ha procedido a la definición de los resultados de los diferentes buses. Como el bus conexión tiene como índice para definir su nombre el valor 0, en este caso se ha partido desde el valor de *e*=0 hasta el valor final de *a*, que queda fijo una vez terminado el bucle de definición de los edificios del sistema. Primero se han obtenido los resultados de cada uno de los buses y después se han imprimido sus resultados, como se hace en la Figura 8.14.

Lo siguiente ha sido realizar las gráficas de dichos buses. Se han indicado una serie de parámetros para cuadrar correctamente los gráficos y para tener una representación limpia y clara de ellos. Se le ha indicado finalmente la función *plt.show* con la que se ha conseguido que se muestren esos gráficos.

Por último, se ha incrementado en 1 el valor de *e* para que el bucle se realice hasta que ya no existan más buses para representar. Cabe destacar que se ha indicado que *e* tiene que ser menor que *a*, ya que el valor final de *a* es una unidad superior al número de edificios, ya que, al ejecutar el bucle de definición de edificios, se ha incrementado siempre en 1 el valor de a, y al llegar al paso final se ha incrementado ese valor, pero no se llega a definir más edificios, por lo que en este apartado es importante indicarle que tiene que ser menor el valor de *e*, en vez de menor o igual que *a*.

```
while e<a:
    electricity_bus[e] = solph.views.node(results, f"Electricidad_{e}")
    print(electricity_bus[e]["sequences"].sum(axis=0))
    if plt is not None:
        fig, ax = plt.subplots(figsize=(10,5))
        electricity_bus[e]["sequences"].plot( ylabel=
        "Flujo energía(kWh)",xlabel="Periodo",
        ax=ax, kind="line", drawstyle="steps-post")
        plt.legend(
        loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
       1.3), ncol=2)

        fig.subplots_adjust(top=0.8)
        plt.show()
    e+=1
```

Figura 8.14. *Resultados y gráficos de los buses*

El mismo proceso que se ha realizado para los buses tiene lugar para las baterías, pero en este caso se ha empezado desde el valor de *i*=1, ya que no existe batería 0, ya que a la batería del sistema general se le ha denominado con otro nombre. Por ello se han inicializado dos variables, *i* y *e*, para emplear una para cada bucle.

El resto de las sentencias son las mismas con los cambios correspondientes para indicar que se representen las soluciones y los gráficos de las baterías, como se muestra en la Figura 8.15, con la única excepción de introducir una sentencia condicional *if* para comprobar si existen baterías en dicho edificio, para lo que ha resultado muy útil la variable *bat* definida previamente. Si el valor es 1 se procede a su representación.

```
        while i<a:
            if bat[i]==1:
                results_baterias[i]=energysystem.groups[f"bateria_{i}"]
                print(results[(results_baterias[i], None)]["sequences"])
                print("")
                custom_baterias[i] = solph.views.node(results, f"bateria_{i}")
                if plt is not None:
                    fig, ax = plt.subplots(figsize=(10, 5))
                    custom_baterias[i]["sequences"].plot( ylabel=
                    "Flujo energía(kWh)",xlabel="Periodo",
                    ax=ax, kind="line", drawstyle="steps-post")
                    plt.legend(
                    loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
                   1.3), ncol=2)
                    fig.subplots_adjust(top=0.8)
                    plt.show()

            i+=1
```

Figura 8.15. *Resultados y gráficos de las baterías*

El mismo proceso de las baterías de los edificios ha tenido lugar con las baterías del sistema general, mostrado en la Figura 8.16, en caso de disponer de dichas baterías, comprobando si la variable *batgen* tiene el valor 1.

```
        if batgen==1:
            results_bateriageneral=energysystem.groups["BateriaGeneral"]
            print(results[(results_bateriageneral, None)]["sequences"])
            print("")
            custom_bateriageneral = solph.views.node(results, "BateriaGeneral")
            if plt is not None:
                fig, ax = plt.subplots(figsize=(10, 5))
                custom_bateriageneral["sequences"].plot( ylabel=
                "Flujo energía(kWh)",xlabel="Periodo",
                ax=ax, kind="line", drawstyle="steps-post")
                plt.legend(
                loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
               1.3), ncol=2)
                fig.subplots_adjust(top=0.8)
                plt.show()
```

Figura 8.16. *Resultados y gráfico de la batería general*

Se ha finalizado con la solución del sistema y con los respectivas representaciones de los resultados de cada elemento, así como de sus gráficos, por lo que se ha comenzado a realizar diferentes pruebas con el programa y se ha comprobado como varían los resultados dependiendo de los parámetros que se disponga en cada sistema.

9. ANÁLISIS DE LOS RESULTADOS

En este apartado se lleva a cabo a cabo un análisis de las funciones del programa desarrollado mediante el análisis de diferentes situaciones posibles y la comparación de los resultados obtenidos de ellas.

Es importante destacar que se pueden realizar múltiples estudios cambiando los parámetros que se tienen que introducir. Debido a ello, el análisis se ha limitado solo a algunos de ellos, con el objetivo de verificar las funcionalidades del programa y analizar las posibilidades que ofrece.

9.1. Caso sin almacenamiento

En esta sección se estudian dos situaciones sin introducir los sistemas de almacenamiento. Estas dos situaciones son las correspondientes a dos meses diferentes del año 2022, los meses de enero y agosto.

Ciertas variables se han definido como constantes para los dos casos. Como se ha explicado, existen muchas variedades de estudio, pero en este caso se ha buscado comparar lo que sucede en dos meses con condiciones meteorológicas y generaciones diferentes. Para la ubicación de la instalación se ha tomado la ciudad de Bilbao, cuyas coordenadas geográficas de latitud y longitud son 43,263 y -2,925 respectivamente. La orientación de los paneles fotovoltaicos ha sido de 180 grados en todos los casos.

Se han introducido 3 edificios, siendo el número 1 el edificio de viviendas, el número 2 el laboral con horario entre semana y el número 3 el laboral con horario predominante el fin de semana. En el caso del edificio de viviendas se ha seleccionado uno con 8 viviendas y 10 módulos fotovoltaicos y en el caso de los edificios de oficinas se ha tomado 400 como valor para los metros cuadrados útiles y los módulos instalados son 15.

A continuación, se muestran las gráficas obtenidas. Se ha comenzado por las de la potencia generada en kW en cada uno de los casos. Las situadas a la izquierda corresponden al mes de enero y las de la derecha al mes de agosto.

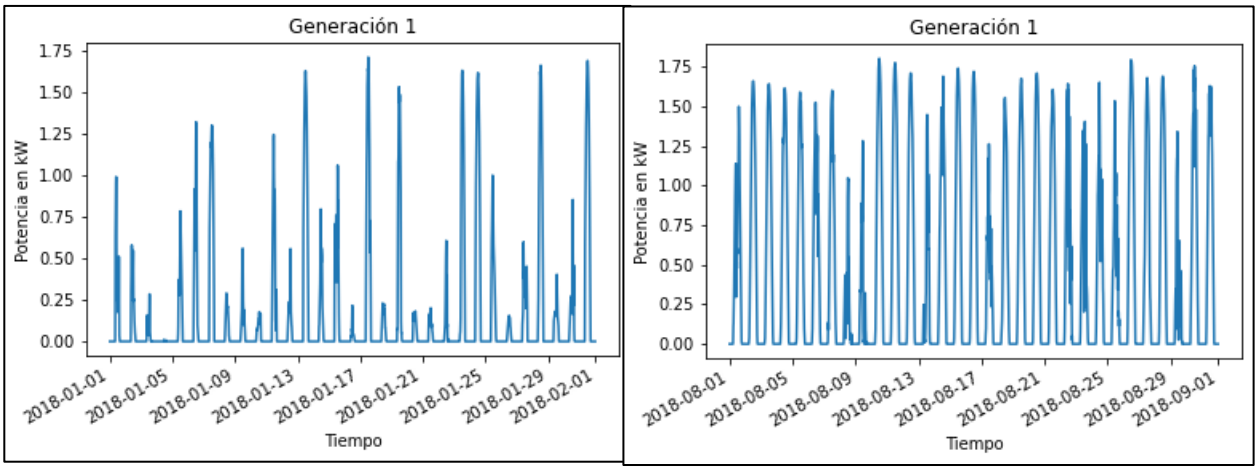

Figura 9.1. *Generación del edificio residencial*

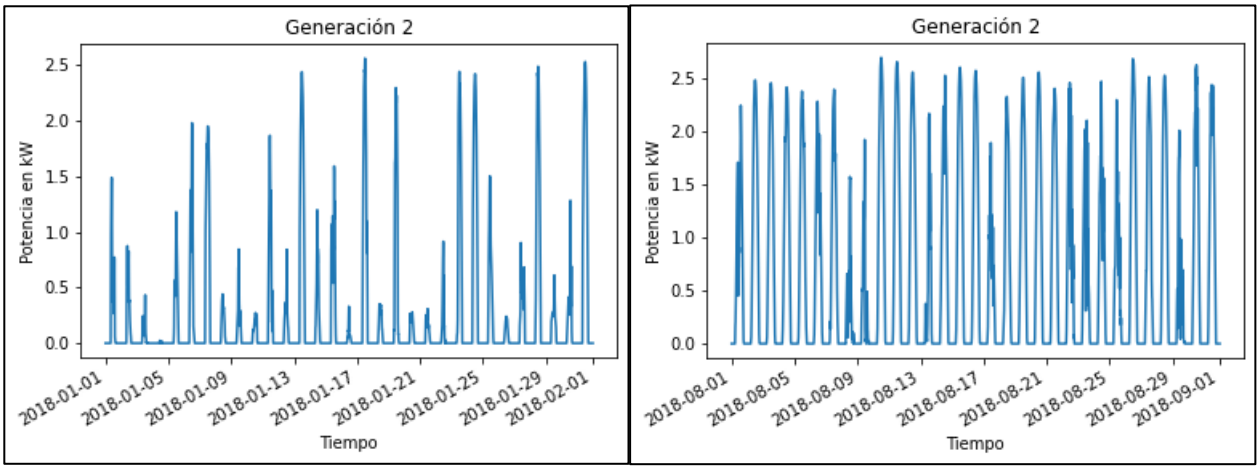

Figura 9.2. *Generación del edificio laboral con horario entre semana*

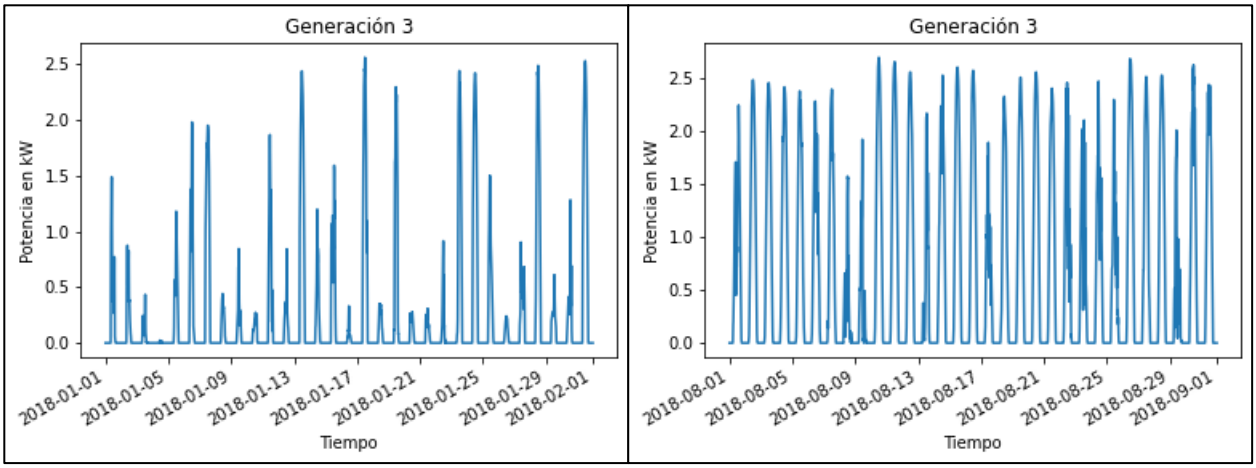

Figura 9.3. *Generación del edificio laboral con horario predominante en fin de semana*

Se observa en las gráficas anteriores como varía la generación dependiendo del mes en el que tenga lugar el estudio. En todos los edificios en el caso del mes de agosto (derecha) la generación ha sido más constante, teniendo potencias parecidas en la mayoría de los días, mientras que en el caso de enero (izquierda) ha sido más variable. Esto es debido a la diferencia de condiciones meteorológicas de cada mes, siendo enero un mes con menos horas de sol y con más días nublados, por lo que se ha comprobado que los resultados tienen sentido con lo que sucede en la realidad.

Las fechas que aparecen en el eje X son incorrectas respecto al año, por la limitación del programa explicada en el apartado 7.2. El año mostrado no corresponde con el del estudio, que es el año 2022, aunque el mes y el día son correctos.

También se aprecia que la generación en los edificios laborales es mayor, ya que en ellos se ha decidido instalar más módulos que en el caso del edificio residencial. A su vez, la generación ha sido la misma en los dos edificios laborales, ya que se han tomado los mismos datos de entrada, como lo son la orientación, el número de módulos o la ubicación.

La muestra de estas figuras se ha realizado para ver las diferencias existentes al variar la meteorología, aunque en apartados posteriores no se han mostrado para centrarse en otras más interesantes como son las de los flujos de cada edificio o las de resumen general de ellos, que se muestran a continuación.

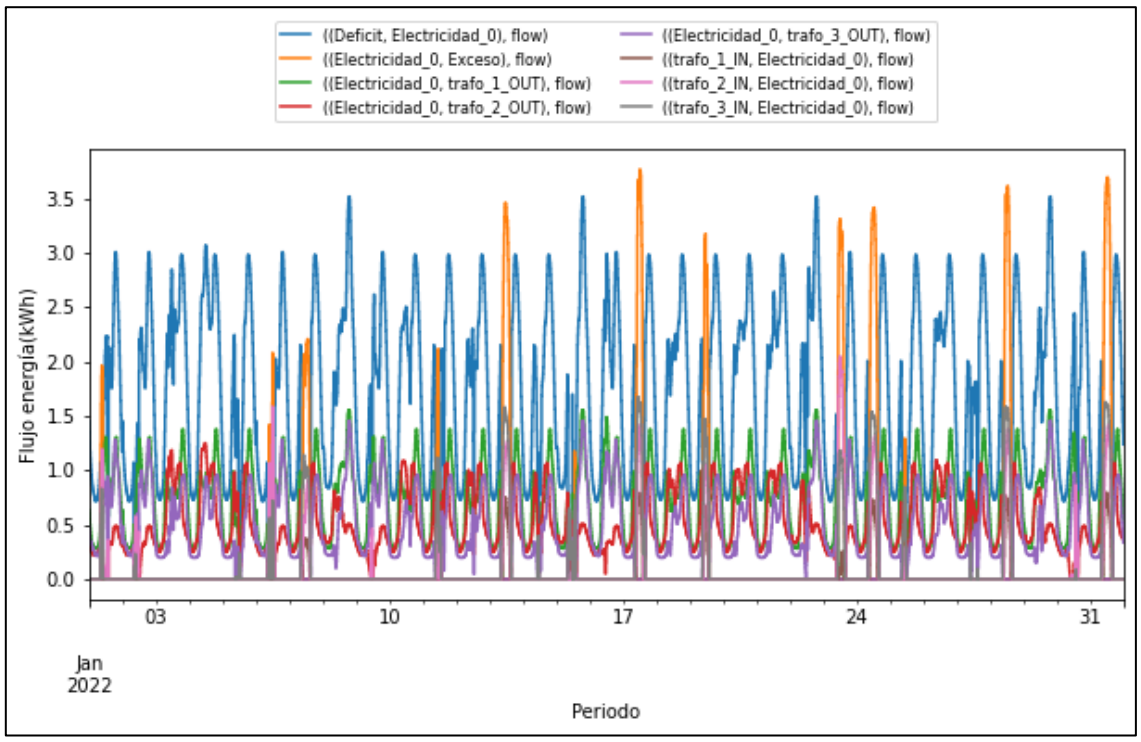

Figura 9.4. *Flujo del bus de conexión en el mes de enero*

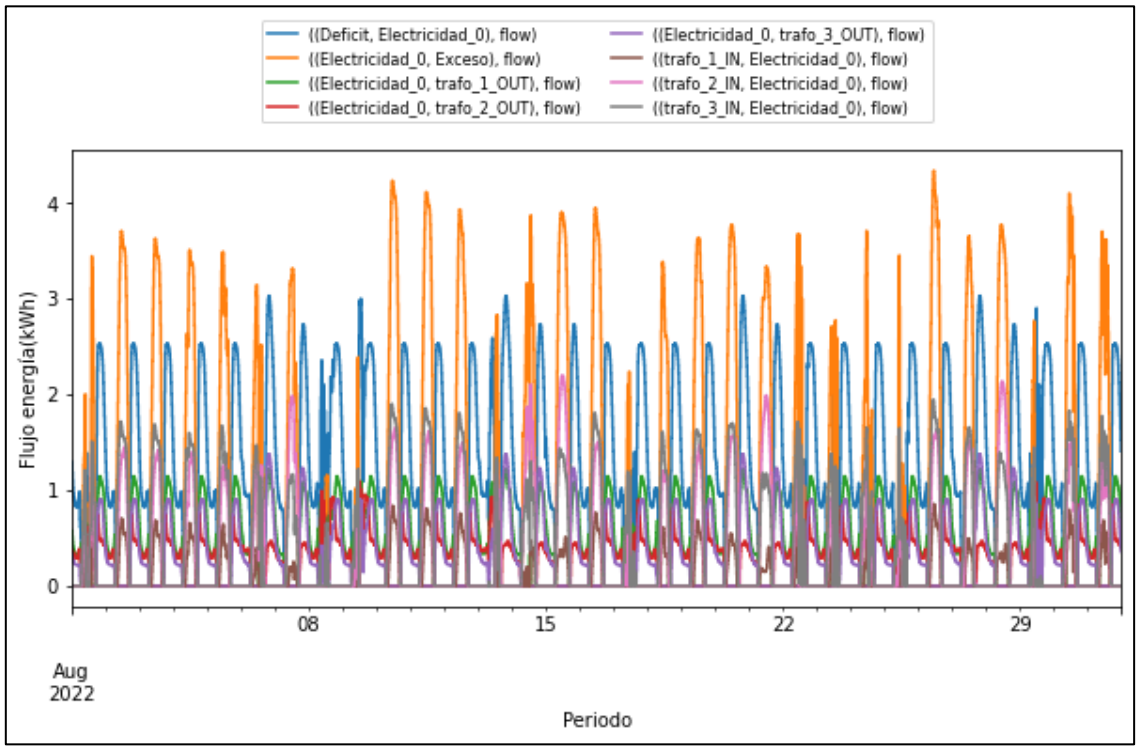

Figura 9.5. *Flujo del bus de conexión en el mes de agosto*

Se han obtenido gran cantidad de datos con la solución del programa y con estas gráficas. Todas ellas, incluidas las homólogas de los siguientes apartados, representan curvas de flujo. Los nombres que se han indicado en la parte superior indican desde donde va el flujo, que corresponde con el nombre indicado a la izquierda, y hacia dónde va, representado por el nombre de la derecha. En el caso de la curva "((Deficit,Electricidad_0),Flow)", el flujo va desde la red hasta el bus de conexión, es decir, la compra de energía a la red.

En el mes de enero (Figura 9.4), la curva correspondiente al déficit del sistema (azul), es decir, la compra de energía a la red ha sido la que predomina respecto a la curva de exceso (naranja), que es la de venta de energía a la red. En el mes de agosto (Figura 9.5), sucede lo contrario, la curva predominante ha sido la del exceso. Esto ocurre debido a lo explicado anteriormente respecto a la generación en función del mes

A pesar de tener mayor exceso que déficit en agosto, se ha comprobado como ambas curvas varían constantemente, teniendo muchos momentos que ha pasado de exceso a déficit y viceversa. Esto es algo que se ha estudiado en los casos posteriores, ya que la introducción de sistemas de baterías implica almacenar la energía durante los momentos de exceso, y emplearla para los momentos en lo que hay déficit, limitando los flujos con la red.

El resto de las curvas representan los flujos entre el bus de conexión y los buses de los edificios, que se representan a su vez en las siguientes figuras.

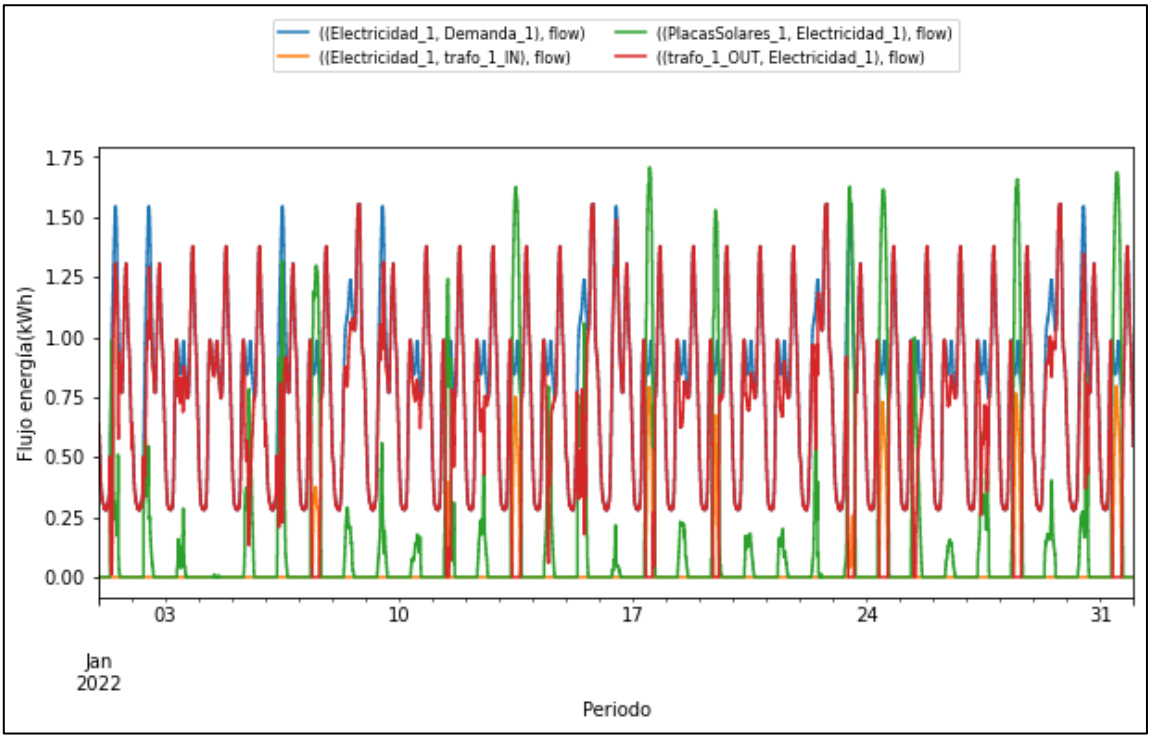

Figura 9.6. *Flujo del edificio residencial en enero*

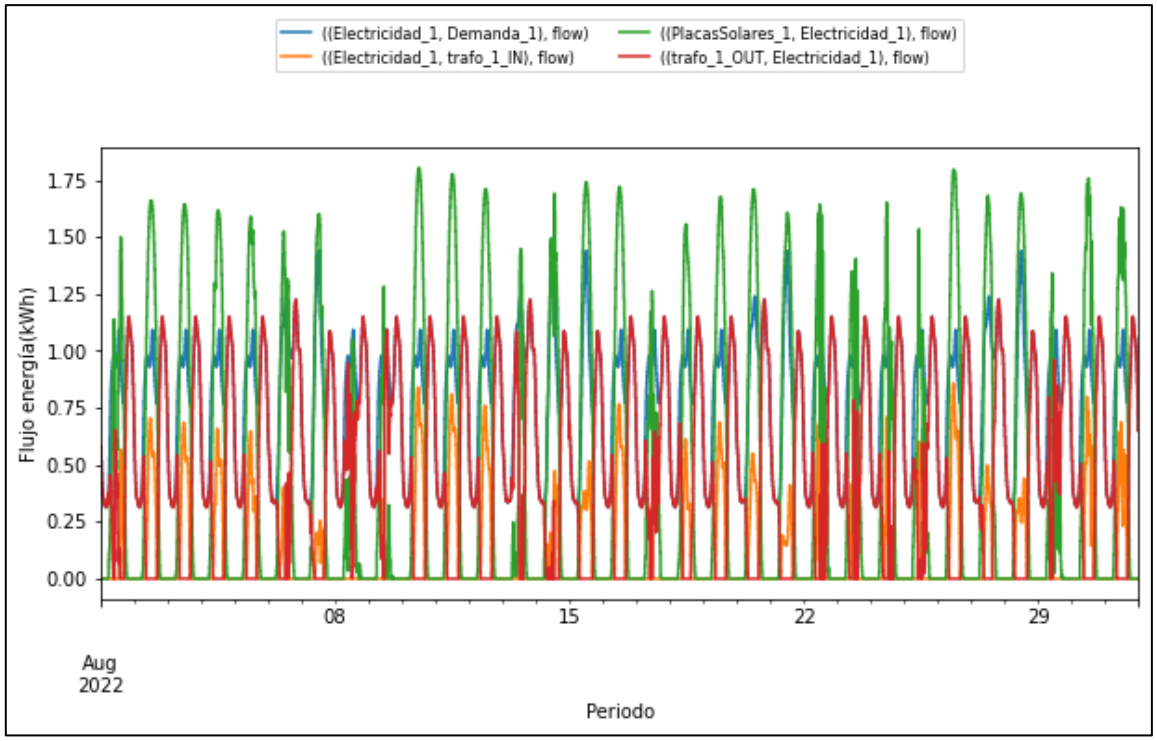

Figura 9.7. *Flujo del edificio residencial en agosto*

En estas figuras se tienen las curvas de la demanda de cada edificio (azul), las de la generación de la instalación fotovoltaica (verde) y las del intercambio de energía con el bus de conexión, siendo la de color naranja la que representa el flujo desde el bus del edificio residencial hasta el bus de conexión, y la de color rojo la que tiene la dirección opuesta.

En este caso se puede observar como la demanda tiene más picos en enero, y la generación es mayor en agosto, como se ha explicado previamente. El flujo hacia el bus de conexión ha sido muy limitado en el primer caso, ya que no se ha contado con mucha energía para poder ofrecer al sistema completo, mientras que la energía que se ha solicitado al sistema general ha sido mayor ya que hay mayor ausencia de ella.

En el caso del mes de agosto, la curva del flujo hacia la conexión general ha sido mayor, ya que se ha dispuesto de más energía generada, pero aun así la curva de energía solicitada ha sido también grande, a pesar de ser menor que en el primer caso, ya que como se ha mencionado, los flujos entre el edificio y la conexión han sido muy variables, al no disponer de sistemas del almacenamiento que permitan reservar el excedente para los momentos en los que haya déficit.

Las mismas situaciones se observan en las figuras que se muestran a continuación, que son las de los dos edificios restantes. En estos casos se ha tenido situaciones similares al primer caso, con la diferencia de que la generación ha sido mayor, y por lo tanto la aportación de energía al sistema ha sido también mayor.

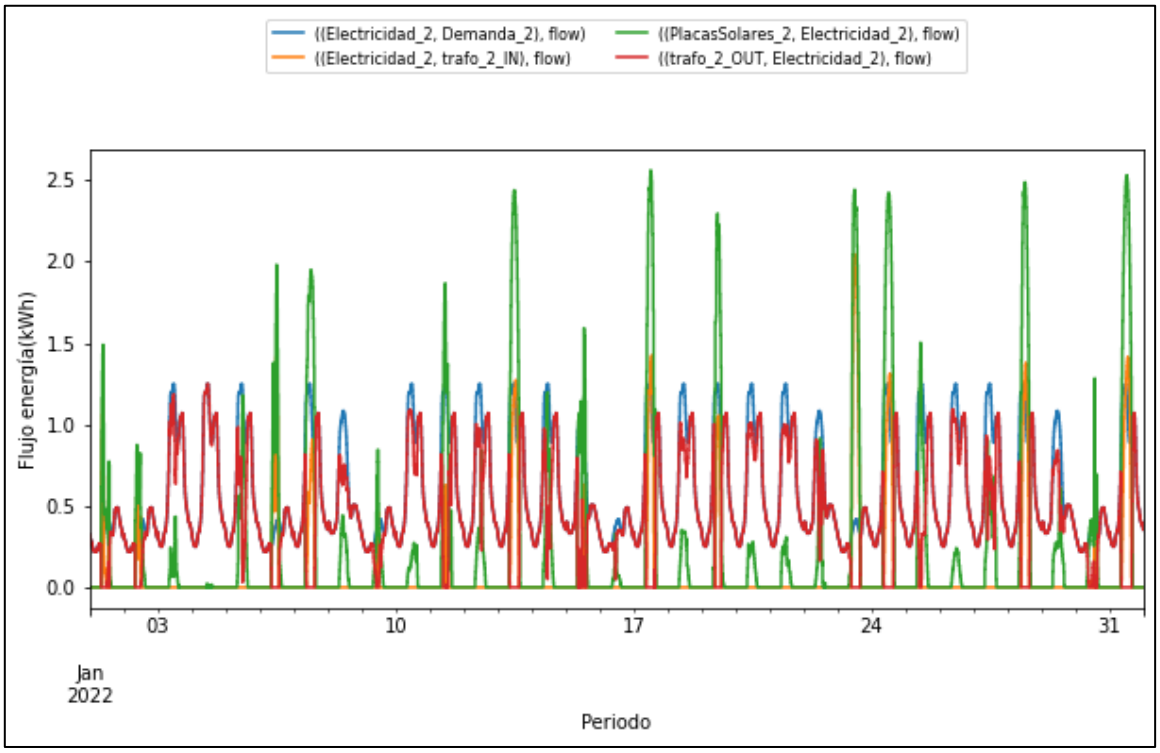

Figura 9.8. *Flujo del edificio entre semana en enero*

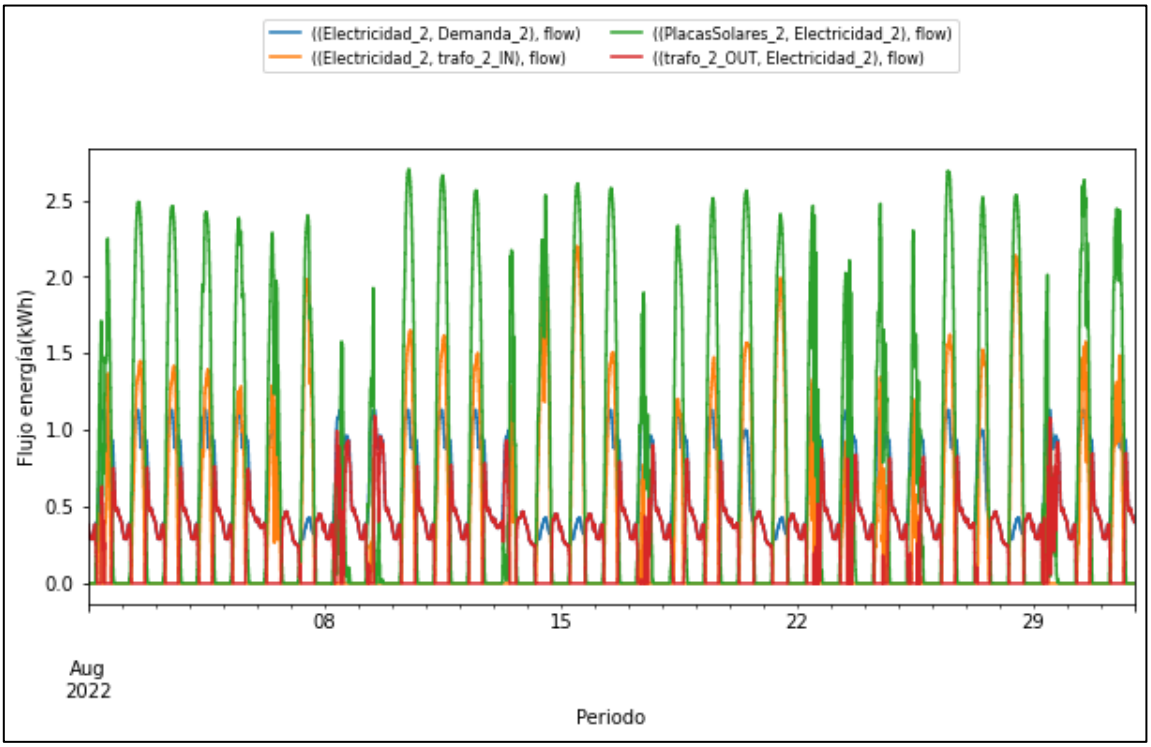

Figura 9.9. *Flujo del edificio con horario entre semana en agosto*

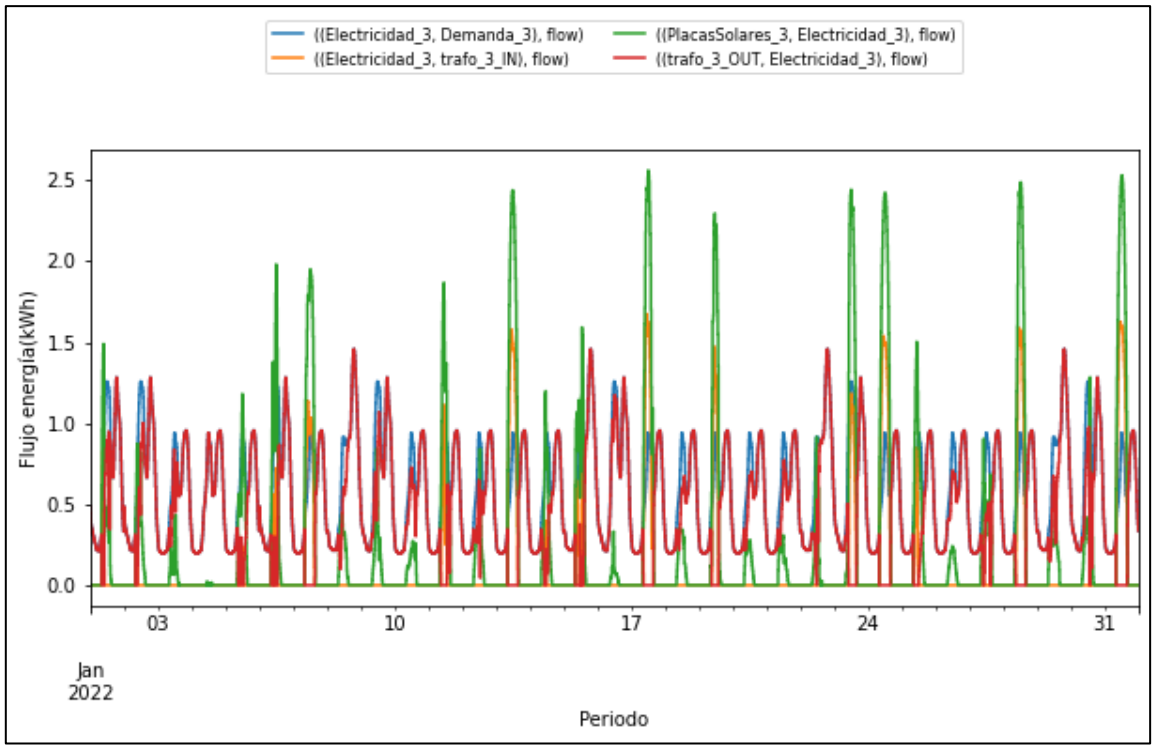

Figura 9.10. *Flujo del edificio con horario de fin de semana en enero*

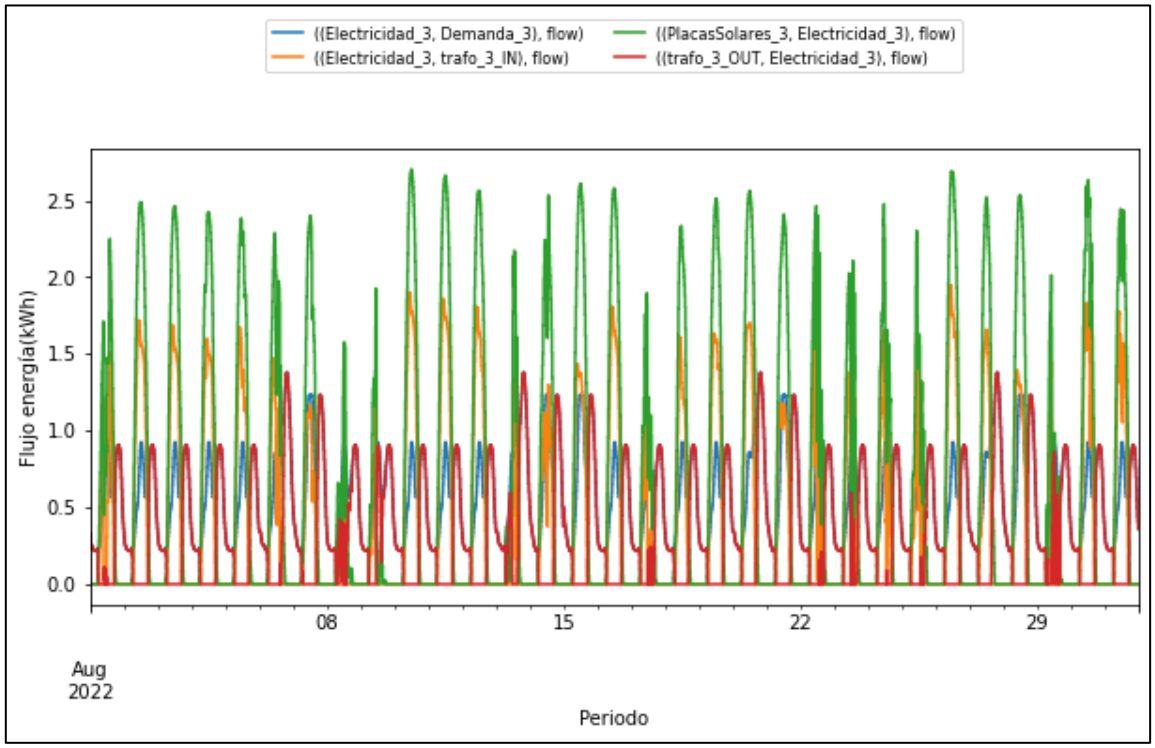

Figura 9.11. *Flujo del edificio con horario de fin de semana en agosto*

Por último, se ha obtenido la Figura 9.12 que representa el resumen general de ambos casos, siendo el caso de enero la parte izquierda y el mes de agosto la parte derecha.

Primero se muestran los datos del bus de conexión y los diferentes flujos que tienen lugar, y a continuación los correspondientes a cada uno de los edificios. La dirección del flujo es desde la variable situada a la izquierda hasta la variable situada a la derecha. Por ejemplo, en el caso de la sentencia "((Déficit, Electricidad_0,flow)" el flujo es desde la red hasta el bus de conexión, es decir, la compra de energía a la red, de la misma manera que se representa en las gráficas.

```
((Deficit, Electricidad_0), flow)        4693.628172
((Electricidad_0, Exceso), flow)          536.020425
((Electricidad_0, trafo_1_OUT), flow)    2010.484355
((Electricidad_0, trafo_2_OUT), flow)    1336.119361
((Electricidad_0, trafo_3_OUT), flow)    1387.589362
((trafo_1_IN, Electricidad_0), flow)       76.703776
((trafo_2_IN, Electricidad_0), flow)      224.224992
((trafo_3_IN, Electricidad_0), flow)      275.656563
dtype: float64
((Electricidad_1, Demanda_1), flow)          2377.588963
((Electricidad_1, trafo_1_IN), flow)           76.703776
((PlacasSolares_1, Electricidad_1), flow)     443.808382
((trafo_1_OUT, Electricidad_1), flow)        2010.484355
dtype: float64
((Electricidad_2, Demanda_2), flow)          1783.220917
((Electricidad_2, trafo_2_IN), flow)          224.224992
((PlacasSolares_2, Electricidad_2), flow)     671.326549
((trafo_2_OUT, Electricidad_2), flow)        1336.119361
dtype: float64
((Electricidad_3, Demanda_3), flow)          1783.259349
((Electricidad_3, trafo_3_IN), flow)          275.656563
((PlacasSolares_3, Electricidad_3), flow)     671.326549
((trafo_3_OUT, Electricidad_3), flow)        1387.589362
dtype: float64
```

```
((Deficit, Electricidad_0), flow)        2992.421734
((Electricidad_0, Exceso), flow)         2317.050193
((Electricidad_0, trafo_1_OUT), flow)    1341.947918
((Electricidad_0, trafo_2_OUT), flow)     789.828332
((Electricidad_0, trafo_3_OUT), flow)     933.164725
((trafo_1_IN, Electricidad_0), flow)      278.151132
((trafo_2_IN, Electricidad_0), flow)      984.068357
((trafo_3_IN, Electricidad_0), flow)     1127.349946
dtype: float64
((Electricidad_1, Demanda_1), flow)          2377.500959
((Electricidad_1, trafo_1_IN), flow)          278.151132
((PlacasSolares_1, Electricidad_1), flow)    1313.704175
((trafo_1_OUT, Electricidad_1), flow)        1341.947918
dtype: float64
((Electricidad_2, Demanda_2), flow)          1783.175154
((Electricidad_2, trafo_2_IN), flow)          984.068357
((PlacasSolares_2, Electricidad_2), flow)    1977.415177
((trafo_2_OUT, Electricidad_2), flow)         789.828332
dtype: float64
((Electricidad_3, Demanda_3), flow)          1783.229957
((Electricidad_3, trafo_3_IN), flow)         1127.349946
((PlacasSolares_3, Electricidad_3), flow)    1977.415177
((trafo_3_OUT, Electricidad_3), flow)         933.164725
dtype: float64
```

Figura 9.12. *Resumen general caso 1*

En el caso del mes de enero el déficit ha sido muy superior al exceso, ya que por las condiciones explicadas los momentos en los que se tiene energía excedente son muy pocos. En este caso la instalación de baterías no resulta tan efectiva ya que existen pocos momentos en los que se disponga de energía para almacenar. En el caso del mes de agosto el déficit y el exceso han tenido un dato final muy similar, por lo mencionado relacionado a las grandes variaciones que aparecen en las curvas anteriores. Aquí es donde resulta efectivo instalar sistemas de baterías, para evitar la necesidad del flujo de energía con la red en las dos direcciones, lo cual se estudia en los siguientes apartados.

En la comparación de los datos de generación de todos los edificios se ha visto cómo los 3 edificios tienen una mayor generación en el caso del mes de agosto.

9.2. Casos con almacenamiento

En el siguiente apartado se procede a introducir los sistemas de almacenamiento. Se han analizado diferentes situaciones para poder analizarlas y estudiar el funcionamiento del sistema.

Para todos estos apartados se toman ciertos valores constantes para poder realizar correctamente la comparación entre ellos. La ubicación se ha mantenido respecto al primer apartado, tomando las coordenadas geográficas de la ciudad de Bilbao. La orientación de los paneles se ha mantenido también con el valor de 180 grados para todos los casos. El periodo de

cada uno de los estudios ha sido de 1 mes, excepto el último caso, con el objetivo de tener gráficas con menos datos y más claras que permiten realizar un mejor análisis, aunque el estudio es posible realizarlo para periodos mayores.

9.2.1. Instalación de un sistema de baterías en los edificios

Este análisis se ha realizado ubicando sistemas de almacenamiento sólo en los edificios. El periodo en el que se han obtenido los datos es el mes de agosto de 2022. Se ha decidido instalar el mismo sistema que en el apartado anterior para el mes de agosto, es decir, está formado por los 3 edificios seleccionados previamente, teniendo los mismos datos, excepto la introducción de las baterías. De esta manera se ha buscado obtener una comparación respecto al caso anterior y la influencia del cambio introducido.

Se han instalado las mismas baterías para todos los edificios. Las baterías seleccionadas para este proyecto para obtener los diferentes datos necesarios para su desarrollo han sido las Cegasa eBick Ultra175, cuyo catálogo se presenta en la Figura 9.13. Para los siguientes apartados también se ha hecho uso de algunas de las baterías presentadas en él. Se han seleccionado estas ya que, como se recoge en el catálogo, se tratan de baterías autogestionables que solo requieren de enchufarlas y están listas para funcionar [23] . Se emplean para nuevas instalaciones de autoconsumo y los parámetros que tienen son adecuados para los estudios realizados. Además, las diferentes configuraciones que se pueden desarrollar permiten tener diferentes tamaños de baterías lo cual resulta muy práctico para los diferentes casos.

Las opción seleccionada para este apartado ha sido el tipo Ultra 175 48V_280Ah, que dispone de 13,5 kWh de energía nominal y una potencia de carga y descarga de 6,72 kW, que se ha obtenido de multiplicar el voltaje nominal (48V) por la corriente recomendada de carga y la corriente nominal de descarga, ambas de valor 140 A, y se ha dividido entre 1000 para pasar las unidades a kW.

	ULTRA 175 48V_280Ah	ULTRA 175 48V_560Ah	ULTRA 175 *Configuración* 3 x 48V_280Ah	ULTRA 175 *Configuración* 2 x 48V_560Ah
Código producto Con zócalo	109639	109640		
Código producto Sin zócalo	109624	-		
Características mecánicas				
Dimensiones equipo (mm)				
Anchura	765			
Profundidad	405			
Altura	600			
Altura sin zócalo	470	-		
Peso total equipo (Kg	105	210		
Acabado / Cierre de batería	IP30			
Características eléctricas				
Voltaje nominal (V)	48		48	
Voltaje máximo	52,2		52,2	
Voltaje mínimo	43		43	
Capacidad nominal (Ah)	**280**	**560**	**840**	**1120**
Energía nominal (KWh)	13,5	27	40,5	54
Ciclabilidad	> 5000 (80%DOD)			
Tipo de comunicaciones	CAN Bus			
Protecciones eléctricas				
Sobrecarga	ok			
Sobredescarga	ok			
Cortocircuito	ok			
Sobrecorriente	ok			
Sobretemperatura	ok			
Equilibrado pasivo	ok			
Nivel de corrientes (A)				
Corriente máxima de carga continuo	175	320	450	500
Corriente recomendada de carga continuo	140	280	400	475
Corriente nominal de descarga continuo	140	280	400	475
Corriente máxima de descarga continuo	175; (8KW)	340; (15KW)	500 (22,5KW)	575 (26KW)
Corriente/tiempo pico de descarga (1)	225 (5 min); (10KW)	450 (5 min); (20KW)	600 (5 min); (26KW)	800 (5 min); (35KW)
Corriente/tiempo pico de descarga (2)	270 (5seg); (12KW)	540 (5seg); (24KW)	750 (5seg); (32KW)	875 (5seg); (40KW)
Corriente/tiempo pico de descarga (3)	400 (<1seg)	800 (<1seg)	1000 (<1seg)	1000 (<1seg)
Conexiones eléctricas				
Potencia	Conector Rema SR 350 Gris (Se entrega conector similar para instalación con pines para 95mm2) Para instalaciones de más de 2 módulos Ultra se recomienda siempre uso de Busbar (no incluido)			
Comunicaciones				
Conector	RJ45			
Homologaciones				
	Marcado CE			
	UN 38.3			
Accesorios				
109637	TCC CAN para comunicaciones con inversores Victron Sunny Island y Studer			
109642	Cable alargador RJ45 para unir en comunicaciones más de 2 módulos Ultra			

Figura 9.13. *Catálogo Cegasa eBick Ultra175 [23]*

A continuación, se muestran la gráfica del flujo del bus de conexión, que se muestra en la Figura 9.14.

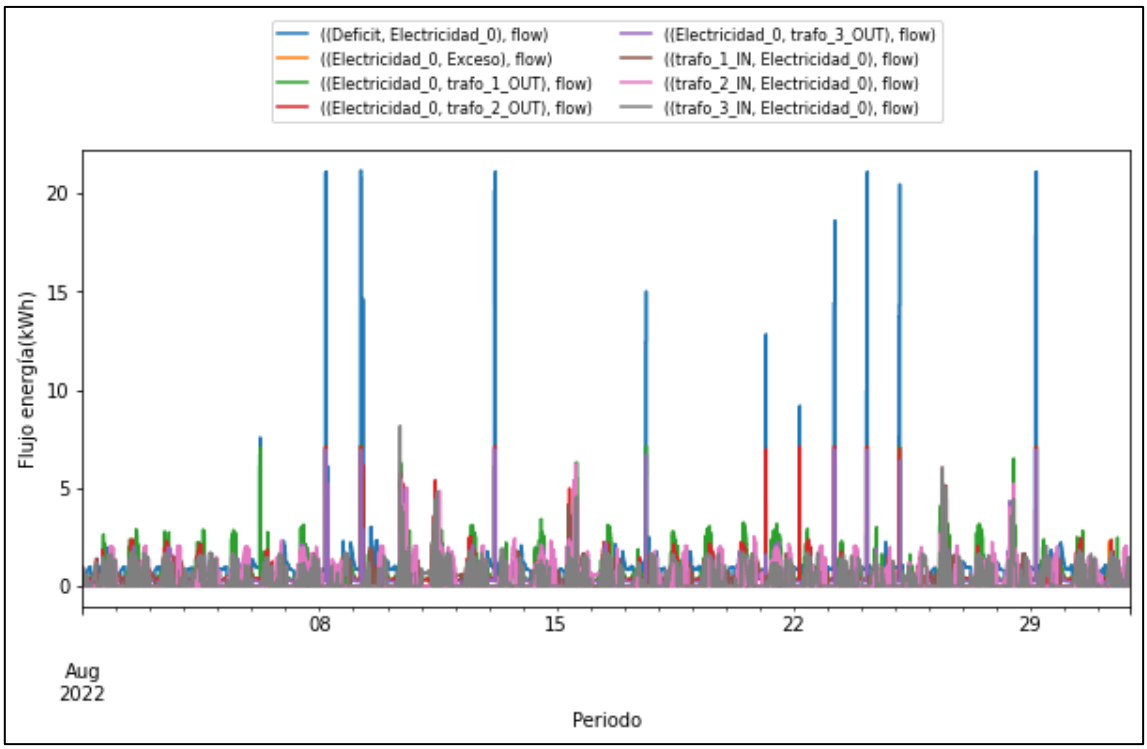

Figura 9.14. *Flujo del bus de conexión caso 2.1.*

Se ha comprobado que el sistema ha requerido de la compra de energía a la red, es decir, la instalación de los sistemas de generación fotovoltaica no ha sido suficiente para cubrir toda la demanda. No ha habido apenas exceso de energía, por lo que la venta de energía a la red es pequeña. Esto ha sido el resultado de la introducción de las baterías. Al poder almacenar energía excedente, se ha reducido la energía comprada y se ha reducido a su vez la venta a la red. En definitiva, se ha limitado la interacción con la red y la dependencia de ella, objetivo principal de las instalaciones introducidas.

Algunas de las curvas han aparecido solapadas y no han ofrecido la mejor visión posible, pero se ha apreciado como la curva verde que es la que indica el flujo desde el bus de conexión hasta el bus del edificio residencial ha sido la que predomina respecto al resto de curvas de solicitud de energía, es decir, el edificio 1 requiere de más energía que los edificios 2 y 3. Han existido periodos breves donde la solicitud tanto del edificio 2 y el edificio 3 han sido mayores, como se observa por ejemplo en la curva roja entre los días 21 y 22, pero la solicitud del edificio ha sido mayor sumando todos los datos del mes ya que es más constante. Ocurre lo mismo, pero al revés en el caso de los flujos de energía aportada al bus de conexión, es decir, los edificios 2 y 3 han aportado más energía para el sistema que el bus 1.

Para observar mejor estas cuestiones relativas a los flujos entre los edificios y el punto de conexión, se han introducido las gráficas de cada edificio, representadas en las Figuras 9.15, 9.16 y 9.17.

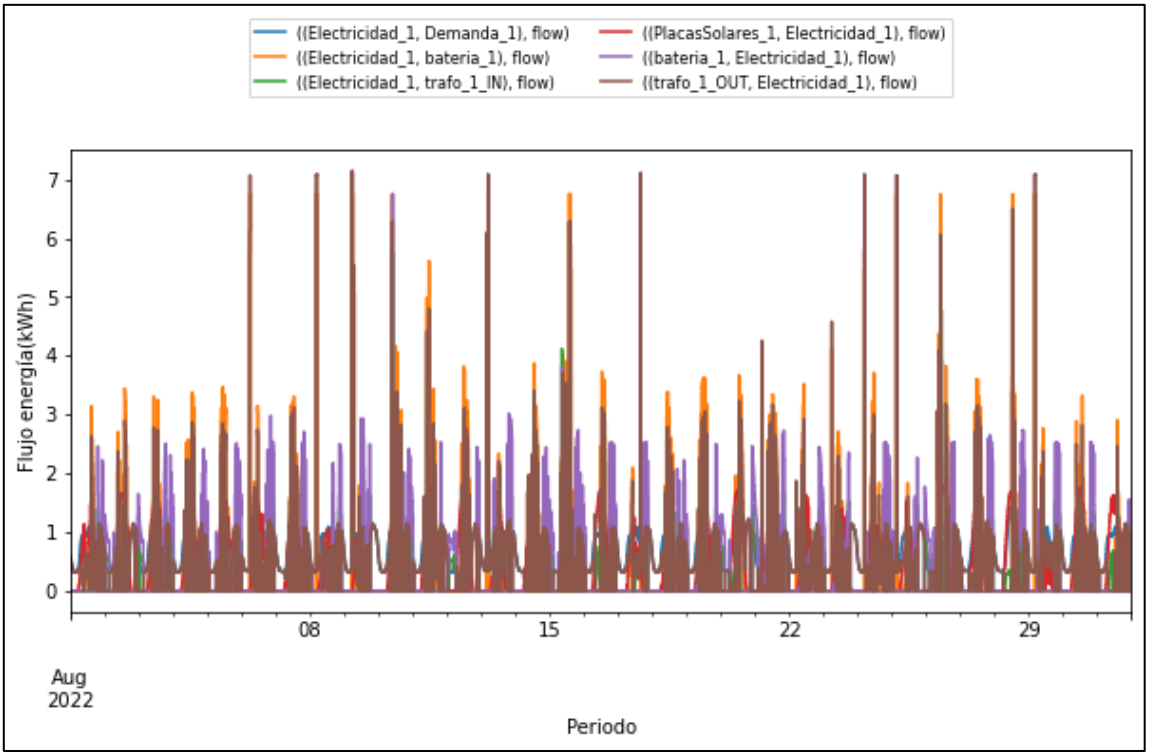

Figura 9.15. *Flujo del bus residencial caso 2.1.*

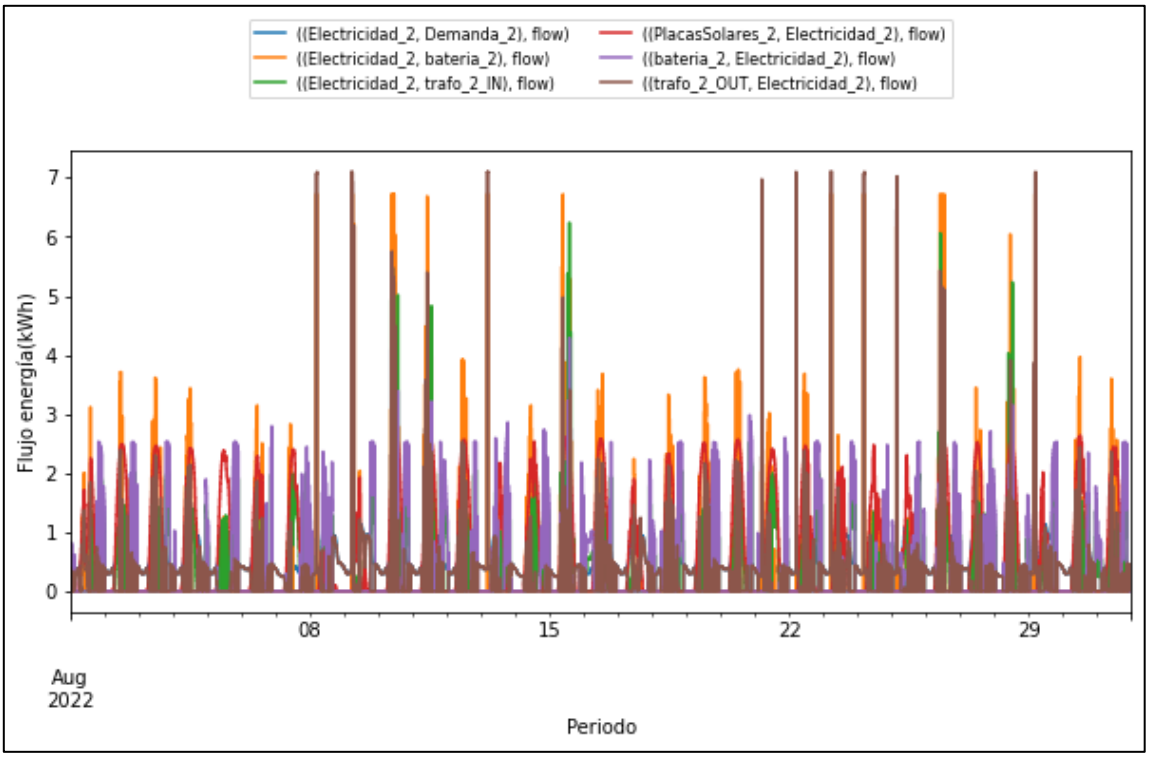

Figura 9.16. *Flujo del bus laboral entre semana caso 2.1.*

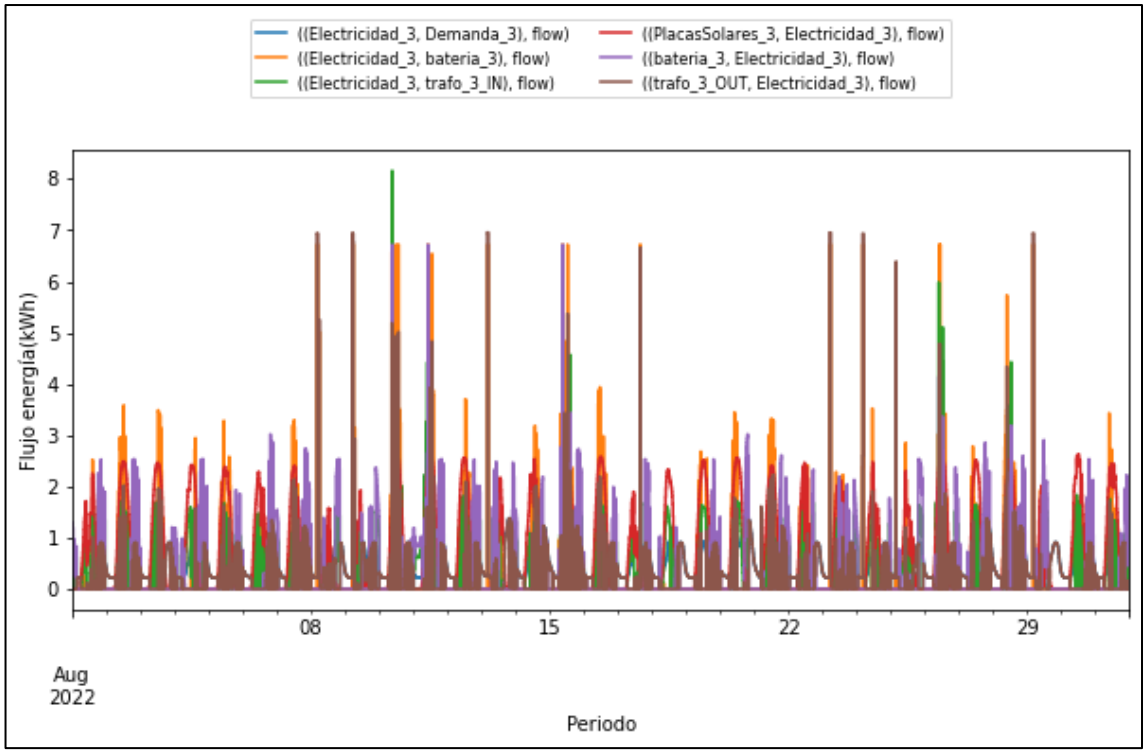

Figura 9.17. *Flujo del bus laboral fin de semana caso 2.1.*

Todas estas cuestiones mencionadas previamente y que en ocasiones pueden no ser fáciles de determinar en las gráficas, se han observado de forma clara en los resultados finales de todo el periodo, mostrados en la Figura 9.18.

```
((Deficit, Electricidad_0), flow)        1216.875722
((Electricidad_0, Exceso), flow)           12.162343
((Electricidad_0, trafo_1_OUT), flow)    1740.454195
((Electricidad_0, trafo_2_OUT), flow)    1062.679613
((Electricidad_0, trafo_3_OUT), flow)    1033.412587
((trafo_1_IN, Electricidad_0), flow)      494.021871
((trafo_2_IN, Electricidad_0), flow)     1080.965360
((trafo_3_IN, Electricidad_0), flow)     1056.845789
dtype: float64
((Electricidad_1, Demanda_1), flow)          2377.500959
((Electricidad_1, bateria_1), flow)          1037.976537
((Electricidad_1, trafo_1_IN), flow)          494.021871
((PlacasSolares_1, Electricidad_1), flow)    1313.704175
((bateria_1, Electricidad_1), flow)           855.340994
((trafo_1_OUT, Electricidad_1), flow)        1740.454195
dtype: float64
((Electricidad_2, Demanda_2), flow)          1783.175154
((Electricidad_2, bateria_2), flow)          1002.811985
((Electricidad_2, trafo_2_IN), flow)         1080.965360
((PlacasSolares_2, Electricidad_2), flow)    1977.415177
((bateria_2, Electricidad_2), flow)           826.857709
((trafo_2_OUT, Electricidad_2), flow)        1062.679613
dtype: float64
((Electricidad_3, Demanda_3), flow)          1783.229957
((Electricidad_3, bateria_3), flow)           975.431680
((Electricidad_3, trafo_3_IN), flow)         1056.845789
((PlacasSolares_3, Electricidad_3), flow)    1977.415177
((bateria_3, Electricidad_3), flow)           804.679662
((trafo_3_OUT, Electricidad_3), flow)        1033.412587
```

Figura 9.18. *Resumen general caso 2.1.*

En el caso previo sin baterías, se han obtenido unos valores finales de 2992,42 kWh para el déficit y de 2317,05 kWh para el exceso. En el caso actual, los resultados finales han sido 1216,87 kWh de déficit y 12,16 kWh de exceso. Se concluye que mediante la introducción de baterías la necesidad de trabajar con la red ha sido mucho menor, no siendo apenas necesaria la venta de energía. La compra a la red se ha reducido, pero sin embargo sigue siendo grande, por lo que sería interesante analizar la introducción de un sistema de generación con mayor capacidad que permita cubrir una mayor parte de la demanda y limitar más la interacción con la red.

También se ha comprobado de estos resultados finales como el edificio que más energía ha demandado al bus conexión es el edificio 1, con 1740,45 kWh, frente a los edificios 2 y 3 que han solicitado algo más de 1000 kWh, con unos valores similares ya que se han diseñado edificios con condiciones parecidas. En cuanto al aporte de los edificios al sistema, los que más han cedido han sido el 2 y el 3, con datos cercanos a los 1000 kWh y muy similares como se ha mencionado, mientras que el edificio 1 solo ha contribuido con 494,02 kWh. Los datos de demanda y generación de cada edificio son los mismos ya que se han mantenido los mismos datos, añadiendo únicamente las baterías.

Para terminar, se muestra la gráfica de uno de los 3 sistemas de almacenamiento, para observar cómo ha fluctuado la energía en las dos direcciones y la capacidad de almacenamiento que han tenido en cada momento. Se observa como a lo largo de cada día se tienen momentos en los que se han descargado mientras que en otros momentos se encuentran cargadas, lo cual tiene sentido ya que durante el días cuando se tiene mayor generación no hace falta emplearlas, mientras que en las horas nocturnas es donde tienen su mayor influencia.

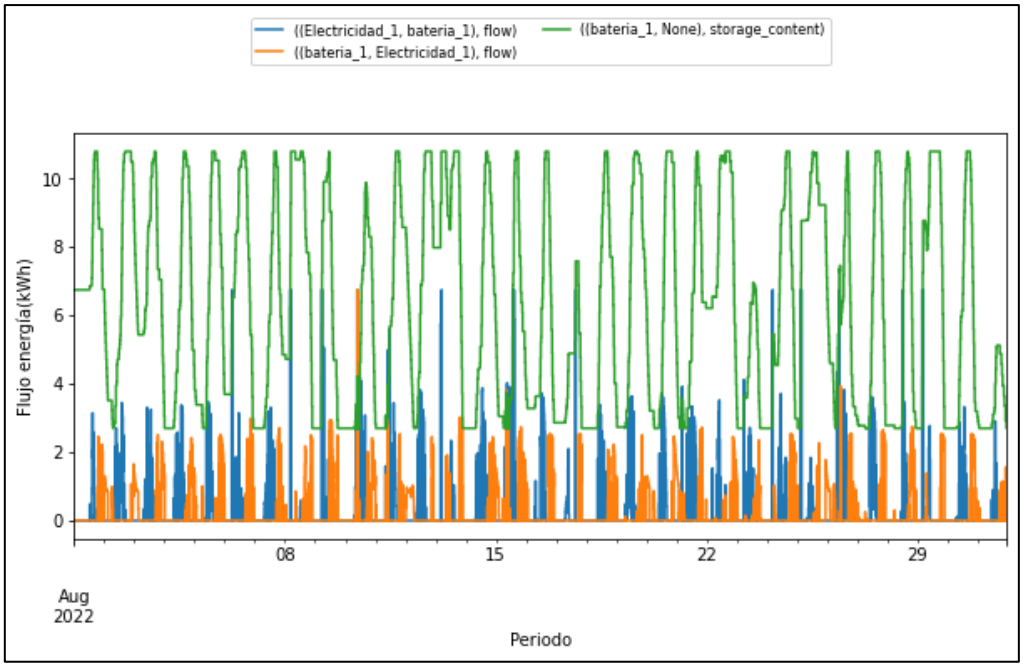

Figura 9.19. *Flujo y almacenamiento de la batería del edificio residencial caso 2.1.*

Las baterías seleccionadas se consideran adecuadas, ya que cuentan con suficiente energía para afrontar las diferentes situaciones que se dan. Se comprueba en la Figura 9.19, donde la batería no ha estado largos periodos con poca carga, lo que indicaría que su capacidad es elevada

para el proyecto; ni pasa mucho tiempo con mucha carga, que señalaría que harían falta baterías con mayor capacidad ya que las seleccionadas no cubren todo el almacenamiento posible.

Es importante destacar que, a la hora de indicar los parámetros de las baterías, se ha indicado una eficiencia de carga y descarga del 90%, por lo que no toda la energía que se transfiere a la batería está a posterior disponible para su uso, ya que en ambos procesos se han producido ciertas pérdidas. Se ha tomado este valor para las eficiencias ya que como se ha explicado en el apartado 5.5 las baterías tienen una eficiencia de entre el 86% y el 96%.

9.2.2. Instalación de un sistema de baterías en los edificios 2

Esta sección se trata un sistema similar al anterior respecto al almacenamiento ya que se han instalado baterías sólo en los edificios, pero sus capacidades y el resto de los parámetros del sistema se han cambiado para comprobar cómo actúa en otra situación. El periodo empleado ha sido también el mes de agosto de 2022.

Está formado por 2 edificios. El primero es un edificio con horario laboral entre semana, que cuenta con 200 metros cuadrados útiles y en él se han instalado 18 módulos fotovoltaicos. El segundo es un edificio residencial en el que hay 4 viviendas y cuenta con 15 módulos. La capacidad de las baterías es la misma para ambos edificios y se ha reducido en la mitad respecto al caso anterior, para comprobar que sucede con unas de menor tamaño en este caso, indicando el valor de 6,75 kWh de energía nominal y 3,36 kW de potencia de carga y descarga.

En la Figura 9.20 se presentan los flujos que tienen lugar en el bus de conexión.

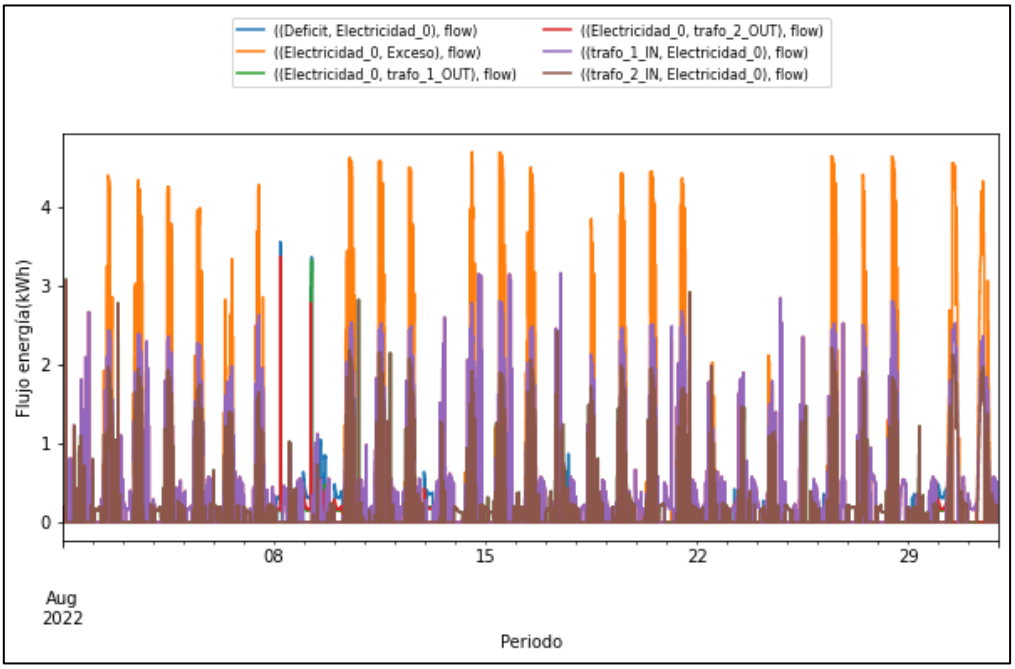

Figura 9.20. *Flujo del bus de conexión caso 2.2.*

La diferencia con el anterior caso es notable. Se observa como la curva que ha predominado es la de exceso, es decir, se ha generado más energía de la necesaria para consumir, y la capacidad

de las baterías no ha sido suficiente. El déficit es pequeño, pero existe, por lo que también se podría haber eliminado con unas baterías con mayor capacidad.

Todas estas conclusiones resultan muy visibles en el resumen general del proceso, mostrado en la Figura 9.21.

```
((Deficit, Electricidad_0), flow)          69.053458
((Electricidad_0, Exceso), flow)         1069.451751
((Electricidad_0, trafo_1_OUT), flow)     311.590304
((Electricidad_0, trafo_2_OUT), flow)     559.439381
((trafo_1_IN, Electricidad_0), flow)     1154.713679
((trafo_2_IN, Electricidad_0), flow)      716.714302
dtype: float64
((Electricidad_1, Demanda_1), flow)           891.587577
((Electricidad_1, bateria_1), flow)          3347.297924
((Electricidad_1, trafo_1_IN), flow)         1154.713679
((PlacasSolares_1, Electricidad_1), flow)    2366.441923
((bateria_1, Electricidad_1), flow)          2715.566952
((trafo_1_OUT, Electricidad_1), flow)         311.590304
dtype: float64
((Electricidad_2, Demanda_2), flow)          1188.750479
((Electricidad_2, bateria_2), flow)          3327.921691
((Electricidad_2, trafo_2_IN), flow)          716.714302
((PlacasSolares_2, Electricidad_2), flow)    1977.415177
((bateria_2, Electricidad_2), flow)          2696.531915
((trafo_2_OUT, Electricidad_2), flow)         559.439381
```

Figura 9.21.*Resumen general caso 2.2.*

El exceso final que ha tenido el conjunto ha sido muy elevado, lo cual se ha debido a la falta de capacidad de las baterías para las condiciones que se han tenido. Los flujos desde los edificios hacia el bus de conexión han sido muy grandes, y al tener dos sistemas de baterías pequeños y mucha generación, el único empleo que le ha quedado a esa energía excedente es el de venderla a la red. Además, a pesar de que la generación ha sido muy superior a la demanda, ha existido déficit en ciertos momentos del periodo determinado. Esto también ha sido una consecuencia de la incorrecta elección del tamaño de las baterías, ya que, teniendo un exceso notable de energía, con unas baterías con una mayor capacidad no hubiera existido necesidad de comprar a la red.

9.2.3. Instalación de un sistema de baterías general

En este caso se ha optado por instalar un sistema de baterías general, que permite almacenar en él los excedentes de todos los edificios del sistema. En la Figura 8.1, en la que se ha mostrado el esquema del sistema, viene representado con el nombre de *BateriaGenera*l, y como se ha observado está conectada al bus de conexión, permitiendo realizar ese almacenamiento común para todo el sistema.

El sistema que se ha tomado es el mismo del apartado 9.2.2., con el objetivo de comprobar si con la introducción de una batería más grande para todo el sistema se consigue eliminar las desventajas que se han obtenido en el apartado anterior. Por lo que se han tenido los mismos 2 edificios con los mismos datos para ellos. El único cambio que se ha introducido es la sustitución

de las 2 baterías que se ubicaban en cada edificio, por una batería general más grande. Se ha escogido la batería Cegasa Ultra 175, con la configuración 3x48V_280Ah, que se ha obtenido de la unión de 3 de las primeras, como se ve en la Figura 9.13. Su energía nominal es 40,5 kWh y sus potencia de carga y descarga son 19,2 kW.

El resumen final que se ha obtenido del funcionamiento es el enseñado en la Figura 9.22.

```
((BateriaGeneral, Electricidad_0), flow)    7952.160910
((Deficit, Electricidad_0), flow)              0.000000
((Electricidad_0, BateriaGeneral), flow)    9791.967664
((Electricidad_0, Exceso), flow)             423.712288
((Electricidad_0, trafo_1_OUT), flow)        345.954532
((Electricidad_0, trafo_2_OUT), flow)        568.601196
((trafo_1_IN, Electricidad_0), flow)        1820.808878
((trafo_2_IN, Electricidad_0), flow)        1357.265895
dtype: float64
((Electricidad_1, Demanda_1), flow)          891.587577
((Electricidad_1, trafo_1_IN), flow)        1820.808878
((PlacasSolares_1, Electricidad_1), flow)   2366.441923
((trafo_1_OUT, Electricidad_1), flow)        345.954532
dtype: float64
((Electricidad_2, Demanda_2), flow)         1188.750479
((Electricidad_2, trafo_2_IN), flow)        1357.265895
((PlacasSolares_2, Electricidad_2), flow)   1977.415177
((trafo_2_OUT, Electricidad_2), flow)        568.601196
```

Figura 9.22. *Resumen general caso 2.3.*

Se ha observado como el déficit se ha conseguido eliminar, mientras que el exceso también se ha reducido notablemente, por lo que con la instalación establecida el sistema actúa de mejor manera ya que no requiere de compra a la red, y el exceso que hay ha sido consecuencia de una mayor generación a la necesitada.

En la Figura 9.23 se presentan los flujos y el almacenamiento que muestra este sistema general. Se ha comprobado como la curva naranja del almacenamiento ha seguido un desarrollo en el que al principio no ha llegado a cargarse del todo, por lo que podría resultar demasiado grande, pero en los días finales sí que se ha hecho uso de la capacidad de las baterías, por lo que se puede concluir que los parámetros seleccionados para su diseño son correctos y ofrecen los beneficios buscados para el funcionamiento de la instalación.

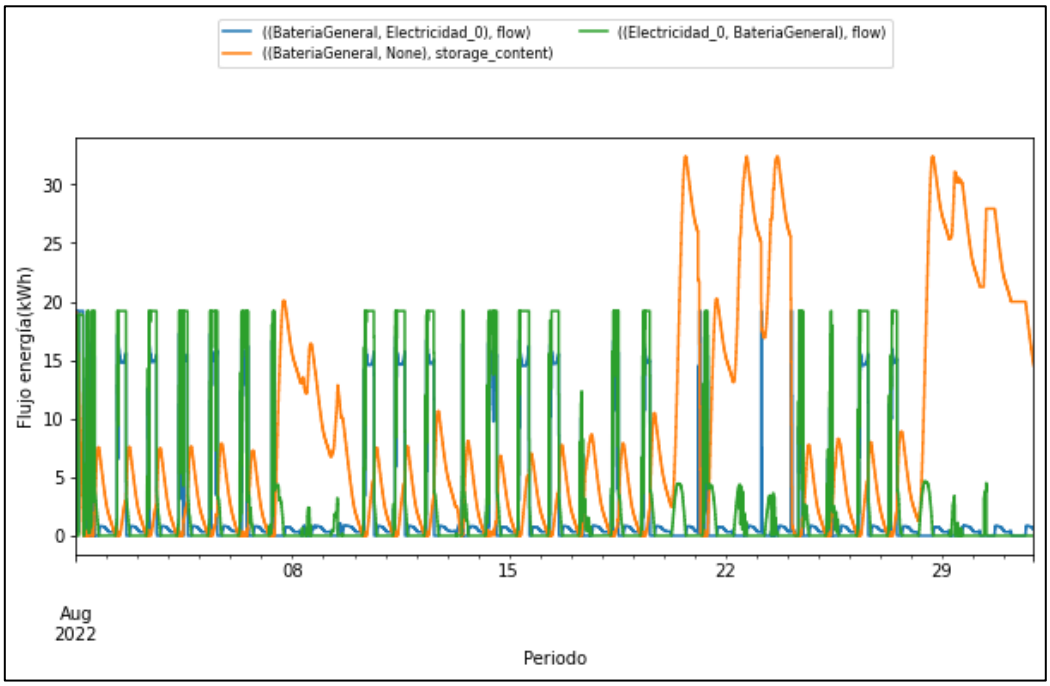

Figura 9.23. *Flujo y almacenamiento de la batería general caso 2.3.*

9.2.4. Instalación de baterías en los edificios y de una batería general

En este estudio se han introducido todas las condiciones antes presentadas para poder ver un análisis con todas las variables. El periodo se ha seleccionado con el objetivo de variarlo respecto a los casos anteriores para observar otra duración diferente y va desde el 1 de abril de 2022 hasta el 12 de mayo de 2022,

Se han añadido 3 edificios. El primero es un edificio residencial que cuenta con 6 viviendas y en el que hay instalados 12 módulos fotovoltaicos. Tiene un sistema de almacenamiento compuesto por el mismo sistema empleado en el apartado 9.2.1., formado por el tipo Ultra 175 48V_280Ah de las baterías Cegasa eBick Ultra175, que dispone de 13,5 kWh de energía nominal y una potencia de carga y descarga de 6,72 kW. El segundo representa otro edificio residencial, que está formado por 8 viviendas y 8 módulos. El tercero es un edificio laboral con horario predominante el fin de semana y cuenta con 300 metros cuadrados útiles y 18 módulos. En el caso de estos dos últimos edificios, no se dispone de almacenamiento.

Se ha instalado también un sistema de baterías general, compuesto por el tipo Ultra 175 48V_560Ah de la Cegasa eBick Ultra175, cuya energía nominal es 27 kWh, mientras que las potencias de carga y descarga, realizando lo mismo que en el primer caso, son de 13,44 kW. Esta última instalación ha resultado práctica al no contar con almacenamiento en los 2 últimos

edificios, por lo que su ausencia se ha compensado con la ubicación de un compuesto general. La gráfica de los flujos y el almacenamiento de estas baterías se muestra en la Figura 9.24.

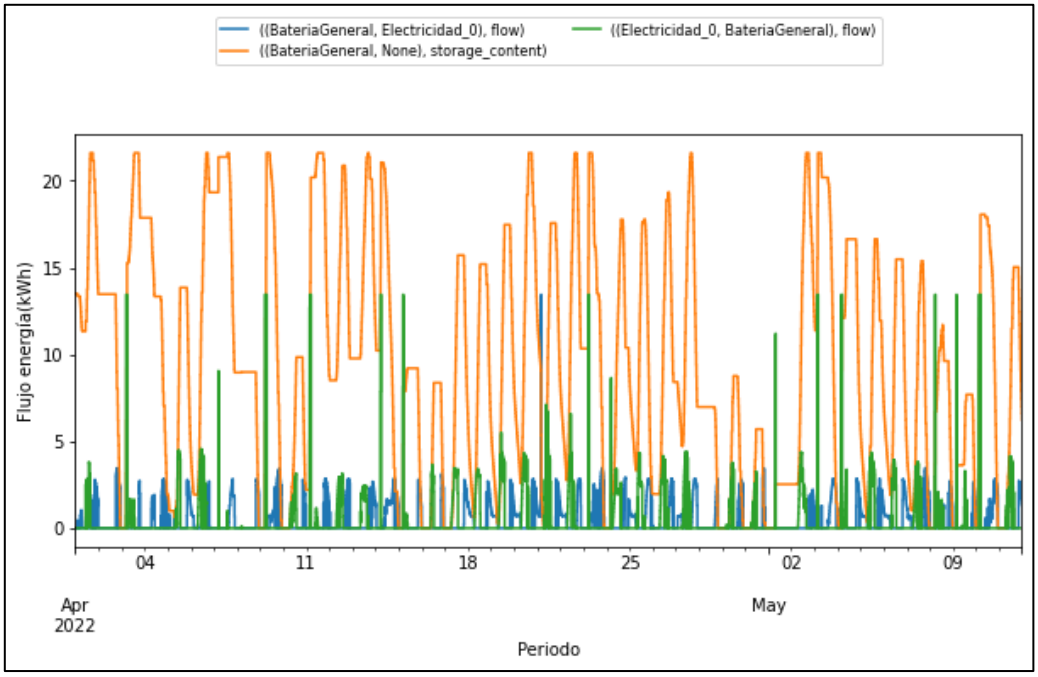

Figura 9.24. *Flujo y almacenamiento de la batería general caso 2.4.*

Se observa en la gráfica como la batería ha estado funcionando y ha podido almacenar energía en momentos en los que ha habido excedente. Ciertos momentos del periodo la batería ha estado con poca carga, por lo que la generación ha sido pequeña algunos días. Esto es posible que se deba a que los meses de abril y mayo cuentan con condiciones meteorológicas cambiantes y no han existido suficientes horas de sol para cubrir la demanda y que exista un excedente.

Para terminar, se tienen los resultados finales en la Figura 9.25. Se ha comprobado que la compra de energía a la red ha sido pequeña, con un dato de 24,32 kWh. El déficit que se ha tenido ha sido bastante grande, con un dato de 1924,67 kWh, que como se ha mencionado antes es debido a las malas condiciones meteorológicas para la generación o a que los sistemas fotovoltaicos no son suficientes para lo requerido por los edificios. Los flujos de las baterías muestran que han podido trabajar porque ha existido excedente en ciertos momentos del periodo y han conseguido reducir la compra a la red, que hubiera sido mayor en caso de no disponer de almacenamiento.

```
((BateriaGeneral, Electricidad_0), flow)    1865.418469
((Deficit, Electricidad_0), flow)           1924.670052
((Electricidad_0, BateriaGeneral), flow)    2270.766670
((Electricidad_0, Exceso), flow)              24.321060
((Electricidad_0, trafo_1_OUT), flow)       1837.382479
((Electricidad_0, trafo_2_OUT), flow)       2001.997793
((Electricidad_0, trafo_3_OUT), flow)        905.225738
((trafo_1_IN, Electricidad_0), flow)        1095.686609
((trafo_2_IN, Electricidad_0), flow)         141.022732
((trafo_3_IN, Electricidad_0), flow)        2012.895880
dtype: float64
((Electricidad_1, Demanda_1), flow)          2358.498742
((Electricidad_1, bateria_1), flow)          1756.325074
((Electricidad_1, trafo_1_IN), flow)         1095.686609
((PlacasSolares_1, Electricidad_1), flow)    1935.924634
((bateria_1, Electricidad_1), flow)          1437.203310
((trafo_1_OUT, Electricidad_1), flow)        1837.382479
dtype: float64
((Electricidad_2, Demanda_2), flow)          3144.664989
((Electricidad_2, trafo_2_IN), flow)          141.022732
((PlacasSolares_2, Electricidad_2), flow)    1283.689930
((trafo_2_OUT, Electricidad_2), flow)        2001.997793
dtype: float64
((Electricidad_3, Demanda_3), flow)          1768.944267
((Electricidad_3, trafo_3_IN), flow)         2012.895880
((PlacasSolares_3, Electricidad_3), flow)    2876.614410
((trafo_3_OUT, Electricidad_3), flow)         905.225738
dtype: float64
```

Figura 9.25. *Resumen general caso 2.4.*

10. PRESUPUESTO DE EJECUCIÓN

En esta sección se presentan los diferentes costes que ha conllevado el trabajo durante su realización.

Tabla 10.1. *Gastos totales del proyecto*

Concepto	**Coste unitario (€/hora)**	**Coste total(€)**
Ingeniero	10	6000
Ordenador	0,02	12
Software	0	0

En él ha intervenido un ingeniero que ha desarrollado todas las tareas. La fase inicial ha correspondido con un periodo de documentación y análisis de los diferentes softwares, en la siguiente fase se ha procedido a programar el sistema creado y en la fase final se ha hecho un estudio de los resultados. En total, las 3 fases han englobado 600 horas de trabajo del ingeniero.

En lo referido al material empleado, se ha empleado un ordenador Acer Aspire E 15, cuyo precio de adquisición ha sido de 300 euros, y se estima que se puede emplear durante 8 años. El periodo utilizado durante el trabajo ha sido las 600 horas de trabajo del ingeniero.

Respecto a los gastos de software, como se ha explicado al inicio del apartado 7, tanto Python como Oemof son gratuitos y de libre acceso, lo que ha aportado una gran ventaja respecto al resto de softwares presentados en el apartado 6. Debido a ello, no han existido gastos en lo que respecta al software.

11. CONCLUSIONES

Por último, se concluye que empleando el programa Python y los paquetes de Oemof se ha conseguido crear un programa que permite el diseño y la solución de sistemas de autoconsumo, con las funcionalidades que se obtienen con los softwares presentados en el apartado 6 y sin requerir de costes de software al tratarse de un programa gratuito.

Sin embargo, el programa cuenta con ciertas limitaciones, como la limitación del periodo para el cálculo de la irradiancia solar y la ausencia de un mayor número de módulos e inversores. Los paquetes empleados para calcular la generación y la demanda han funcionado correctamente y se ha demostrado su utilidad, pero se han encontrado ciertas restricciones en ellos. El tiempo requerido para el cálculo de la potencia generada mediante los paquetes Feedinlib ha sido muy grande cuando los periodos de estudio son grandes, ya que obtiene la información de la red y el proceso se ha hecho muy largo. El cálculo de la demanda se trata de una aproximación, por lo que para proyectos finales puede resultar más interesante obtener los datos reales de los edificios instalados. Por lo tanto, se determina que ellos paquetes de Oemof todavía requieren de evoluciones y actualizaciones que permitan hacer un análisis más completo y sin errores.

El análisis que se ha realizado determina que la introducción de los sistemas de almacenamiento es un apartado muy beneficioso para estos sistemas ya que al realizar el balance final se ha reducido la interacción con la red eléctrica, lo cual es uno de sus objetivos, ya que los precios de la electricidad son muy elevados y la dependencia de las compañías eléctricas es alta. Uno de los objetivos principales de las comunidades energéticas y los sistemas de autoconsumo es el de reducir esta dependencia, por lo que se observa que mediante esta propuesta se ha conseguido un resultado positivo que incluso supera las expectativas iniciales.

REFERENCIAS

[1] Comisión Europea, 30 de noviembre de 2016. Energía limpia para todos los europeos. **En**: *Comisión Europea* **[en línea].** Disponible en: https://ec.europa.eu/commission/presscorner/detail/es/IP_16_4009 **[consulta: abril de 2022]**.

[2] Comunidades Energéticas: En el marco del Plan de Recuperación, Transformación y Resiliencia. **En**: *IDAE* **[en línea].** Disponible en: **https://www.idae.es/ayudas-y-financiacion/comunidades-energeticas [consulta: abril de 2022]**.

[3] GONZÁLEZ RÍOS, Isabel, 2020. Las ‹‹Comunidades energéticas locales››. **En**: *Dialnet* **[en línea].** Disponible en: **https://dialnet.unirioja.es/servlet/articulo?codigo=7563618**

[consulta: abril de 2022].

[4] Real Decreto-ley 15/2018, de 5 de octubre, de medidas urgentes para la transición energética y la protección de los consumidores. **En**: *Boletín Oficial del Estado* **[en línea].** Disponible en: https://www.boe.es/buscar/doc.php?id=BOE-A-2018-13593 **[consulta: abril de 2022]**.

[5] El auge del autoconsumo fotovoltaico, un aliado contra el cambio climático **En**: *Iberdrola***[en línea].** Disponible en: **https://www.iberdrola.com/innovacion/autoconsumo-fotovoltaico [consulta: abril de 2022]**.

[6] CARMONA LÓPEZ; Omar; VIDAL SANTO; Adrián; MARTÍNEZ LÓPEZ, Andrea; CONDE, Jorge; TINOCO MAGAÑA, Luis, 2016. Estudio de la viabilidad técnica para la implementación de un sistema de autoconsumo eléctrico basado en paneles fotovoltaicos para una vivienda. **En:** *QUID* **[en línea].** Disponible en: **https://dialnet.unirioja.es/servlet/articulo?codigo=5704160 [consulta: abril de 2022]**.

[7] Real Decreto 244/2019, de 5 de abril, por el que se regulan las condiciones administrativas, técnicas y económicas del autoconsumo de energía eléctrica. **En**: *Boletín Oficial del Estado* **[en línea].** Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-5089 **[consulta: abril de 2022]**.

[8] Análisis del estado actual del almacenamiento detrás del contado en España. **En**: *IDAE* **[en línea].** Disponible en: **https://www.idae.es/sites/default/files/ESTUDIO8_Ana%CC%81lisis%20del%20estado%20actual%20del%20almacenamiento_21.pdf [consulta: abril de 2022]**.

[9] BARRERO, Antonio, 19 de noviembre de 2021. Autoconsumo: todo lo que querías saber sobre las baterías. **En**: *ENERGÍAS RENOVABLES* **[en línea].** Disponible en: **https://www.energias-renovables.com/almacenamiento/autoconsumo-todo-lo-que-querias-saber-sobre-20211119 [consulta: abril de 2022]**.

[10] Página web de Homer Energy. Disponible en: https://www.homerenergy.com/index.html **[consulta: julio de 2020]**.

[11]Homer Software. **En**: *Okinawa Enetech* **[en línea].** Disponible en: **https://openjicareport.jica.go.jp/pdf/12265039_03.pdf [consulta: abril de 2022]**.

[12] Página web de VALENTIN Software. Disponible en: **https://valentin-software.com/en/products/pvsol/ [consulta: julio de 2020]**.

[13] PVGIS en España: cómo utilizarlo. **En**: *SMART SPAIN* **[en línea].** Disponible en: **https://smartspain.es/pvgis-espana/ [consulta: abril de 2022]**.

[14] Página web de PV SYST. Disponible en: **https://www.pvsyst.com/features/ [consulta: julio de 2020]**.

[15] The end-to-end software to design and engineer photovoltaic plants. **En**: *RATED POWER* **[en línea].** Disponible en: **https://ratedpower.com/pvdesign/ [consulta: abril de 2022]**.

[16] Página web de Mathworks. Disponible en: **https://es.mathworks.com/products/matlab.html [consulta: abril de 2022]**.

[17] Página web de Oemof. Disponible en: https://oemof.readthedocs.io/en/latest/ **[consulta: abril de 2022]**.

[18] Código fuente de Oemof. Disponible en: https://github.com/oemof **[consulta: abril de 2022]**.

[19] Electrical Profiles. **En**: *Oemof* **[en línea].** Disponible en: https://demandlib.readthedocs.io/en/latest/bdew.html **[consulta: mayo de 2022]**.

[20] ¿Cuánto cuesta la luz al mes? Consumo medio de luz en España. **En**: *Tarifasgasluz* **[en línea].** Disponible en: https://tarifasgasluz.com/faq/cuanto-cuesta-luz-mes **[consulta: mayo de 2022]**.

[21] Energía en edificios de oficinas. **En**: *Enectiva* **[en línea].** Disponible en: https://www.enectiva.cz/es/blog/2015/06/ideas-energia-edificio-de-oficinas/ **[consulta: mayo de 2022]**.

[22] ALONSO LORENZO, José Alfonso, 2019. Radiación, Geometría, Recorrido óptico, Irradiancia y HSP. **En**: *SUNFIELDS EUROPE* **[en línea].** Disponible en: **https://www.sfe-solar.com/noticias/articulos/energia-fotovoltaica-radiacion-geometria-recorrido-optico-irradiancia-y-hsp/ [consulta: junio de 2022]**.

[23] Sistema modular plug and play de alta potencia para aplicaciones desde los 13 hasta los 54 kWh **En**: *Cegasa* **[en línea].** Disponible en: **https://www.cegasa.com/datos/documentosDescargas/archivo_79_1/ebick_ultra_175_esp_web.pdf [consulta: junio de 2022]**.

[24] Página web de Esios. Disponible en: **https://www.esios.ree.es/es?locale=es [consulta: junio de 2022]**.

ANEXOS

A) Código de la función *diasHastaFecha*

```
def diasHastaFecha(day1, month1, year1, day2, month2, year2):

    # Función para calcular si un año es bisiesto o no
    def esBisiesto(year):
        return year % 4 == 0 and year % 100 != 0 or year % 400 == 0

    # Caso de años diferentes
    if (year1<year2):

        # Días restante primer año
        if esBisiesto(year1) == False:
            diasMes = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        else:
            diasMes = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        restoMes = diasMes[month1] - day1
        restoYear = 0
        i = month1 + 1
        while i <= 12:
            restoYear = restoYear + diasMes[i]
            i = i + 1

        primerYear = restoMes + restoYear
        # Suma de días de los años que hay en medio
        sumYear = year1 + 1
        totalDias = 0
        while (sumYear<year2):
            if esBisiesto(sumYear) == False:
                totalDias = totalDias + 365
                sumYear = sumYear + 1
            else:
                totalDias = totalDias + 366
```

```
            sumYear = sumYear + 1
        # Dias año actual
        if esBisiesto(year2) == False:
            diasMes = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        else:
            diasMes = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        llevaYear = 0
        lastYear = 0
        i = 1
        while i < month2:
            llevaYear = llevaYear + diasMes[i]
            i = i + 1
        lastYear = day2 + llevaYear
        return totalDias + primerYear + lastYear
    # Si estamos en el mismo año
    else:
        if esBisiesto(year1) == False:
            diasMes = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        else:
            diasMes = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        llevaYear = 0
        total = 0
        i = month1
        if i < month2:
            while i < month2:
                llevaYear = llevaYear + diasMes[i]
                i = i + 1
            total = day2 + llevaYear - 1
            return total
        else:
            total = day2 - day1
            return total
```

B) Código de las funciones de los paquetes Demandlib

```
import datetime
import numpy as np
from matplotlib import pyplot as plt
import demandlib.bdew as bdew
from funciones.DiasEntreFechas import diasHastaFecha

#Este programa sirve para obtener la demanda de los edificios del sistema,
#diferenciando entre los diferentes tipos de edificios que se pueden tener.
#Se obtienen datos de demanda para cada 15 minutos, a partir del dato de demanda
#anual

class Demand():

  def Demanda():
    #Para definir las vacaciones se toma como referencia el calendario
    #festivo del País Vasco
    holidays = {
        datetime.date(2022,1,1): "Año nuevo",
        datetime.date(2022,1,6): "Día de Reyes",
        datetime.date(2022,4,14): "Jueves Santo",
        datetime.date(2022,4,15): "Viernes Santo",
        datetime.date(2022,4,18): "Lunes de Pascua",
        datetime.date(2022,7, 25): "Santiago",
        datetime.date(2022,8,15): "Asunción de la Virgen",
        datetime.date(2022,9,6): "Día de Elcano",
        datetime.date(2022,10,12): "Fiesta Nacional de España",
        datetime.date(2022,11,1): "Todos los Santos",
        datetime.date(2022,12,6): "Día de la Constitución",
        datetime.date(2022,12,8): "Inmaculada",
        }
    Año1=int(input("Indique el año en el que se va a iniciar el estudio:"
```

```
            " "))
Año2=int(input("Indique el año en el que se va a finalizar el estudio:"
            " "))
Mes1=int(input("Indique el mes de inicio del estudio: "))
Dia1=int(input("Indique el día de inicio del estudio: "))
Mes2=int(input("Indique el mes de fin del estudio: "))
Dia2=int(input("Indique el día de fin del estudio: "))
dias=diasHastaFecha(Dia1,Mes1,Año1,Dia2,Mes2,Año2)

#Para la demanda anual, en el caso de los edificios de viviendas(h0), se
#toma la demanda anual(kWh) aproximada de una vivienda y se multiplica
#por el número de viviendas. Para el resto de edificios se toma el dato
#de demanda(kW) por metro cuadrado útil, y después se multiplica por
#este valor.
demanda_por_sector = {
   "g0": 52.5*(dias/365),
   "h0": 3500*(dias/365),
   "g6": 52.5*(dias/365),
   }

# A continuación se obtienen los perfiles de demanda
dias=dias*24
perfiles = bdew.ElecSlp(Dia1,Mes1,Año1,Dia2,Mes2,Año2,dias,
                holidays=holidays)

# Aquí se multiplica por la demanda anual seleccionada
elec_demand = perfiles.get_profile(demanda_por_sector)

DemandaElectricakWh=elec_demand.truediv(4)

# Para graficar, se emplea la función resample para obtener valores
```

```
    # cada hora
    elec_demand_resampled = elec_demand.resample("H").mean()

    # Gráfico de la demanda
    ax = elec_demand_resampled.plot()
    ax.set_xlabel("Fecha")
    ax.set_ylabel("Demanda eléctrica")
    plt.show()

    for key in demanda_por_sector:
      assert np.isclose(
      elec_demand[key].sum() / 4, demanda_por_sector[key]
    )
    return DemandaElectricakWh,Mes1,Dia1,Mes2,Dia2,Año1,Año2,dias
```

C) <u>Código de las funciones de los paquetes Feedinlib</u>

```
from feedinlib.powerplants import Photovoltaic
from feedinlib.open_FRED import Weather
from feedinlib.open_FRED import defaultdb
from shapely.geometry import Point
import warnings
warnings.filterwarnings("ignore")
from feedinlib.models import get_power_plant_data
import matplotlib.pyplot as plt

class FeedIn():

  #Con esta función se obtienen los diferentes módulos fotovoltaicos e
  #inversores que permite el programa.
  def ModulosInversores():
    #obtener los módulos
    module_df = get_power_plant_data(dataset='sandiamod')
```

```
    # imprimir los módulos
    module_df.iloc[:, 1:5]
    #obtener inversores y imprimirlos
    inverter_df = get_power_plant_data(dataset='cecinverter')
    inverter_df.iloc[:, 1:5]
    return module_df,inverter_df


#Esta función permite calcular la irradiancia de un determinado lugar a lo
#largo de un periodo de tiempo
def IrradianciaSistemaFotovoltaico(Mes1,Dia1,Mes2,Dia2,Año1,Año2):

    azimuth=int(input("Indica la orientación que van a tener los paneles"
                      " teniendo en cuenta lo siguiente:\n"
                      "-Norte=0\n"
                      "-Este=90\n"
                      "-Sur=180\n"
                      "-Oeste=270\n"
                      "IMPORTANTE: En caso de poder colocar los paneles en"
                      " cualquier dirección, hacerlo en la dirección "
                      "Sur=180, donde se conseguirá la mayor potencia.\n"
                      "\nSelección:"))

    system_data = {
        'module_name': 'Advent_Solar_Ventura_210___2008_',
        'inverter_name': 'ABB__PVI_3_0_OUTD_S_US__240V_' ,
        'azimuth': azimuth,
        'tilt': 30,
        'albedo': 0.2}
    placas=int(input("Indica la cantidad de módulos fotovoltaicos que se"
                      " van a instalar en este edificio: "))
```

```
system_data['modules_per_string']= placas

#con estos datos se genera el sistema
pv_system = Photovoltaic(**system_data)

#aqui se da la ubicación del sistema y se obtienen los datos
latitud=float(input("Ingresa la latitud del punto de colocación de los"
            " paneles fotovoltaicos: "))

longitud=float(input("Ingresa la longitud del punto de colocación de "
             "los paneles fotovoltaicos: "))

lat = latitud
lon = longitud
location = Point(lon, lat)
print("")
print("-----------------------------------------------------------------"
    "------------------------------")
print("El año límite para el cálculo de la irradiancia es el 2018 por"
    " limitaciones del programa,"
    "por lo que se va a emplear este año o años previos,"
    "en caso de que la duración sea mayor a 1 año, para el cálculo"
    " de la irradiancia")
print("-----------------------------------------------------------------"
    "------------------------------")
if Año2<=2018:
   inicio=f"{Año1}/{Mes1}/{Dia1}"
   final=f"{Año2}/{Mes2}/{Dia2}"

elif Año2==2019 and Mes2==1 and Dia2==1:
   inicio=f"{Año1}/{Mes1}/{Dia1}"
```

```
        final=f"{Año2}/{Mes2}/{Dia2}"

    else:
        if Mes2==1 and Dia2==1:
            Año=Año2-Año1
            inicio=f"{2018-Año}/{Mes1}/{Dia1}"
            final=f"2019/{Mes2}/{Dia2}"
        else:
            Año=Año2-Año1
            inicio=f"{2018-Año}/{Mes1}/{Dia1}"
            final=f"2018/{Mes2}/{Dia2}"



    open_FRED_weather_data = Weather(
        start=inicio, stop=final,
        locations=[location],
        variables="pvlib",
        **defaultdb())

    weather_df = open_FRED_weather_data.df(location=location, lib="pvlib")

    return pv_system, weather_df, lat, lon

#Con el valor de irradiancia obtenido en la función anterior, se obtiene
#con esta función el valor de la potencia en KW que se puede obtener
def PotenciaEnkW(weather,latitud,longitud,sistema,a):

    feedin = sistema.feedin(
    weather=weather,
    location=(latitud, longitud))
    feedin=feedin/1000
```

```
    feedin=feedin.replace(-0.0001,0)

    ax1 = feedin.plot(title=f"Generación {a}")
    ax1.set_xlabel('Tiempo')
    ax1.set_ylabel('Potencia en kW')
    plt.show()

    return feedin
```

D) <u>Código de las funciones de los paquetes oemof.solph</u>

```
from oemof import solph

#Este programa se emplea para crear funciones para representar los diferentes
#elementos del sistema eléctrico.

class FuncionesSolph():

  #Representación de los buses.
  def Buses(a):
    bus=solph.Bus(label=f"Electricidad_{a}")
    return bus

  #Aquí se representan los componentes de exceso y déficit del sistema, que
  #se emplearán dependiendo las necesidades de éste.
  def Exceso(bus_conexion,coste):
    exceso = solph.Sink(label="Exceso",
                inputs={bus_conexion:
                    solph.Flow(variable_costs=coste)})
    return exceso

  def Deficit(bus_conexion,coste):
    deficit=solph.Source(label="Deficit",outputs={bus_conexion:
```

```
    solph.Flow(variable_costs=coste)})
    return deficit

#Función para la generación mediante placas solares
def PlacasSolares(bus,Potencia,a):
    placas=solph.Source(label=f"PlacasSolares_{a}",
    outputs={bus: solph.Flow(fix=Potencia,
    nominal_value=1, Fixed=True)})
    return placas

#Representación de la demanda del edificio, empleando la función Demanda de
#Demandlib1
def DemandaEdificio(bus,Demanda,a):
    DemandaEdificio=solph.Sink(label=f"Demanda_{a}",
    inputs={bus:
    solph.Flow(fix=Demanda, nominal_value=1,
    Fixed=True)})
    return DemandaEdificio

#La siguiente función se emplea para conectar los elementos del sistema.
def Conexion(bus_in,bus_out,a,b):
    Conexion=solph.Transformer(label=f"trafo_{a}_{b}",
    inputs={bus_in: solph.Flow()}, outputs={bus_out: solph.Flow()},)
    return Conexion

#Para finalizar, se representan las baterías y sus diferentes parámetros.
#Se realiza una función para las posibles baterías de cada edificio, y otra
#para el caso de que se instale una para todo el sistema. En el caso de los
#edificios dependiendo el tamaño se emplean diferentes baterías.

def Bateria(bus,a,cap,Pcarga,Pdescarga):
```

```
    bateria = solph.GenericStorage(label=f"bateria_{a}",
    nominal_storage_capacity=cap,
    inputs={bus: solph.Flow(nominal_value=Pcarga,
    variable_costs=0.001)},
    outputs={bus: solph.Flow(nominal_value=Pdescarga,
    variable_costs=0.001)},
    loss_rate=0.00,
    initial_storage_level=0.5,
    balanced=False,
    inflow_conversion_factor=0.9, #eficiencia al cargar
    outflow_conversion_factor=0.9, #eficiencia al descargar
    min_storage_level=0.2, #Estado de carga (SoC mín)
    max_storage_level=0.8, #Estado de carga (SoC máx)
    )

    return bateria

def BateriaGen(bus,cap,Pcarga,Pdescarga):
    bateria = solph.GenericStorage(label="BateriaGeneral",
    nominal_storage_capacity=cap,
    inputs={bus: solph.Flow(nominal_value=Pcarga,
    variable_costs=0.002)},
    outputs={bus: solph.Flow(nominal_value=Pdescarga,
    variable_costs=0.002)},
    loss_rate=0.00,
    initial_storage_level=0.5,
    balanced=False,
    inflow_conversion_factor=0.9, #eficiencia de carga
    outflow_conversion_factor=0.9, #eficiencia de descarga
    min_storage_level=0, #Soc min
    max_storage_level=0.8, #Soc máx
```

```
    )
    return batería
```

E) Código del programa final

```
from funciones.feedinlib1 import FeedIn
from funciones.demandlib1 import Demand
from funciones.Solph import FuncionesSolph
import pandas as pd
from oemof import solph
import logging
import matplotlib.pyplot as plt
import os

filename=os.path.join(os.getcwd(),"Precios.csv")
data=pd.read_csv(filename, sep=";")

print("#########################################################")
print("")
print("Este programa sirve para realizar el diseño de un sistema "
    "de autoconsumo fotovoltaico.")
print("")
print("Indique las variables solicitadas para diseñar su sistema.")
print("")
print("#########################################################")

Demanda15min,Mes1,Dia1,Mes2,Dia2,Año1,Año2,periodos=Demand.Demanda()

print("")
print("-----------------------------------------------------------------"
    "----------")
print("Una vez seleccionado el periodo en el que se va a realizar el estudio, "
    "se va a proceder a diseñar el sistema.\n")
```

```
inicio=f"{Año1}/{Mes1}/{Dia1}"


logging.info("Inicializar el sistema")
date_time_index=pd.date_range(inicio,periods=periodos*4, freq="15min")
#formato fecha mm/dd/yyyy
energysystem=solph.EnergySystem(timeindex=date_time_index)

#Se crean los buses de conexión y los componentes de exceso y déficit,
#que van a ser independientes del número de edificios que tenga el sistema
a=0
BusConexion=FuncionesSolph.Buses(a)
Excedente=FuncionesSolph.Exceso(BusConexion, data["precioexcedente"])
Deficit=FuncionesSolph.Deficit(BusConexion, data["preciocompra"])
energysystem.add(BusConexion,Excedente,Deficit)

print("Existe la posibilidad de ubicar una batería para todo el sistema que "
    "almacene los posibles excedentes de todos los edificios.\n")
print("Seleccione 1 en caso de querer colocar esta batería, o seleccione 2 en"
    " caso de no querer disponer de ella.\n")
batgen=int(input("Selección: "))
print("--------------------------------------------------------------------"
    "----------")

if batgen==1:
   print("Indique los valores de los siguientes parámetros de la batería:")
   cap=float(input("Capacidad nominal (kWh): "))
   Pcarga=float(input("Potencia de carga (kW): "))
   Pdescarga=float(input("Potencia de descarga (kW): "))
   BateriaGeneral=FuncionesSolph.BateriaGen(BusConexion,cap,Pcarga,Pdescarga)
   energysystem.add(BateriaGeneral)
```

```
print("\nAhora se procede a añadir los edificios que van a formar el sistema:"
    "\n")
op=1
a=1
Bus=dict()
Generacion=dict()
Consumo=dict()
TrafoIn=dict()
TrafoOut=dict()
Bateria=dict()
bat=dict()
DatosPotencia=pd.DataFrame()

while op != 4: #Se ejecuta mientras op sea diferente de 4
  print('1.Añadir edificio residencial\n2.Añadir edificio con horario '
      'laboral'
      '(colegios,edificios de oficinas...)\n3.Añadir edificio con horario '
      'predominante el fin de semana\n4.Selección de edificios terminada')
  #Muestra las opciones
  op = int(input('Ingresa una opcion: ')) # Usuario ingresa opcion
  print("------------------------------------------------------------------"
      "-------------------------")

  if op == 1:
    print('¿Desea instalar un sistema de almacenamiento en el edificio '
        'seleccionado?')
    print('Seleccione una de las siguientes opciones:')
    print('1->Edificio con almacenamiento')
    print('2->Edificio sin almacenamiento')
    bat[a]=int(input('Indica la opción seleccionada: '))
```

```
    n=int(input(
      "Indica el número de viviendas que contiene el edificio: "))

    #Obtener los datos de potencia de este sistema
    sistema,tiempo,latitud,longitud=FeedIn.IrradianciaSistemaFotovoltaico(
      Mes1,Dia1,Mes2,Dia2,Año1,Año2)
    Potencia=FeedIn.PotenciaEnkW(tiempo,latitud,longitud,sistema,a)
    PotenciaData=pd.DataFrame(Potencia)
    DatosPotencia[a]=PotenciaData

    #Ahora se va a crear el sistema
    Bus[a]=FuncionesSolph.Buses(a)
    Generacion[a]=FuncionesSolph.PlacasSolares(Bus[a],DatosPotencia[a],a)
    Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a], Demanda15min.h0*n,a)
    TrafoIn[a]=FuncionesSolph.Conexion(Bus[a], BusConexion,a,"IN")
    TrafoOut[a]=FuncionesSolph.Conexion(BusConexion, Bus[a],a,"OUT")
    energysystem.add(Bus[a],Generacion[a],Consumo[a],
                TrafoIn[a],TrafoOut[a])

    if bat[a]==1:
      print("Indique los valores de los siguientes parámetros"
          "de la batería:")
      cap=float(input("Capacidad nominal (kWh): "))
      Pcarga=float(input("Potencia de carga (kW): "))
      Pdescarga=float(input("Potencia de descarga (kW): "))
      Bateria[a]=FuncionesSolph.Bateria(Bus[a],a,cap,Pcarga,Pdescarga)
      energysystem.add(Bateria[a])

    a+=1
  elif op == 2 or op==3:
    print('¿Desea instalar un sistema de almacenamiento en el edificio '
```

```
    'seleccionado?')
print('Seleccione una de las siguientes opciones:')
print('1->Edificio con almacenamiento')
print('2->Edificio sin almacenamiento')
bat[a]=int(input('Indica la opción seleccionada: '))
m2=int(input("Indica los metros cuadrados útiles de los que dispone el"
        " edificio: "))

#Obtener los datos de potencia de este sistema
sistema,tiempo,latitud,longitud=FeedIn.IrradianciaSistemaFotovoltaico(
  Mes1,Dia1,Mes2,Dia2,Año1,Año2)
Potencia=FeedIn.PotenciaEnkW(tiempo,latitud,longitud,sistema,a)
PotenciaData=pd.DataFrame(Potencia)
DatosPotencia[a]=PotenciaData
#Ahora se va a crear el sistema
Bus[a]=FuncionesSolph.Buses(a)
Generacion[a]=FuncionesSolph.PlacasSolares(Bus[a],DatosPotencia[a],a)
if op==2:
  Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a],
                        Demanda15min.g0*m2,a)
elif op==3:
  Consumo[a]=FuncionesSolph.DemandaEdificio(Bus[a],
                        Demanda15min.g6*m2,a)
TrafoIn[a]=FuncionesSolph.Conexion(Bus[a], BusConexion,a,"IN")
TrafoOut[a]=FuncionesSolph.Conexion(BusConexion, Bus[a],a,"OUT")
energysystem.add(Bus[a],Generacion[a],Consumo[a],
          TrafoIn[a],TrafoOut[a])
if bat[a]==1:
  print("Indique los valores de los siguientes parámetros "
      "de la batería:")
  cap=float(input("Capacidad nominal (kWh): "))
```

```
        Pcarga=float(input("Potencia de carga (kW): "))
        Pdescarga=float(input("Potencia de descarga (kW): "))
        Bateria[a]=FuncionesSolph.Bateria(Bus[a],a,cap,Pcarga,Pdescarga)
        energysystem.add(Bateria[a])

    a+=1

elif op == 4:
    print("-----------------------------------------------------------------"
        "-----------------------------")
    print('Diseño del sistema completado, se procede a realizar los '
        'cálculos')
    print("-----------------------------------------------------------------"
        "-----------------------------")
    model = solph.Model(energysystem)
    model.solve(solver="cbc", solve_kwargs={'tee': True})

    energysystem.results["main"] = solph.processing.results(model)
    energysystem.results["meta"] = solph.processing.meta_results(model)
    energysystem.dump(dpath=None, filename=None)
    energysystem = solph.EnergySystem()
    energysystem.restore(dpath=None, filename=None)

    results = energysystem.results["main"]
    i=1
    e=0
    results_baterias=dict()
    custom_baterias=dict()
    electricity_bus=dict()

    while e<a:
```

```
    electricity_bus[e] = solph.views.node(results, f"Electricidad_{e}")
    print(electricity_bus[e]["sequences"].sum(axis=0))
    if plt is not None:
      fig, ax = plt.subplots(figsize=(10,5))
      electricity_bus[e]["sequences"].plot( ylabel=
      "Flujo energía(kWh)",xlabel="Periodo",
      ax=ax, kind="line", drawstyle="steps-post")
      plt.legend(
      loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
      1.3), ncol=2)

      fig.subplots_adjust(top=0.8)
      plt.show()
    e+=1

  while i<a:
    if bat[i]==1:
      results_baterias[i]=energysystem.groups[f"bateria_{i}"]
      print(results[(results_baterias[i], None)]["sequences"])
      print("")
      custom_baterias[i] = solph.views.node(results, f"bateria_{i}")
      if plt is not None:
        fig, ax = plt.subplots(figsize=(10, 5))
        custom_baterias[i]["sequences"].plot( ylabel=
        "Flujo energía(kWh)",xlabel="Periodo",
        ax=ax, kind="line", drawstyle="steps-post")
        plt.legend(
        loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
        1.3), ncol=2)
        fig.subplots_adjust(top=0.8)
        plt.show()
```

```
        i+=1
    if batgen==1:
        results_bateriageneral=energysystem.groups["BateriaGeneral"]
        print(results[(results_bateriageneral, None)]["sequences"])
        print("")
        custom_bateriageneral = solph.views.node(results, "BateriaGeneral")
        if plt is not None:
            fig, ax = plt.subplots(figsize=(10, 5))
            custom_bateriageneral["sequences"].plot( ylabel=
            "Flujo energía(kWh)",xlabel="Periodo",
            ax=ax, kind="line", drawstyle="steps-post")
            plt.legend(
            loc="upper center", prop={"size": 8}, bbox_to_anchor=(0.5,
            1.3), ncol=2)
            fig.subplots_adjust(top=0.8)
            plt.show()

else:
    print("----------------------------------------------------------------"
        "------------------------------")
    print('Ingrese una opcion valida')
    print("----------------------------------------------------------------"
        "------------------------------")
```