Trabajo Fin de Grado
Ingeniería de tecnologías industriales

Modificación del modelo SHIPCal para simular en pasos sub-horarios.

Autor: Daniel Montalvo Rufián

Tutor: Miguel Frasquet Herraiz

Tutor ponente: Manuel Antonio Silva Pérez

Dpto. Ingeniería Energética
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería Energética

Modificación del modelo SHIPCal para simular en pasos sub-horarios.

Autor:

Daniel Montalvo Rufián

Tutor:

Miguel Frasquet Herraiz

Tutor ponente:

Manuel Antonio Silva Pérez

Dpto. de Ingeniería Energética

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Modificación del modelo SHIPCal para simular en pasos sub-horarios.

Autor:	Daniel Montalvo Rufián
Tutor:	Miguel Frasquet Herraiz
Tutor ponente:	Manuel Antonio Silva Pérez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2020

Resumen

La energía térmica tiene un importante papel en los procesos industriales en la actualidad. Para conseguirla, se recurre a calderas para generar el calor y transferirlo al fluido caloportador del proceso. Esta tarea puede verse auxiliado por la energía solar de concentración que provee energía térmica de calidad y con iguales prestaciones que el calor producido por la caldera.

El simulador solar SHIPCal (*Solar Heat for Industrial Processes CALculator)* es un modelo de uso público desarrollado en lenguaje Python en el que, mediante iteraciones horarias, se analizan los parámetros y prestaciones de un campo solar de concentración integrado en un proceso durante un tiempo determinado de simulación. Actualmente tiene implementado 14 modelos de integración de procesos industriales que se recogen en la guía de integración de la Task 49 de la Agencia Internacional de la Energía: *Solar Process Heat for Production and Advanced Applications- Deliverable B2: Integration Guideline*. [1]

El objetivo de este trabajo es modificar el código fuente de SHIPCal de manera que simule en pasos iterativos de 10 minutos y de 15 minutos para que se puedan conseguir simulaciones más detalladas y precisas respecto a la simulación de paso horario. Por ello es necesario cambiar las distintas funciones del modelo para habilitarlo a leer datos introducidos en formato minutal, calcular los parámetros solares correspondientes y procesarlos para calcular las prestaciones energéticas según el modelo de integración industrial para finalmente generar los correspondientes gráficos e informes.

En el trabajo se explica el funcionamiento y estructura de SHIPCal para justificar así los cambios introducidos. Al finalizar las modificaciones, se detallan los resultados obtenidos con ejemplos de simulaciones de un año completo de duración y de un día concreto. También se comparan los resultados y diferencias entre la simulación horaria y la simulación de pasos diezminutales y quinceminutales.

Abstract

Thermal energy plays an important role in industrial processes today. To achieve this, boilers are used to generate heat and transfer it to the heat transfer fluid of the process. This task can be assisted by concentrating solar energy that provides quality thermal energy with the same performance as the heat produced by the boiler.

The SHIPCal solar simulator (Solar heat for industrial processes CALculator) is a model for public use developed in Python language in which, by means of hourly iterations, the parameters and performance of a solar concentration field integrated in a process are analyzed for a certain time. simulation. It currently has 14 industrial process integration models implemented which are included in the integration guide of Task 49 of the International Energy Agency: Solar process heat for production and advanced applications - Deliverable B2: Integration guideline [1].

The objective of this work is to modify the source code of SHIPCal so that it simulates in iterative steps of 10 minutes and 15 minutes so that more detailed and precise simulations can be achieved with respect to the time step simulation. For this reason, it will be necessary to change the different functions of the model to enable it to read data entered in minute format, calculate the corresponding solar parameters and process them to calculate the energy performance according to the industrial integration model to finally generate the corresponding graphs and reports. The project explains the operation and structure of SHIPCal, as well as the changes introduced.

At the end of the modifications, the results obtained are detailed with examples of simulations lasting a full year and on a specific day. The results and differences between the hourly simulation and the simulation of ten-minute and fifteen-minute steps are also compared.

Índice

Resumen ..... **vii**
Abstract ..... **viii**
Índice ..... **ix**
Índice de Figuras ..... **xii**
Índice de Tablas ..... **xiv**
1 Introducción. **1**
1.1 Energía térmica solar para procesos industriales. *1*
1.2 Tecnología termosolar de alta temperatura. *2*
1.2.1 Concentradores tipo Fresnel. 2
1.2.2 Modelos de integración de tecnología termosolar para calor de procesos industriales. 4
1.3 Uso del TMY. *5*
2 Objetivos. **7**
3 Funcionamiento de SHIPCal y modificaciones. **8**
3.1 Terminal de introducción de variables. *10*
3.1.1 Estructura. 10
3.1.2 Modificaciones introducidas. 12
3.2 Función SHIPCal. *13*
3.2.1 Bloque 1: inicialización de variables. 13
3.2.1.1 Bloque 1.1. Control de simulación. 13
3.2.1.1.1 Modificaciones en subbloque 1.1. 13
3.2.1.2 Bloque 1.2. parámetros fijos de simulación. 14
3.2.1.3 Bloque 1.3: variables del sistema. 14
3.2.1.3.1 Modificaciones en el subbloque 1.3. 17
3.2.2 Bloque 2: simulación solar. 17
3.2.2.1 Bloque 2.1: variables del proceso. 18
3.2.2.1.1 Modificaciones en el subbloque 2.1. 20
3.2.2.2 Bloque 2.2: integración. 21
3.2.2.2.1 Modificaciones en el subbloque 2.2. 21
3.2.3 Bloque 3: Simulación financiera. 21
3.2.3.1 Incremento del coste de combustible. 21
3.2.3.2 Costes de inversión. 21
3.2.3.3 Costes de mantenimiento. 22
3.2.3.4 Modelos de negocio. 22
3.2.3.4.1 Modelo “llave en mano”. 22
3.2.3.4.2 Modelo “ESCO”. 25
3.2.3.5 Modificaciones en el bloque 3. 26
3.2.4 Bloque 4: generación de gráficos. 27
3.2.4.1 Modificaciones en el bloque 4. 27
3.2.5 Bloque 5: Generación de informes. 27

3.3 Funciones auxiliares a SHIPCal. 28
3.3.1 Función demandCreator2. 28
3.3.2 Función calc_min_year. 30
3.3.3 Calc_day_year. 32
3.3.4 Función DemandData2. 32
3.3.5 SolarData2. 34
3.3.6 Meteo_Data2. 36
3.3.7 SolarEQ_simple2. 36
3.3.7.1 Coordenadas solares. 36
3.3.7.2 Hora solar verdadera. 38
3.3.7.3 Estructura de la función. 38
3.3.8 Función waterFromGrid_v3_min. 40
3.3.9 Función waterFromGrid_trim2. 41
3.3.10 Función arraysMonth2. 41
3.3.11 Arrays_Savings_Month2. 42
3.3.12 Funciones auxiliares para producción y recirculación. 43
3.3.12.1 Función operationSimple. 43
3.3.12.2 Función outputStorageSimple. 48
3.3.12.3 Función outputWithoutStorageSimple. 50
3.3.12.4 Función operationDSG. 50
3.3.12.5 Modificaciones en las funciones auxiliares de producción. 52
3.3.13 Funciones auxiliares para generar gráficos. 53
4 Aplicación web. 55
4.1 Funcionamiento y modificaciones. 55
4.1.1 Carpeta de modelos. 55
4.1.1.1 Modificaciones en los modelos. 56
4.1.1.1.1 Carpeta de simforms. 57
4.1.1.1.2 Carpeta de results. 57
4.1.2 Carpeta de formularios. 57
4.1.2.1 Modificaciones en el formulario. 58
4.1.3 Carpeta de views. 59
4.1.3.1 Modificaciones en views. 60
4.1.3.1.1 Modificaciones en carpeta de simforms. 60
4.1.3.1.2 Modificaciones en carpeta de results. 61
4.1.4 Carpetas de HTML. 61
4.1.4.1 HTML para formularios. 62
4.1.4.1.1 Modificaciones en los HTML de formularios. 62
4.1.4.2 HTML para resultados. 62
4.1.4.2.1 Modificaciones en los HTML de resultados. 63
5 Ejemplo de simulación. 64
5.1 Aspectos que se deben considerar. 65
5.1.1 Irradiancia estable y sin recirculación. 65
5.1.2 Irradiancia inestable y sin recirculación. 66
5.1.2.1 Irradiancia insuficiente en modo horario. 67
5.1.2.2 Irradiancia insuficiente en modo diezminutal. 68
5.1.3 Disminución debido a recirculación. 69
5.2 Simulaciones comparativas. 70
5.2.1 Día con irradiancia estable. 71

5.2.2 Día con irradiancia inestable. 75
5.2.3 Simulación mensual. 77
5.2.4 Simulación anual. 81
6 Conclusiones 84
7 Anexo. 85
7.1 Funciones auxiliares a SHIPCal. 85
7.1.1 Función DemandCreator2. 85
7.1.2 Función Calc_min_year. 87
7.1.3 Función DemanData2. 88
7.1.4 Función SolarData2. 89
7.1.5 Función Meteo_data2. 92
7.1.6 Función SolarEQ_simple2. 93
7.1.7 Función calc_day_year. 95
7.1.8 Función waterFromGrid_v3_min. 95
7.1.9 Función waterFromGrid_trim2. 97
7.1.10 Función ArraysMonth2. 97
7.1.11 Función arrays_Savings_Month2. 102
Referencias 108

ÍNDICE DE FIGURAS

Figura 1: rangos de temperaturas según el proceso industrial. 2
Figura 2: concentrador tipo Fresnel con un único receptor y un concentrador secundario. 3
Figura 3: concentrador tipo Fresnel con doble tubo receptor y sin concentrador secundario. 3
Figura 4: TMY diezminutal de Sevilla. 6
Figura 5: bloque, subbloque y sección. 8
Figura 6: diagrama de flujo del script de SHIPCal.py en caso de simular desde la terminal. 9
Figura 7: diagrama de flujo para simulación con front-end. 9
Figura 8: comienzo del bloque de introducción, tras el código de SHIPCal. 10
Figura 9: creación del vector anual de demanda con *demandCreator*. 15
Figura 10: importación de los datos de *inputsDjango* y carga del fichero TMY en *file_loc*. 15
Figura 11: creación de vector de demanda de simulación. 16
Figura 12: ángulos de incidencia y coordenadas solares. 18
Figura 13: pérdidas en el plano longitudinal. 19
Figura 14: estructura básica del bucle de simulación. 20
Figura 15: diagrama de flujo de *calc_min_year*. 31
Figura 16: diagrama de flujo de función *DemanData2*. 33
Figura 17: diagrama de flujo de *SolarData2* (parte 1). 34
Figura 18: diagrama de flujo de *SolarData2* (parte 2). 35
Figura 19: acimut solar. 37
Figura 20: β representa la elevación y α el acimut. 37
Figura 21: diagrama de flujo de la función *waterFromGrid_v3_min*. 40
Figura 22: diagrama de flujo de la función *waterFromGrid_trim2*. 41
Figura 23: diagrama de flujo de *arraysMonth2*. 42
Figura 24: temperatura de entrada al campo solar 44
Figura 25: temperatura de entrada al campo solar con agua extraída de la red. 44
Figura 26: temperaturas del campo solar. 45
Figura 27: se añade la variable factor a *operationSimple2*. 52
Figura 28: variable *factor* en *operationDSG2*. 53
Figura 29: función *storageSummer2* como ejemplo de aplicación del factor de conversión. 53
Figura 30: definición de un modelo. 55
Figura 31: uso de modelos *Fuel* y *FuelUnits* para agregar un nuevo combustible. 56
Figura 32: vista de la ventana para añadir ubicaciones con TMY. 56
Figura 33: vista final por pantalla de *models.TimeField*. 57
Figura 34: estructura para definir un formulario que se mostrará por pantalla. 58
Figura 35: recogida de las variables procedentes de la base datos y creación de los vectores. 59
Figura 36: variables de inicio y final de simulación según *itercontrol* y *annual* (parte 1). 60
Figura 37: variables de inicio y final de simulación para modo horario (parte 2). 61
Figura 38: cabecera del formulario principal (*simulation_form.html*). 62
Figura 39: cabecera de la ventana de resultados financieros (*imp_finance.html*). 63
Figura 40: demanda y producción con irradiancia insuficiente en modo horario. 67
Figura 41: demanda, producción e irradiancia en modo diezminutal. 67
Figura 42: demanda, producción e irradiancia en modo horario. 68
Figura 43: demanda y producción con irradiancia insuficiente en modo diezminutal. 68
Figura 44: recirculación (verde) en modo horario. 69

Figura 45: recirculación (verde) en modo diezminutal. 70
Figura 46: ilustración de ejemplo. Simulación diezminutal (SL_L_P). 71
Figura 47: ilustración de ejemplo. Simulación horaria (SL_L_P). 71
Figura 48: simulación del 6 de junio de SL_L_S_PH, modo horario. 73
Figura 49: simulación del 6 de junio de SL_L_S_PH, modo diezminutal. 74
Figura 50: simulación diezminutal. Ilustración con SL_L_P de ejemplo. 75
Figura 51: simulación horaria. Ilustración con SL_L_P de ejemplo. 75
Figura 52: producción y demanda en cada mes del año. 80
Figura 53: ahorro solar en cada mes del año. 80

ÍNDICE DE TABLAS

Tabla 1: constante B según tipo de integración. 22
Tabla 2: constante A según el tipo de fluido. 22
Tabla 3: valores de las variables que recibe la función. 31
Tabla 4: variables que recibe la función. 33
Tabla 5: variables para el cálculo de la elevación solar. 39
Tabla 6: variables fijadas para las simulaciones. 65
Tabla 7: irradiancia y ángulos de incidencia longitudinal. 65
Tabla 8: resultados para una hora con irradiancia estable. 66
Tabla 9: irradiancia y ángulos de incidencia longitudinal. 66
Tabla 10: resultados para una hora con irradiancia estable. 66
Tabla 11: resultados para una hora con insuficiente irradiancia para modo diezminutal. 69
Tabla 12: variación de resultados en una hora con recirculación. 69
Tabla 13: resultados para un día con irradiancia estable. 73
Tabla 14: resultados para un día con irradiancia inestable. 77
Tabla 15: resultados para cada mes del año con la integración SL_L_P 79
Tabla 16: resultados de simulación anual. 82

1 INTRODUCCIÓN.

Actualmente, la digitalización de datos e innovación en las aplicaciones web ha permitido que cualquier usuario pueda acceder a un programa de simulación en la web. Bien sea con un fin comercial, docente o de investigación, varias son las empresas y entidades privadas que desarrollan sus modelos de simulación. Las entidades públicas también desarrollan sus propios modelos principalmente con un fin de investigación, aunque puede darse la colaboración con entes privados. Por ejemplo, el programa de simulación de sistemas fotovoltaicos IESPRO [2] permite simular y analizar el comportamiento con distintos tipos de generadores fotovoltaicos. Ha sido desarrollado por el Grupo de Sistemas Fotovoltaicos del Instituto de Energía Solar de la Universidad Politécnica de Madrid. Este programa, escrito en lenguaje de programación Matlab, se utiliza para fines de investigación, para usos profesionales, pues se ha trasferido a empresas del sector fotovoltaico, y para la docencia dentro de la universidad.

En el caso de este trabajo, el modelo SHIPcal es una herramienta creada en 2015 y traducida a lenguaje Python en 2016 que sirve como motor para el front-end del simulador solar de procesos industriales RESSSPI (*Red de Sistemas Solares Simulados para Procesos Industriales)* de la empresa Solatom y de la interfaz de CIMAV del Centro de Investigación de Materiales Avanzados en México. Su motor es la función SHIPCal (*Solar Heat for Industrial Processes Calculator*) que utiliza otras funciones auxiliares y módulos de datos para poder funcionar y está en constante cambio y mejora. Su uso es de acceso libre y se encuentra en repositorio GitHub (*https://github.com/mfrasquet/SHIPcal*).

1.1 Energía térmica solar para procesos industriales.

El potencial que se presenta para la energía solar de concentración es que actualmente la mayoría de los sectores industriales consumen más energía térmica que eléctrica y pueden verse provistos del calor solar para varios de sus procesos productivos. También puede aprovecharse para suplir la demanda de calor en procesos del sector agroalimentario. Actualmente las tres variantes para aprovechar el calor solar en la industria son:

- Colectores solares de aire: se basan en calentar directamente el aire. Se usan generalmente en la industria alimentaria para procesos de secado.
- Sistemas solares de baja temperatura: dedicados al aprovechamiento de la energía solar para calentar agua. Se usan principalmente en residencias y viviendas, aunque puede aprovecharse para procesos industriales consiguiendo temperaturas de 150 º C.
- Sistemas solares de alta temperatura: basado en la reflexión y concentración de la radiación solar en un punto o área por la que circula un fluido caloportador. Para aplicaciones industriales, estos sistemas pueden ser de varios tipos: cilindro-parabólico, disco parabólico, lineal (Fresnel) y receptor central. Pueden alcanzar mayores temperaturas que los otros dos tipos de sistemas termosolares, con máximos en torno a 500 ºC.

Por tanto, la energía solar es una fuente de energía que puede proveer de calor a temperaturas varias, desde temperaturas bajas (menos de 100ºC) a altas (aproximadamente 500ºC). Por otro lado, la demanda de calor en la mayoría de los procesos industriales se sitúa en rangos de temperatura asequibles por los sistemas solares

pues, excepto la minería y la tratamiento térmico, la mayoría de los procesos operan en un rango entre 100 y 200ºC:

INDUSTRIA	PROCESO	RANGO DE TEMPERATURA (°C)
Láctea	Esterilización Secado	100-120 120-180
Comida enlatada	Esterilización	110-120
Textil	Secado, desengrasado Fijado	100-130 160-180
Papel	Blanqueo	130-150
Química	Jabones Caucho sintético Calor de proceso Petróleo	200-260 150-200 120-180 100-150
Subproductos de la madera	Preparación de pulpa	120-170
Desalinización	Incremento de temperatura del fluido caloportador	100-250
Minera	Secado Fundición del concentrado Calentamiento de soluciones Lavado	100-400
Plásticos	Preparación Destilación Separación Extensión Secado Mezclado	120-140 140-150 200-220 140-160 180-200 120-140
Tratamiento térmico	Revenido medio	350-450
Refrigeración	Máquina de absorción de doble efecto	120-190

Figura 1: rangos de temperaturas según el proceso industrial. [3]

Para estos rangos de temperaturas, el sistema más apropiado será el sistema termosolar de concentración de alta temperatura. Cabe recordar que las instalaciones termosolares no son sustitutas de la caldera convencional que se utiliza en el proceso, sino que se complementan a la hora de suplir la demanda de energía del proceso.

1.2 Tecnología termosolar de alta temperatura.

Como se ha mencionado, para aplicaciones industriales los principales sistemas termosolares de alta temperatura son cilindro parabólico, disco parabólico, Fresnel y receptor central. Respecto a centrales de potencia, se usan los ya mencionados en mayor o menor medida, añadiendo además el sistema termosolar de torre o receptor central. El sistema de torre y el sistema de disco parabólico concentra la componente directa de la radiación sobre un punto o área reducida (foco puntual), mientras que el sistema cilindro parabólico y el sistema Fresnel focalizan la radiación en un foco lineal.

1.2.1 Concentradores tipo Fresnel.

El tipo de instalación termosolar para la que simula SHIPCal es un sistema compuesto por concentradores de tipo Fresnel, aunque con pocas modificaciones se puede simular con otros sistemas de concentración. Los

concentradores lineales de Fresnel son instalaciones que utilizan espejos lineales planos o con una pequeña curvatura, dispuestos sobre un eje longitudinal de rotación para seguir el desplazamiento del Sol y concentrar sobre un tubo receptor, situado justo encima de los espejos concentradores.

Los sistemas Fresnel pueden ser de un único tubo receptor con un concentrador secundario o con dos tubos en paralelo, minimizando sombras y bloqueos. La temperatura promedio de operación tiende a ser moderada, aproximadamente a 200ºC y su uso más frecuente es para la generación de vapor. En SHIPCal, el fluido puede ser vapor, agua, aceite térmico o sales fundidas.

En general, el coste de instalación es menor que en otros sistemas solares, dado que la estructura es más ligera, los reflectores son menos costosos que los heliostatos de sistemas de receptor central, disco parabólico o los sistemas cilindro parabólico. Los tubos receptores son más baratos por trabajar a priori a temperaturas medias-bajas. Sin embargo, el hecho de tener esta limitación de temperatura y una relación de concentración no muy elevada conlleva que no sea la mejor opción en cuanto a rendimiento térmico para producir potencia eléctrica con respecto a los otros sistemas de concentración, pero asequible para producir potencia en forma de calor para suministrar a procesos industriales. Actualmente se busca habilitar este tipo de captadores para trabajar a mayores temperaturas, en torno a 500 ºC y extender su uso para plantas de gran potencia. [5]

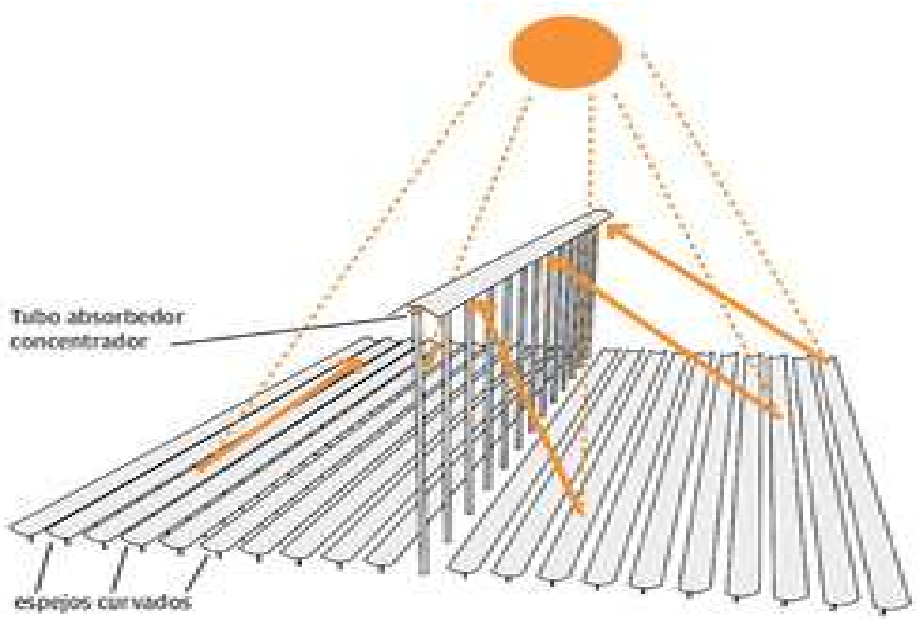

Figura 2: concentrador tipo Fresnel con un único receptor y un concentrador secundario.[6]

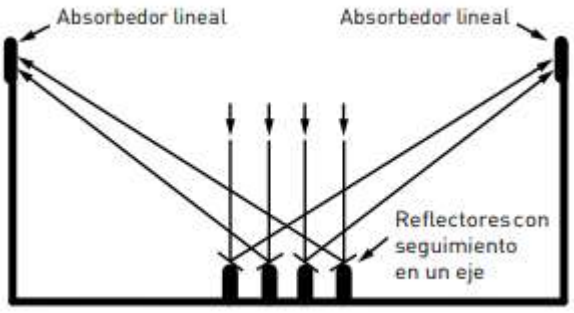

Figura 3: concentrador tipo Fresnel con doble tubo receptor y sin concentrador secundario. [5]

Si se evalúa el rendimiento global de la instalación solar Fresnel, se deben tener en cuenta tres tipos de pérdidas:

- Pérdidas térmicas: se presentan por conducción de calor desde el tubo absorbente hacia la estructura, además de por convección y radiación con el ambiente. Si se habla de rendimiento térmico, la producción de energía se ve afectada por estas pérdidas térmicas, por estar la temperatura del receptor elevada a la cuarta en la ecuación de Stefan-Boltzmann:

$$\eta_{\text{térmico,colectores solares}} \sim \sigma \cdot \varepsilon \cdot (T_{superficie}^4 - T_{ambiente}^4)$$

- Pérdidas geométricas: se deben a la disminución de área efectiva de captación de los espejos. Están causadas principalmente por pérdidas por sombras, bloqueos entre filas de colectores y pérdidas por sistema de seguimiento.

  Las pérdidas por sombra y bloqueos se deben a la posición que ocupan unos espejos de concentración respecto a otros causándose sombras (al recibir irradiancia) y bloqueos (al reflectar) entre sí, aunque en los sistemas Fresnel, este efecto es menor que en otros sistemas termosolares.

  Las pérdidas por seguimiento se deben a que los sistemas Fresnel, junto a los sistemas de cilindro parabólico, se dispone de rotación en un solo eje, que es el eje longitudinal. Para evaluar la rotación respecto a la radiación incidente, se usa el concepto de ángulo de incidencia, explicado más adelante.

- Pérdidas ópticas: se deben a los materiales que intervienen en la transmisión del calor y a las imperfecciones de las superficies de captación y del tubo absorbedor.

1.2.2 Modelos de integración de tecnología termosolar para calor de procesos industriales.

Según el dimensionado y las prestaciones, las instalaciones pueden integrarse de una manera determinada en el proceso industrial para aportar energía en forma de calor. Estas integraciones vienen clasificadas en la guía de integración de la IEA para la producción de calor de procesos y aplicaciones avanzadas (*Solar Process Heat for Production and Advanced Applications: Integration Guideline)*[1]. En esta guía se describen los distintos aspectos para tener en cuenta y las distintas formas de suministrar calor al circuito del proceso. En relación con SHIPCal, el simulador tiene implementado 13 modelos de integración:

- SL_L_P (*Supply level with liquid heat-parallel integration*): el fluido es líquido térmico (L). Cuando se sale del proceso industrial, se extrae el fluido del circuito primario y se dirige hacia el campo solar (circuito secundario). La extracción desde el campo solar de vuelta al circuito principal se hace mezclando el fluido que sale del campo solar con el que sale de la caldera justo antes de entrar en el proceso industrial, ya que el campo solar aporta un salto térmico equivalente al aportado por la caldera, por lo que el fluido se puede suministrar directamente.

- SL_L_PS: añade almacenamiento (S) auxiliar al sistema SL_L_P

- SL_L_S (*Supply level with liquid and storage)*: este sistema añade un acumulador (S) al sistema SL_L_P. El campo solar sirve para calentar dicho almacenamiento, por lo que el campo solar es independiente del circuito primario, separados ambos por el acumulador. Utiliza líquido térmico (L).

- SL_L_S_PH (*Supply level with liquid, storage and preheated)*: similar a SL_L_S, pero el campo solar se usa para precalentar (PH, preheated) el fluido que sale del acumulador y va a la caldera. Por ello, el circuito secundario descargará el fluido en algún punto del circuito primario antes de entrar en la caldera. Utiliza líquido térmico (L).

- SL_L_RF (*Supply level liquid return flow*): precalentamiento del fluido antes de entrar en la caldera, pero sin almacenamiento en el circuito secundario. Para ello, en el modelo de SHIPCal el circuito secundario es independiente del circuito primario al estar separado por un intercambiador. Se controla el caudal de fluido del circuito primario que entra en el intercambiador con una válvula de tres vías. Utiliza líquido térmico (L).

- SL_L_DRF (*Supply level liquid direct return flow*): igual que SL_L_RF, pero no lleva intercambiador y, por tanto, los circuitos no son independientes. Utiliza líquido térmico (L). Aún no implementado.

- SL_S_FW (*Supply level steam feed water*): el fluido es agua. Se utiliza el campo solar para precalentar el agua antes de entrar en la caldera (FW, *feed water*). Para ello, lo normal es instalar la planta solar a la salida del desgasificador.

- SL_S_FWS (*Supply level steam feed water and storage*): similar a SL_S_FW, pero con almacenamiento auxiliar a la salida del campo solar. El fluido es agua.

- PL_E_PM (*Process level external heat exchanger for process medium*): el fluido que sale del campo solar y va al circuito primario también calienta un circuito terciario mediante un intercambiador auxiliar.

- SL_S_MW (*Supply level steam make-up water)*: el campo solar calienta el agua de alimentación de la red que entra nueva al circuito primario, antes de entrar al desgasificador.

- SL_S_MWS (*Supply level steam make-up water and storage)*: similar al SL_S_MW, pero con almacenamiento (S) intermedio.

- SL_S_PD (*Supply level steam parallel and direct)*: el agua que entra al campo solar sale del desgasificador. En el campo solar se obtiene vapor que se introduce en el circuito primario tras la caldera (integración paralela). Es similar a SL_L_P.

- SL_S_PDS (*Supply level steam parallel and direct with storage):* añade un acumulador auxiliar al modelo SL_S_PD.

1.3 Uso del TMY.

El TMY *(Typical Meteorological Year),* es un conjunto de datos meteorológicos de una ubicación geográfica. El formato más frecuente es el formato horario, con datos característicos de cada hora. Los datos suelen seleccionarse en un periodo de larga duración, habiendo TMY con datos de cada hora durante un año completo o con datos de varios años.

Los parámetros característicos suelen ser: irradiancia, temperatura ambiente, humedad relativa y velocidad del viento, entre otros.

Su uso suele ser frecuente en la construcción de edificios por la condicionalidad del clima local con los materiales de construcción. Sin embargo, representan el clima medio y no casos extremos a los que pueden enfrentarse los materiales frente al clima. También se usan para diseño de instalaciones solares.

Para simular en SHIPCal, será necesario pues introducir un TMY con los datos meteorológicos de la ubicación en cuestión. En este trabajo, el TMY será referente a Sevilla. Pudiendo ser diezminutal u horario:

1	MES	DÍA	HORA	MINUTO	Ibn (W/m^2)	Ig0 (W/m^2)	Temp (ºC)	HR (%)	VV (m/s)	DV (º)
25800	6	180	3	40	0,0	0,0	20,8	69,4	1,7	35,4
25801	6	180	3	50	0,0	0,0	20,7	70,0	2,0	31,7
25802	6	180	4	0	0,0	0,0	20,6	70,0	1,9	38,7
25803	6	180	4	10	0,0	0,0	20,6	70,5	1,7	34,4
25804	6	180	4	20	0,0	0,0	20,4	71,0	1,7	28,1
25805	6	180	4	30	0,0	0,0	20,4	71,5	1,3	29,2
25806	6	180	4	40	0,0	0,0	20,2	72,3	1,6	24,2
25807	6	180	4	50	0,0	0,0	20,1	72,9	1,5	30,2
25808	6	180	5	0	0,0	0,0	20,0	73,0	1,8	33,7
25809	6	180	5	10	0,1	1,5	20,0	73,0	1,4	32,6
25810	6	180	5	20	30,7	8,0	19,9	74,0	1,5	34,4

Figura 4: TMY diezminutal de Sevilla.

2 OBJETIVOS.

Los objetivos de este trabajo son:

- Habilitar a SHIPcal para simular en pasos diferentes al paso horario, optando principalmente por paso diezminutal y quinceminutal. Para modificar el código fuente de SHIPCal, se ha recurrido a la herramienta Spyder, (similar a Matlab).

- Habilitar al front-end a simular en pasos diezminutales y quinceminutales. También para simular para periodos menores a un año, pues únicamente está habilitado para simulaciones anuales.

- Comparar las prestaciones y parámetros de una instalación solar de concentración entre el modelo de paso horario y el modelo de paso minutal.

3 FUNCIONAMIENTO DE SHIPCAL Y MODIFICACIONES.

El modelo de SHIPCal se compone de una función formada por 5 bloques en los que se realiza los distintos procesos de la simulación, desde la lectura de datos hasta la generación de informes y gráficas representativas. Cada uno de estos bloques recurre a otras funciones externas a la función SHIPCal para poder simular:

- Bloque 1: inicialización de variables. En este bloque, SHIPCal lee e inicializa las variables de control y librerías privadas para simular. En función de la interfaz desde donde se llame a SHIPcal y de la identidad del usuario se accederá a un paquete de datos determinado. Estos datos introducidos por el usuario comprenden parámetros de localización (TMY), idioma, costes, datos de diseño, etc.
- Bloque 2: simulación solar. Se calculan los parámetros del ciclo energético en función del modo de integración escogido por el usuario para cada paso de simulación o para un año completo.
- Bloque 3: simulación financiera. Se calcula los costes de mantenimiento e inversión, así como otros parámetros económicos como TIR o ahorro de energía.
- Bloque 4: generación de gráficos.
- Bloque 5: generación de informes.

En el script de SHIPCal.py se importan las funciones auxiliares antes de comenzar a definir el código de SHIPCal y se definen los argumentos de entrada al final, en el fragmento denominado como "terminal de introducción de variables". La nomenclatura que se ha escogido para definir la estructura es:

- Bloque: es cada una de las 5 partes que componen SHIPCal.
- Subbloque: cada una de las partes suficientemente diferenciadas que componen un bloque.
- Sección: las partes que componen un subbloque. Éstas únicamente sirven para indicar al usuario a entender lo que se está haciendo en ese fragmento de código en concreto.

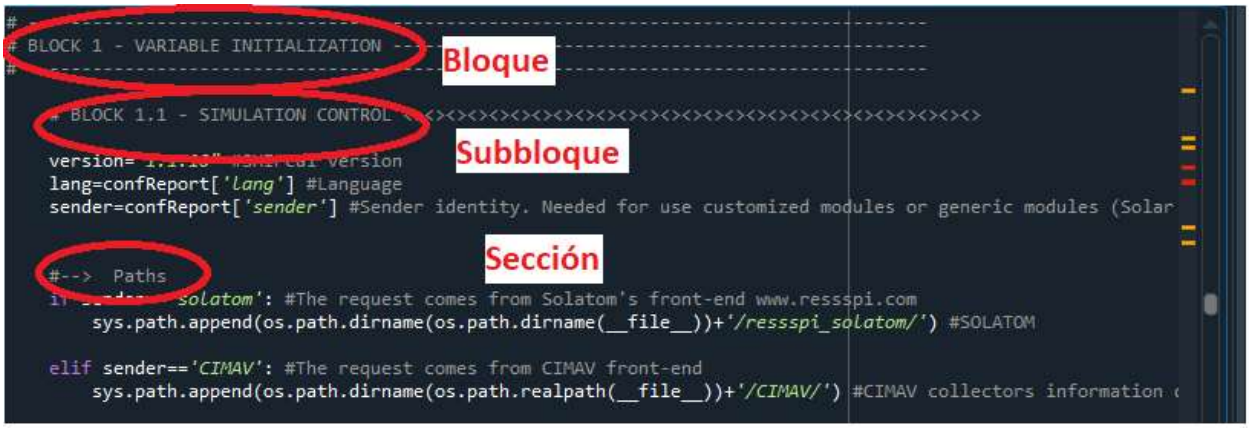

Figura 5: bloque, subbloque y sección.

Dado que SHIPCal es de acceso libre, pero ha sido diseñado principalmente por Solatom y CIMAV, ciertos parámetros, ficheros o funciones de carácter confidencial solo pueden usarse desde los repositorios de Solatom o CIMAV. En caso de ser un usuario particular u otra institución, accederá a funciones, ficheros o datos estándar que podrán cambiarse si se desea desde el PC particular. La identidad del usuario se identifica mediante la variable *sender*.

Otra variable indicativa es la clave llamada *origin*. Mediante esta variable, SHIPCal identificará que se está llamando directamente desde la consola (*origin* igual a 0) o desde un front-end (*origin* distinto de 0).

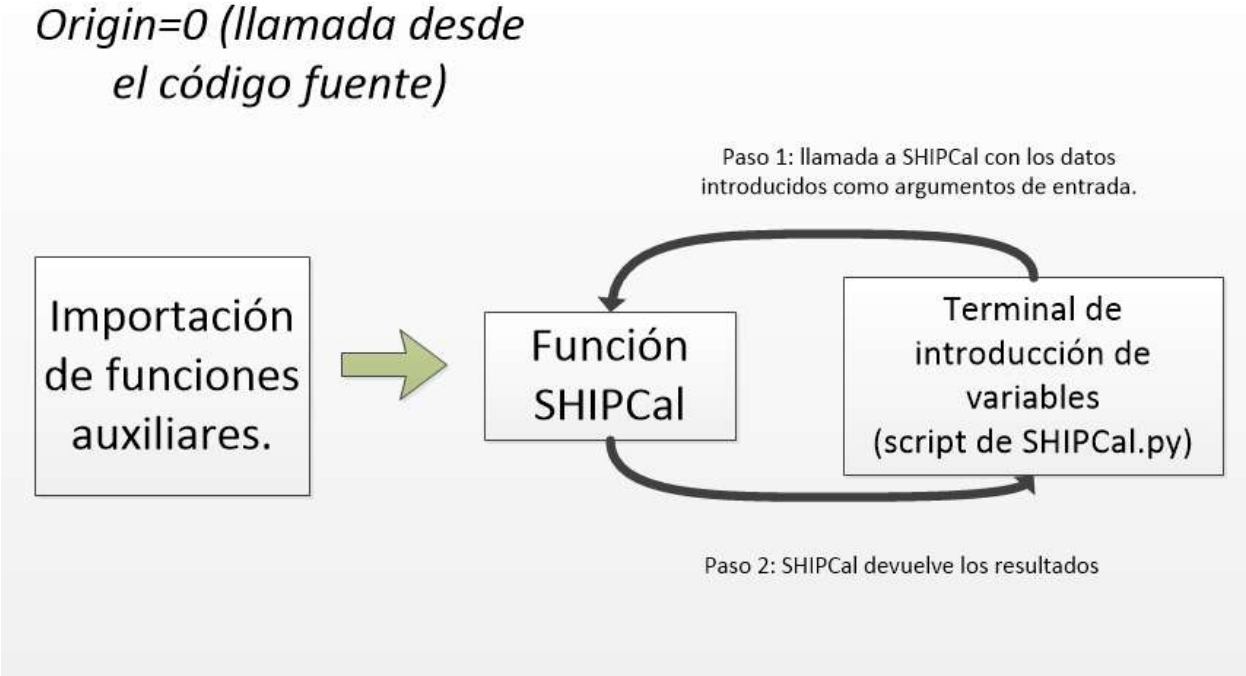

Figura 6: diagrama de flujo del script de SHIPCal.py en caso de simular desde la terminal.

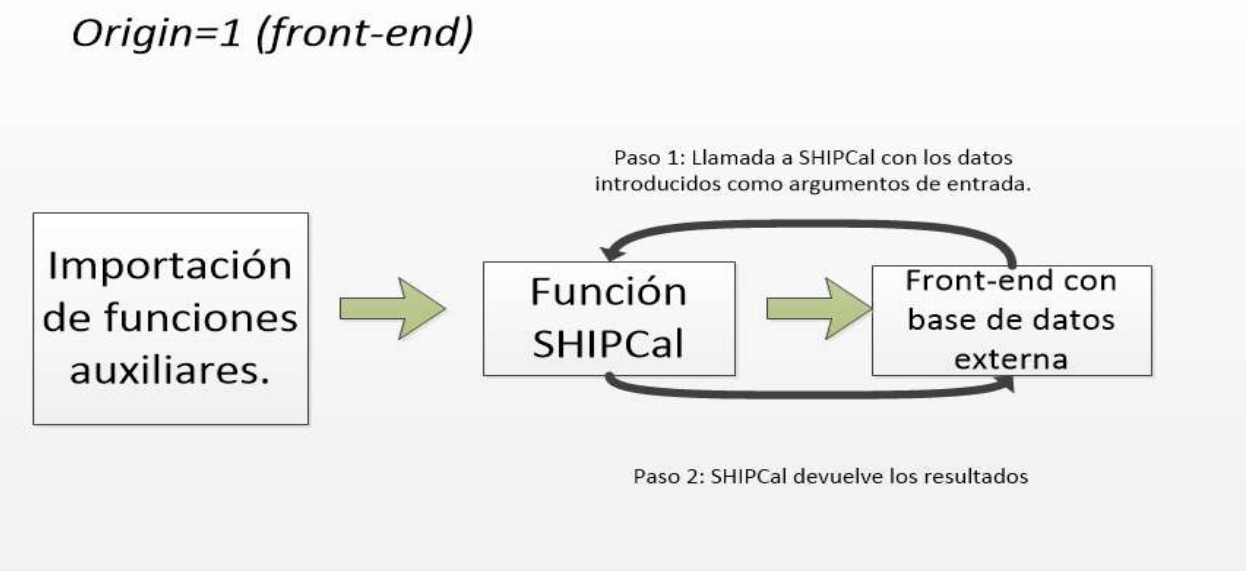

Figura 7: diagrama de flujo para simulación con front-end.

3.1 Terminal de introducción de variables.

La terminal de introducción de variables se refiere al bloque de código situado en el script de SHIPCal y definido después de éste. Cuando se termina de definir el código de SHIPCal tras finalizar el bloque 5, se necesita introducir los parámetros y variables para llamar a SHIPCal si se trabaja directamente desde el código fuente. En caso de utilizar un front-end, las variables se introducen de manera externa.

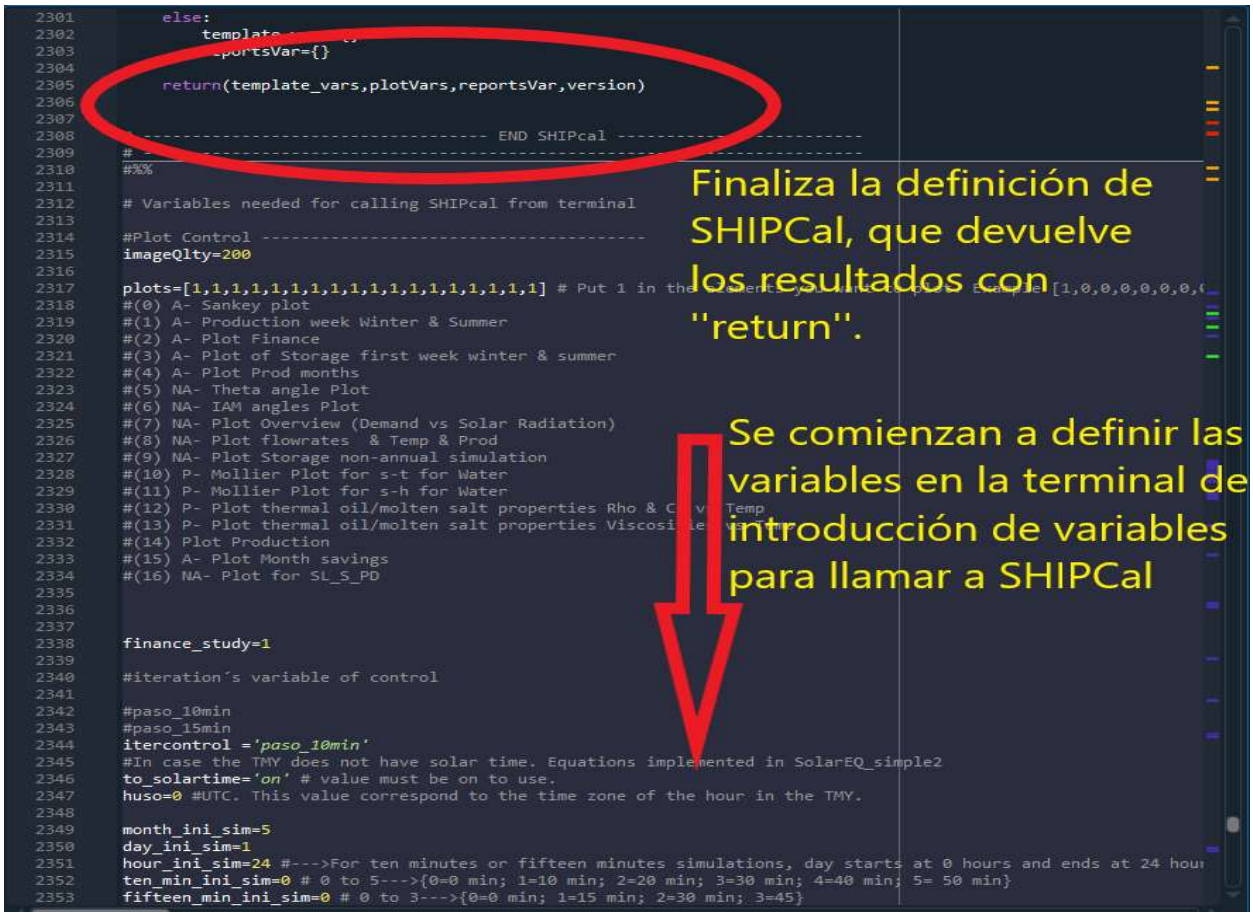

Figura 8: comienzo del bloque de introducción, tras el código de SHIPCal.

3.1.1 Estructura.

Este fragmento de código se compone de:

-*Plots*: variable de tipo vector compuesta de unos o ceros en función de los gráficos que el usuario requiera de SHIPCal. Cada componente del vector se refiere a un gráfico determinado y si el valor de dicha componente es 1, SHIPCal desarrollará el gráfico asociado.

-*Finance_study*: variable que debe valer 1 si se quiere realizar estudio financiero.

-Variables de inicio y finalización de simulación: estas variables son de tipo entero y son introducidas por teclado y determinan el intervalo de simulación y el número de iteraciones que se realizará. Son las siguientes:

- *Month_ini_sim*: variable que señala el mes en el que se inicia la simulación y su valor está entre 1 y 12.

- *Day_ini_sim*: variable que señala el día en el que se inicia la simulación y su valor está entre 1 y el último día del mes introducido en *Month_ini_sim*.
- *Hour_ini_sim*: variable que señala la hora en la que empieza la simulación y su valor está entre 1 y 24.
- *Month_fin_sim*: variable que señala el mes en el que se finaliza la simulación y su valor está entre 1 y 12. Debe ser mayor o igual que *Month_ini_sim.*
- *Day_fin_sim*: variable que señala el día en el que se finaliza la simulación y su valor está entre 1 y el último día del mes introducido en *Month_fin_sim*. Debe ser mayor o igual que el valor introducido en *Day_ini_sim.*
- *Hour_fin_sim*: variable que señala la hora en la que finaliza la simulación y su valor está entre 1 y 24. El valor introducido deberá ser mayor o igual a la introducida en *Hour_ini_sim.*

-Parámetros de ajuste de simulación: se utilizan para ofrecer la posibilidad de incorporar modificadores a la simulación, pudiendo valer entre 0 y 1, siendo por tanto una variable flotante. Los modificadores son variables que se utilizan para ajustar manualmente los valores de coste de inversión (variable *mofINV*), producción energética (*mofPROD*) e irradiancia (*mofDNI*). Si por ejemplo se establece 0.5 en *mofPROD,* la producción se reduce un 50%.

-Parámetros de dimensionado de planta: estas variables se refieren al número de lazos (*num_loops*), el número de colectores por lazo (*n_coll_loop*), el tipo de integración de la IEA (*type_integration*) y la capacidad en litros del almacenamiento (*almVolumen*). El tipo de integración se clasifica según la task 49 de la IEA [1] explicada en la introducción.

-Listado de variables: estos diccionarios recogen las variables que se introducirán en SHIPCal, algunas definidas anteriormente, por ejemplo, en el diccionario *designDict* se recogen los parámetros de dimensionado de planta. Otras se definen directamente dentro del listado, por ejemplo: en el diccionario *confReport* se definen el idioma, el usuario y la cabecera. En total son 4:

- *ConfReport:* recoge idioma (*lang*) e identidad del usuario (*sender*) y cabecera del informe (*cabecera*).
- *Modificators:* recoge las variables de ajuste *mofPROD*, *movINV* y *mofDNI*.
- *DesignDict:* recoge las variables de dimensionado de planta.
- *SimContol:* recoge *finance_study*, las variables de inicio y finalización de simulación, *to_solartime*, *huso* e *itercontrol*.

La variable *sender* para control de identidad puede valer:

- CIMAV: cuando la simulación la realiza CIMAV.
- Solatom: la simulación la realiza Solatom desde su interfaz: RESSSPI.
- SHIPcal: sender toma este valor automáticamente cuando se simula desde un front-end que no sea RESSSPI o CIMAV.
- Cualquier otro valor en el caso de que se llame directamente desde el código fuente.

-*Origin:* esta variable es el identificativo de la interfaz desde donde se llama a SHIPCal. En función de su valor, se cargarán unos parámetros de funcionamiento y diseño determinados según esté configurada la interfaz desde la que se opera. Estos parámetros pueden ser: precio de combustible, demanda, país, tasas de CO_2, etc. En caso de trabajar con front-end, se almacenan en un diccionario llamado *inputsDjango.* En caso contrario, están definidos dentro de SHIPCal y no se introducen desde la terminal de introducción de variables.

- Si el valor de *origin* es 0, significa que se ejecuta SHIPCal desde el código fuente y no desde un front-end. Para este caso, *inputsDjango* no contiene valores y se introducen manualmente dentro de SHIPCal.
- Si *origin* vale 1, se ejecuta desde un front-end externo particular.
- Si el valor es -2, significa que se ejecuta desde RESSSPI.
- Si vale -3, se llama desde el front-end de CIMAV.

Por último, se llama a SHIPCal con variables y listados anteriores definidos y finaliza la terminal de introducción de variables:

[jSonResults,plotVars,reportsVar,version]=SHIPcal(origin,inputsDjango,plots,imageQlty,confReport,modificators,desginDict,simControl,last_reg)

3.1.2 Modificaciones introducidas.

En primer lugar, se define la variable *itercontrol*, una variable de control de modo de simulación que servirá para elegir qué tipo de simulación (horaria, quinceminutal o diezminutal) se elegirá. En función de la cadena de caracteres se tomará un modo u otro:

- *'Paso_10min*:' valor que se le asigna a *itercontrol* si se requiere simular en paso diezminutal. Es una modificación para paso diezminutal**.**
- *'Paso_15min'*: valor que se le asigna a *itercontrol* si se requiere simular en paso quinceminutal. Es una modificación para paso quinceminutal**.**
- Si la cadena de caracteres no es igual que las dos anteriores SHIPCal simulará en modo horario.

Tras esta variable se crean *to_solartime* y *huso* que son dos variables nuevas para el cálculo de hora solar, explicadas en la introducción. A continuación, se añaden 4 variables nuevas de inicio y finalización de simulación. Las dos variables para paso diezminutal tendrán valores que van de 0 a 5, pues el TMY tiene 6 pasos por hora que empiezan por 0 y acaban en 50 minutos. Si la variable es 0, se leerá la fila con 0 minutos. Puede entenderse que en la fila de 0 minutos se reflejan los datos de 0 al minuto 10. En la siguiente fila, de 10 al minuto 20 y así hasta llegar a la fila 5, que refleja el tiempo de 50 a 60 minutos y se sumarán así 6 pasos y completándose 1 hora:

- *Ten_min_ini_sim*: esta variable señala la porción diezminutal (respecto a 1 hora) en la que se empieza la simulación.

- *Ten_min_fin_sim*: esta variable señala la porción diezminutal (respecto a 1 hora) en la que se finaliza la simulación.

Las dos variables que se añaden para paso quinceminutal pueden valer entre 0 y 3 para sumar 4 pasos en una hora. Si, por ejemplo, se le asigna un valor de 3, comienza a simular en el minuto 45.

- *Fifteen_min_ini_sim*: señala la porción quinceminutal (respecto a 1 hora) en la que se inicia la simulación.
- *Fifteen_min_fin_sim*: señala la porción quinceminutal en la que se finaliza la simulación.

Dado que en el TMY para paso diezminutal o quinceminutal la columna de horas comienza en 0 para cada día, se debe introducir 0 si se quiere simular desde la primera hora del día, a diferencia del modo horario, que empezaba en 1. Las variables de control de ajuste para modificar la simulación y las variables de dimensionado de planta no requieren cambios. Por último, para que SHIPCal pueda recibir las nuevas variables introducidas, se creará la opción de que uno de los diccionarios, el denominado *SimControl,* recoja las nuevas variables de inicio y fin de simulación en función de si se usa paso diezminutal o quinceminutal según el valor de la variable *itercontrol.* También recogerá las variables *to_solartime* y *huso*.

3.2 Función SHIPCal.

Tras haber modificado el bloque de introducción de variables, se procede a modificar la función SHIPCal, que como se ha dicho, se compone de 5 bloques. A continuación, se detalla la estructura de cada bloque y las modificaciones incluidas.

3.2.1 Bloque 1: inicialización de variables.

Se cargan los datos introducidos en la terminal de introducción de variables.

3.2.1.1 Bloque 1.1. Control de simulación.

En este subbloque se leen las variables introducidas en la terminal de introducción de variables o en el front-end referentes a identidad (*sender*) e idioma (*lang*), se identifica desde donde se llama a SHIPCal mediante la variable *origin* y se cargan las librerías privadas. Tras esto, se leen las variables importadas de inicio y finalización de simulación.

3.2.1.1.1 Modificaciones en subbloque 1.1.

Como se ha dicho, en el subbloque 1.1 se leen las variables de inicio y fin de simulación. Estas variables se leen desde el diccionario *SimControl* creado en la terminal de introducción de variables y es necesario añadir las 4 nuevas variables *ten_min_ini_sim, ten_min_fin_sim, fifteen_min_ini_sim* y *fifteen_min_fin_sim* dentro de SHIPCal.

3.2.1.2 Bloque 1.2. parámetros fijos de simulación.

En este subbloque se introducen otros parámetros de operación que no cambiarán en ningún momento de la simulación, por ejemplo: la eficiencia de la caldera, límite mínimo de radiación para que el campo solar funcione o caudal mínimo de las bombas. En caso de usar un front-end, se introducen externamente, aunque actualmente algunas variables solo pueden introducirse desde el código fuente. No se requieren cambios para el objetivo de este trabajo, únicamente se cambian las variables según sean las características de la simulación en concreto.

3.2.1.3 Bloque 1.3: variables del sistema.

Se leen los parámetros de ajuste de simulación y las variables de dimensionado de planta introducidas en la terminal de introducción de variables o desde el front-end. En este subbloque también se cargan los parámetros del colector solar en función de la identidad del usuario (definida en la variable *sender* en la terminal de introducción de variables).

Si el *sender* es CIMAV o Solatom, se utiliza una función concreta a la que solo se puede acceder cargando un paquete de funciones confidencial. Si el *sender* es cualquier otro, los parámetros se deben introducir manualmente. Los parámetros característicos son: tipo de colector, tubo receptor, área captación, rendimiento óptico pico y longitud de cada módulo.

Con los datos cargados hasta ahora, se calcula el área total de captación y el número de módulos sabiendo el número de lazos, el número de colectores y el área de cada colector:

$$Area_{total} = Area_{colector} \cdot N\º colectores_{lazo} \cdot N\º lazos$$

A continuación, en la sección *Front-end inputs* en función desde dónde se llama a SHIPCal (definido por la variable *origin)* se cargará la latitud (*lat*) de la ubicación geográfica y la asignación de la dirección dentro del repositorio de SHIPCal del fichero TMY, guardándola en la variable *file_loc*.

También se cargará el tipo de integración y la capacidad de almacenamiento recogidos en el diccionario *designDict* (importado desde el front-end o desde el bloque de introducción de variables). En caso de no usar front-end (*origin* igual a cero) se introduce manualmente el fluido de trabajo, las temperaturas de entrada y salida del campo solar, la presión de operación, consumo anual demandado por la caldera (*totalConsumption*), precio del combustible, coste de la tonelada de CO_2 producido y modelo de negocio (actualmente "llave en mano" o modelo ESCO). Si se está utilizando un front-end (*origin* distinto de cero) se cargarán los datos anteriormente citados, pero desde el diccionario importado *inputsDjango*.

Los datos de *inputsDjango* se procesan con una función llamada *djangoReport* que los devuelve en un diccionario llamado *inputs,* además de devolver el consumo demandado por la caldera, la presión de operación y lo vectores ponderados de demanda.

- Vectores ponderados de demanda: *weekArray, monthArray, dayArray*. Cada uno de estos vectores temporales suma la unidad y reflejan la fracción de demanda de cada hora respecto al día (*dayArray*), lo que se consume en cada día respecto a la semana (*weekArray*) y lo porción que consume cada mes respecto al año (*monthArray*). En caso de simular desde un front-end, se construirán en la función *djangoReport* y en caso de simular directamente desde la terminal se introducirán manualmente. Por ejemplo, en el caso de *weekArray* estará formado por 7 componentes. El valor de cada día puede ser 1/7 en caso de que los 7 días de la semana haya demanda y sea la misma, o x/N, siendo N el número de días

de demanda (al ser menor que 7 significa que hay días con demanda 0) y x la fracción de la demanda de cada día determinado.

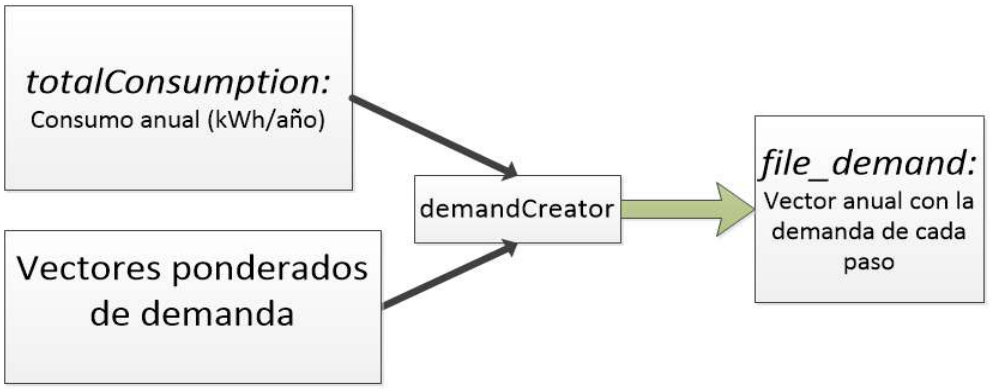

Figura 9: creación del vector anual de demanda con *demandCreator*.

Tras la asignación de valores a las distintas variables, se accede la función *demandCreator*. La función *demandCreator* se usa para calcular el vector de la demanda de energía en el tiempo de 1 año. Recibe los vectores ponderados de demanda y el consumo total del año para crear un vector (*file_demand*) con 8.760 componentes cuyos valores son la energía demandada en cada iteración horaria. La energía se introduce mediante la variable *totalConsumption,* que representa la energía total demandada por la caldera en el año expresado en kWh. Esta función se debe modificar para conseguir un vector de 35.040 componentes en caso de ser paso quinceminutal o 52.560 en caso de ser diezminutal. Para ello se creará la función *demandCreator2*, que surge de modificar la función *demandCreator*.

```
838         [inputs,annualConsumptionkWh,P_op_bar,monthArray,weekArray,dayArray]=djangoReport(inputsDjango)
839     ## METEO (free available meteo sets)
840     locationFromFrontEnd=inputs['location']
841
842     if sender=='solatom': #Use Solatom propietary meteo DB. This is only necessary to be able to use solatom da
843         meteoDB = pd.read_csv(os.path.dirname(os.path.dirname(__file__))+"/ressspi_solatom/METEO/meteoDB.csv",
844         file_loc=os.path.dirname(os.path.dirname(__file__))+"/ressspi_solatom/METEO/"+localMeteo
845         Lat=meteoDB.loc[meteoDB['meteoFile'] == localMeteo, 'Latitud'].iloc[0]
846         Huso=meteoDB.loc[meteoDB['meteoFile'] == localMeteo, 'Huso'].iloc[0]
847     elif sender=='SHIPcal':
848         from simforms.models import Locations
849         file_loc = locationFromFrontEnd
850         localMeteo= Locations.objects.get(pk=locationFromFrontEnd).city
851         Lat = Locations.objects.get(pk=locationFromFrontEnd).lat
852         Huso = 0 #Not used currently, it is assumed to have solar hourly data
853         #meteo_data.order_by('hour_year_sim').values_list('DNI',flat=True)
854         long = Locations.objects.get(pk=locationFromFrontEnd).lon
855     else:
856         meteoDB = pd.read_csv(os.path.dirname(__file__)+"/Meteo_modules/meteoDB.csv", sep=',')
857         localMeteo=meteoDB.loc[meteoDB['Provincia'] == locationFromFrontEnd, 'meteoFile'].iloc[0]
858         file_loc=os.path.dirname(__file__)+"/Meteo_modules/"+localMeteo
859         Lat=meteoDB.loc[meteoDB['meteoFile'] == localMeteo, 'Latitud'].iloc[0]
860         Huso=meteoDB.loc[meteoDB['meteoFile'] == localMeteo, 'Huso'].iloc[0]
861         long=meteoDB.loc[meteoDB['meteoFile'] == localMeteo, 'Long'].iloc[0]
862     ## INTEGRATION
863     type_integration=desginDict['type_integration'] # Type of integration scheme from IEA SHC Task 49 "Integrat
864     almVolumen=desginDict['almVolumen'] # Storage capacity [litres]
865
866     ## INDUSTRIAL APPLICATION
867         #>> PROCESS
868     fluidInput=inputs['fluid'] #Type of fluid
869     T_process_in=inputs['outletTemp'] #HIGH - Process temperature [ºC]
870     T_process_out=inputs['inletTemp'] #LOW - Temperature at the return of the process [ºC]
871     P_op_bar=P_op_bar
```

Figura 10: importación de los datos de *inputsDjango* y carga del fichero TMY en *file_loc.*

Como se observa en la ilustración, tras llamar a la función *djangoReport* que recibe el diccionario *inputsDjango* se devuelven los parámetros en el diccionario *inputs.* En la zona redondeada, se observa cómo se dan los valores a las variables de dimensionado de planta con las que trabajará SHIPCal desde el diccionario importado *designDict* y se crean las variables de operación del fluido con los parámetros en el diccionario *inputs*.

La imagen de la figura anterior hace referencia a *origin* 1 y como se observa, en la variable *file_loc* se carga la ubicación del archivo TMY para poder acceder a ella y leerla.

Tras la carga de latitud y longitud, los datos de *inputsDjango* (en caso de no utilizar un front-end se introducen a mano) y el cálculo del vector de demanda anual con *demandCreator*, se calcula el vector de demanda de simulación.

El vector de demanda de simulación (*Energy_Before*) se refiere a un vector con tantas componentes como pasos tenga la simulación requerida. Para ello, se llama a la función auxiliar *DemandData.* Tras calcular el vector de demanda de simulación, se aplica la eficiencia de la caldera en la sección *Demand of energy before the boiler* para obtener la energía real que se llega al proceso en cada iteración.

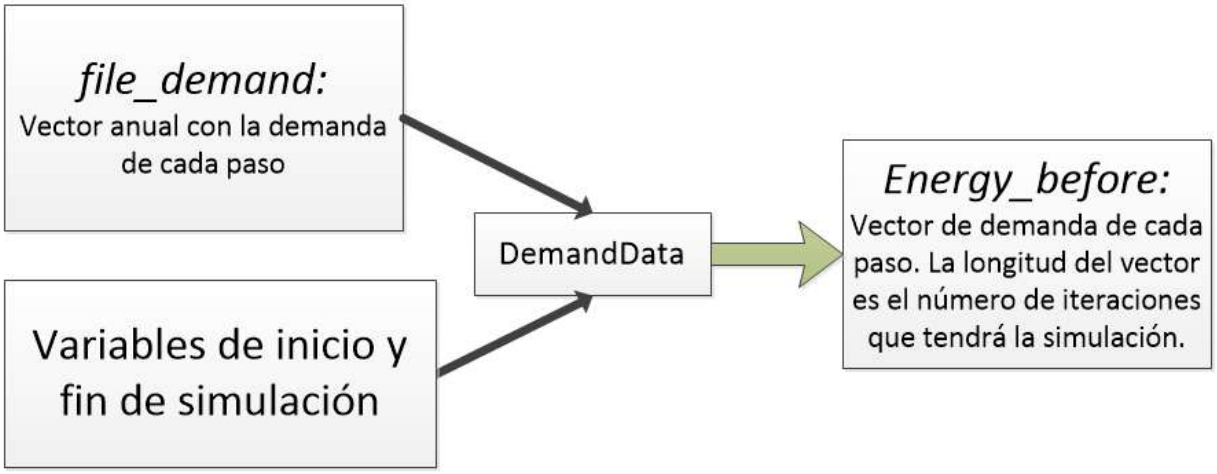

Figura 11: creación de vector de demanda de simulación.

A continuación, se necesita crear sendos vectores con los datos meteorológicos y parámetros solares del fichero TMY cargado. Para ello se recurre a *SolarData*. Esta función recibe la ubicación del archivo TMY, las variables para cálculo de hora solar y las variables de inicio y final de simulación para devolver una única matriz *(output)* con todos los datos temporales y solares para poder simular. Estos datos se copiarán en vectores listos para usarse en el bloque 2 para calcular las variables termodinámicas, producción, IAM, etc.

- *Output* [0]: meses en los que transcurre la simulación.
- *Output* [1]: días en los que se sitúa la simulación.
- *Output* [2]: horas de cada día de la simulación.
- *Output* [3]: contador de horas respecto al año.
- *Output* [4]: hora angular.
- *Output* [5]: elevación solar.
- *Output* [6]: acimut solar.
- *Output* [7]: declinación solar.

- *Output* [8]: ángulo cenital.
- *Output* [9]: irradiancia.
- *Output* [10]: temperatura ambiente.
- *Output* [11]: contador de pasos de simulación respecto al año.

El último paso en este bloque será calcular un vector con la temperatura del agua de la red. Este vector se utilizará en los modelos de integración que operen en circuito abierto. Para ello se accede a la función *waterFromGrid_v3* y *waterFromGrid_trim.* La función *waterFromGrid_trim* devuelve un vector con tantas componentes como pasos de simulación haya y con una misma temperatura determinada para las componentes de un mismo día.

3.2.1.3.1 Modificaciones en el subbloque 1.3.

Las modificaciones de funciones auxiliares se explican más adelante, en el apartado 3.3. Independientemente de si se usa front-end o no, en ambos casos se necesita leer la longitud (variable *long*) de la ubicación de simulación si se quiere calcular la hora solar y también se necesita llamar a la función *demandCreator2,* como modificación de la función *demandCreator.* Además, se necesita los dos nuevos vectores ponderados de demanda, que se establecerán con la misma demanda para cada cuarto de hora o fracción diezminutal:

$$tenmin_{array} = \left[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right]$$

$$fifteenmin_{array} = \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]$$

Tras terminar la sección *front-end inputs,* es necesario añadir la función *DemanData2*, tras modificar la función *DemanData* para calcular el vector de demanda de simulación con la extensión apropiada según el modo de simulación. También se deberá modificar la función *SolarData* y las funciones auxiliares a las que ésta llama incluir el cálculo de la hora solar y modificar las extensiones de los vectores y matrices para funcionar con paso diezminutal o quinceminutal.

Se debe tener en cuenta que la matriz devuelta por *SolarData* es distinta según si se usa simulación horaria o diezminutal/quinceminutal pues en estos dos últimos modos tiene una columna más con los minutos, por tanto, las columnas de 4 a 12 no corresponden al mismo parámetro que en paso horario. Las funciones de cálculo de la temperatura del agua de red también deben modificarse.

3.2.2 Bloque 2: simulación solar.

En este bloque se empieza calculando las variables termodinámicas de inicio de simulación a partir de los datos introducidos manualmente (*origin*=0) o con *InputsDjango*, los modificadores de ángulo de incidencia y se realiza la simulación. Por ello, en función de la integración, temperaturas del proceso, demanda y fluido escogido, los resultados de la simulación tendrán unos valores u otros.

3.2.2.1 Bloque 2.1: variables del proceso.

En este subbloque se comienza calculando las variables termodinámicas de entrada y salida del proceso industrial según el fluido caloportador (*fluid_input*) y el tipo de integración industrial definido con la variable *type_integration*. También se calculan las variables del intercambiador de calor o acumulador en caso de haberlos. Para el acumulador se calcula la energía máxima capaz de almacenarse y el almacenamiento inicial en función de si usan sales fundidas, aceite o agua.

La segunda parte del subbloque es la simulación. En la primera iteración de la simulación, siempre se considera que el proceso de captación solar está parado y las temperaturas de entrada y salida al campo solar serán la temperatura ambiental leída en el TMY en caso de ser circuito cerrado o la de agua de la red en caso de ser circuito abierto y usar agua.

A continuación, se recorre el bucle de simulación. Como se dijo en la introducción, para evaluar la rotación respecto a la radiación incidente, se usa el concepto de ángulo de incidencia. Por tanto, lo primero que se calcula en cada iteración son los ángulos de incidencia longitudinal y transversal con las coordenadas solares (elevación, acimut y cenit), calculados con la función *SolarEQ_simple* (función auxiliar a *SolarData*) devueltos en la matriz *output.* También es necesario la inclinación y orientación del campo solar.

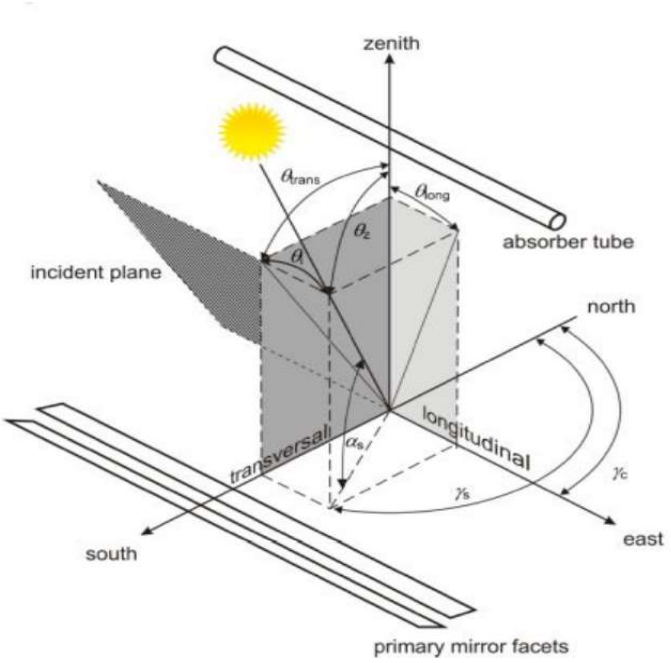

Figura 12: ángulos de incidencia y coordenadas solares. [7]

El ángulo de incidencia es el formado por la irradiancia directa y la normal del espejo concentrador. Así, la máxima captación por parte del colector será cuando el ángulo de incidencia sea 0, y la radiación tenga la dirección de la normal al espejo. Dado que existe seguimiento para la componente en el plano transversal (rotación con eje longitudinal), esta componente sí puede aprovecharse de manera óptima, pero en la componente longitudinal de la radiación no se conseguirá una captación eficiente, especialmente en los extremos del colector:

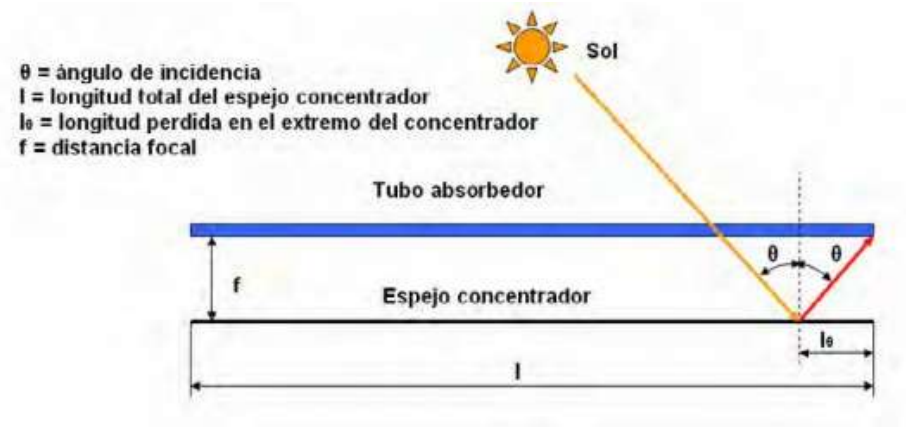

Figura 13: pérdidas en el plano longitudinal. [8]

Como se observa, hay un tramo de tubo receptor que no recibirá radiación. Además de en los extremos, el hecho de que la radiación no llegue en dirección normal a los espejos afecta a parámetros ópticos del conjunto, como pueden ser reflectividad, transmisividad y absortividad. Para reflejar el efecto de los ángulos de incidencia en estas pérdidas ópticas, se utiliza el denominado modificador de ángulo de incidencia, que se llamará IAM (incidence angle modificator) en SHIPCal y se utilizará un IAM para el ángulo transversal y otro para el ángulo longitudinal. Este modificador vale 1 cuando el ángulo de incidencia es 0º y vale 0 cuando el ángulo de incidencia vale 90º. Para calcular los ángulo de incidencia, se usarán las siguientes ecuaciones:

$$\theta_{long} = arccos\sqrt{1 - (cos(SUN_{ELV} - beta) - cos(SUN_{ELV}) \times cos(beta) \times (1 - cos(SUN_{AZ} - orient)))^2}$$

$$\theta_{transv} = arctan\ \frac{(cos(SUN_{ELV}) \times sen(SUN_{AZ} - orient))}{sen(SUN_{ELV} - beta) + sen(beta) \times cos\ (SUN_{ELV}) \times (1 - cos(SUN_{AZ} - orient)}$$

Donde *beta* es la inclinación y *orient* la orientación y actualmente tienen valor 0 pues no están implementados en SHIPCal. Seguidamente, para poder cuantificar las pérdidas, se calcula el modificador de ángulo de incidencia (IAM), definido como el producto del IAM transversal y longitudinal, calculados mediante raytracing:

$$IAM = IAM_{LONGITUDINAL} \cdot IAM_{TRANSVERSAL}$$

Por último, tras calcular los IAM, se calcula en función del tipo de integración y el estado de operación (producción, recirculación o parada) la energía producida, energía útil, pérdidas térmicas, caudal, etc., según sea la demanda, los IAM y el resto de los parámetros calculados o establecidos en el bloque 1. Para estos cálculos se usan las funciones auxiliares explicadas en el apartado 3.3. Los tres modos que adopta el campo solar según sean las condiciones de operación en cada iteración de simulación son:

- Modo parado: este modo se da cuando en la iteración actual la elevación solar es menor que 0 o la irradiancia es menor que el mínimo impuesto en el subbloque 1.2.
- Modo producción: la elevación solar es mayor que 0, la irradiancia es mayor que el mínimo impuesto en el subbloque 1.2 y el caudal en ese instante es mayor que el mínimo impuesto por la bomba, también fijado en el subbloque 1.2.

- Modo recirculación: se cumplen todos los requisitos del modo "producción" excepto la condición de caudal superior al mínimo impuesto por la bomba.

Tras haberse establecido el estado de operación, se elegirá unas funciones auxiliares u otras dentro de las funciones de producción/recirculación o parada según sea un proceso con almacenamiento o no:

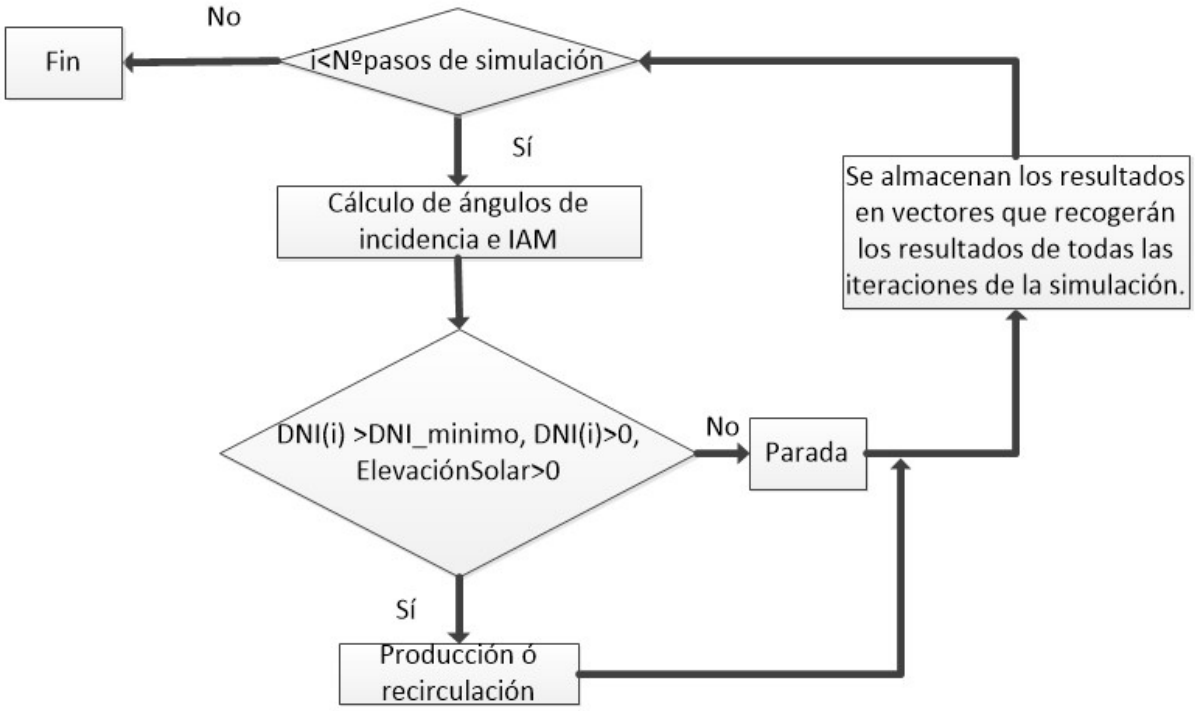

Figura 14: estructura básica del bucle de simulación.

3.2.2.1.1 Modificaciones en el subbloque 2.1.

Debe incluirse una corrección en el cálculo, para poder obtener los kWh en las funciones auxiliares de cálculo de energía producida, pérdidas y energía desenfocada. Esto se debe a que el concepto de kWh conlleva que los W o kW que se calculan en cada iteración en la simulación horaria equivalen a los kWh de cada hora. Pero en la simulación de pasos subhorarios, la potencia de cada iteración diezminutal o quinceminutal no equivale a los kWh reales de esa porción minutal, sino que reflejarían los kWh que se producirían si durante una hora completa se tuviera esa potencia. Por ello, para conseguir magnitudes de kWh a partir de la potencia calculada (W o kW), se debe dividir la potencia entre 6 si se simula con modo diezminutal o entre 4 si se simula en modo quinceminutal.

También debe modificarse el cálculo de ángulo de incidencia para el paso horario. Esto se debe a que con los datos del TMY horario, el ángulo de incidencia que se calcula para una hora es justo en ese instante, pero para que la simulación horaria sea más precisa y se asemeje al modo minutal, debe tomarse un ángulo de incidencia promedio de la hora. Para ello, se tomará como la media entre el ángulo del instante actual y el instante anterior del TMY horario. Por ejemplo, si resulta que en la simulación horario, a las 17.00 el ángulo de incidencia longitudinal vale 45º, y el ángulo de las 16.00 es 52º, el ángulo que se tomará para la hora 16.00-17.00 será 48.5º.

3.2.2.2 Bloque 2.2: integración.

Para generar informes y algunos gráficos de las simulaciones, se necesita la suma de los componentes de los vectores resultantes en la simulación y obtener cantidades totales de energía producida, perdidas, toneladas de CO_2 ahorrado...

3.2.2.2.1 Modificaciones en el subbloque 2.2.

En este bloque se hace la suma de toda la energía (kWh) de la irradiancia existente a lo largo del tiempo de simulación. Al igual que en el bloque anterior, para el modo horario es inmediato y la suma de unidades de potencia (kW) equivale a unidades de energía (kWh), pero en simulaciones subhorarias, se debe dividir la suma obtenida por un factor que será 6 (diezminutal) o 4 (quinceminutal) para tener en cuenta la conversión de kW a kWh.

3.2.3 Bloque 3: Simulación financiera.

Se realiza estudio financiero en el caso de que se realice una simulación anual y que la variable *finance_study* tenga valor 1. En este bloque se calculan costes de inversión, payback, ahorros por uso de recurso solar, amortizaciones, etc.

3.2.3.1 Incremento del coste de combustible.

El incremento del coste de combustible se establece como un incremento debido al IPC y un incremento del propio combustible. Ambos parámetros están en el subbloque 1.2 y actualmente solo se pueden cambiar desde el código fuente.

$$CostRaise\ [€] = \frac{IPC}{100} + \frac{FuelCostRaise}{100}$$

3.2.3.2 Costes de inversión.

Se definen tres costes de inversión. Las ecuaciones de costes están definidas por experimentación en la empresa SOLATOM:

1. El coste de adquisición, transporte e instalación de cada módulo colector se define como:

$$C_{módulo}[€] = 6.410 \cdot \left(\frac{Nº_{colectores}}{4}\right)^{-0.0401}$$

2. El coste del bloque de potencia, referido al bloque hidráulico. Dependerá del tipo de integración industrial escogido:

$$C_{bloque\ de\ potencia}[€] = B \cdot \left(\frac{Nº_{colectores}}{4}\right)^{-0.804}$$

SL_L_P	$\boldsymbol{B = 7.170}$
SL_L_RF	$B = 7.170$
SL_S_FW	$B = 7.170$
SL_S_PD	$B = 12.573$
SL_L_S	$B = 7.170$
Otro tipo de integración	$B = 0$

Tabla 1: constante B según tipo de integración.

3. Coste de acumulador para almacenamiento, que dependerá del fluido para almacenar y de la capacidad en litros:

$$C_{almacenamiento}\left[\frac{[€]}{L}\right] = almVolumen[L] \cdot A \cdot \left(\frac{almVolumen}{1.000}\right)^{-0.5}$$

Agua	$\boldsymbol{A = 2,1}$
Vapor	$A = 2{,}1$
Aceite térmico	$A = 6{,}4$

Tabla 2: constante A según el tipo de fluido.

El coste total de inversión será:

$$Coste_{planta}[€] = C_{módulo} \cdot N^{\underline{o}}{}_{colectores} + C_{bloque\ de\ potencia} \cdot N^{\underline{o}}{}_{colectores} + C_{almacenamiento}$$

3.2.3.3 Costes de mantenimiento.

Al igual que los costes de inversión, los costes de mantenimiento anuales están definidos por experimentación de la empresa SOLATOM. Se han establecido como:

$$O\&M\ [€] = 70 \cdot N^{\underline{o}}{}_{colectores}$$

3.2.3.4 Modelos de negocio.

Los modelos de negocio implementados en SHIPCal son modelo “llave en mano” y “ESCO”:

3.2.3.4.1 Modelo “llave en mano”.

El diseño y ejecución del proyecto se efectúan bajo responsabilidad únicamente del contratista del proyecto. Determinados aspectos pueden estar fijados por el contratante, pero el desarrollo detallado del proyecto, incluyendo la adquisición de maquinaria y la consiguiente instalación están bajo cargo y supervisión del contratista.

Antes de explicar cómo está implementado en SHIPCal este modelo, se explica brevemente el término LCOE.

- El LCOE (Levelized Cost Of Energy): es el coste unitario actual de la energía producida [$\frac{€}{kWh}$]. Se calcula con la siguiente expresión:

$$LCOE\ \left[\frac{€}{kWh}\right] = \frac{TLCC}{Energía_{anual} \cdot \sum_{i=0}^{años-1} \frac{1}{(1+r)^{años}}}$$

Donde $Energía_{anual}$ es la energía producida cada año y r es la tasa de descuento. $TLCC$ (Total life-cicle cost) se refiere al valor actual de los costes totales durante todo el ciclo de vida, que con el término $\frac{1}{\sum_{i=0}^{años-1}\frac{1}{(1+r)^{años}}}$ se anualiza para tener un valor normalizado de los costes cada año. Estos costes normalizados se dividen entre la energía promedio producida en un año , $Energía_{anual}$, y se obtiene el coste en [$\frac{€}{kWh}$]. En el caso de las simulaciones de SHIPCal solo se consideran como costes la inversión inicial (año 0) y los costes de mantenimiento (año 1 hasta el último año). Así, el primer año (año 0) el valor de la inversión ya será el valor actualizado. El resto de los años, se debe actualizar los costes de mantenimiento al año actual (año 0):

$$O\&M_{0,i}[€] = O\&M \cdot (\frac{1}{(1+r)^{i}})$$

$O\&M_{0,i}$ será el valor actual en el año 0 del coste de mantenimiento del año i. Por tanto $TLCC$ se puede definir como:

$$TLCC\ [€] = Inversión_{inicial} + \sum_{i=0}^{años-1} \frac{O\&M_i}{(1+r)^{años}}.$$

Tras explicar el LCOE, se explica el modelo de llave en mano en SHIPCal. El análisis de costes y financiero se hará de la siguiente manera: se recorre un bucle desde 0 hasta $Nº_{años} - 1$. El primer año (año 0) el coste del combustible no ha sufrido incremento aún y el único flujo de caja es el coste de inversión inicial, que será negativo. El coste para el $TLCC$ de este año será la inversión y la producción solar será nula:

$$numerador_{LCOE,año\ 0} = inversión_{inicial}$$

$$denominador_{LCOE,año\ 0} = 0$$

El resto de los años:

- Energía producida: se considera igual todos los años y su valor es la suma de toda la energía producida durante todas las iteraciones de un año completo (8.760, 52.560 o 35.040), ya que se recuerda que el estudio financiero se realiza solo para simulaciones en las que las variables de inicio y final de simulación representan 1 año completo de duración. Este valor se obtiene en el "bloque 2.2: integración anual".

$$Energía_{producida,\ año\ i}[kWh] = Energía_{anual}$$

- Ahorro bruto: se considera como el coste que supondría el uso de combustible en la caldera para obtener la energía producida mediante recurso solar y el ahorro conseguido al evitar tasas por toneladas de CO_2 producido.

$$Ahorro_{año\ i}[€] = Energía_{anual} \cdot \frac{Coste_{fuel} \cdot (1 + \Delta coste)^{i-1}}{Eficiencia_{caldera}} + Ahorro_{CO_2}$$

  El término $(1 + \Delta coste)^{i-1}$ calcula el valor futuro, en un año i, del coste actual (año 0) del combustible, ya que este coste tiene un incremento anual promedio ($\Delta coste$). El ahorro de CO_2 no sufre incremento y tampoco se considera inflación.

- Numerador y denominador del LCOE para año i: como se ha explicado, para cada año se guarda el numerador y denominador del LCOE para luego hacer el sumatorio de las componentes del vector con todos los años.

$$numerador_{LCOE,año\ i} = O\&M \cdot \frac{1}{(1+r)^i}$$

$$denominador_{LCOE,año\ i} = Energía_{anual} \cdot \frac{1}{(1+r)^i}$$

- Ahorro neto: la diferencia entre el ahorro bruto y los costes anuales de mantenimiento ($O\&M$).

- Flujo de caja (año i):

$$FCF[i][€] = ahorro_{neto}[i]$$

- Flujo de caja acumulado: flujos de caja de los años anteriores sumado al ahorro neto del año i:

$$Acum_{FCF}[€] = Acum_{FCF} \cdot [i-1] + ahorro_{neto,i}$$

Al terminar el bucle, se calcula el LCOE, haciendo el sumatorio de los vectores de numerador y del denominador. El sumatorio del numerador será el $TLCC$.

$$LCOE\ [€] = \frac{\sum Numerador_{LCOE}}{\sum Denominador_{LCOE}}$$

Se calculará la TIR (Tasa Interna de Retorno) con el comando de Python *np.irr* y el vector de flujos de caja*:*

$$TIR = np.irr(FCF)$$

Por último, se calcula el número de años para amortizar la inversión, sumando el número de años que tiene flujos de caja acumulados ($Acum_{FCF}$) negativos, ya que inicialmente los flujos de caja son negativos hasta que el ahorro neto acumulado compensa la inversión:

$$FCF_{año\ 0}[€] = -inversión$$

$$Inversión\ amortizada \Leftrightarrow Acum_{FCF} \cdot [i-1] + ahorro_{neto,i} > 0$$

3.2.3.4.2 Modelo "ESCO".

Muchas empresas e industrias no tienen la capacidad o prioridad de destinar fondos a innovaciones energéticas con un tiempo de retorno de inversión medio. En el modelo ESCO (Energy Services Company), una empresa contratista que habitualmente suele ser una empresa especializada en servicios energéticos hace el estudio de viabilidad del proyecto y realiza la inversión para llevarlo a cabo para una empresa contratante. La empresa contratante no realiza el desembolso de inversión y paga a la empresa ESCO mediante beneficios o ahorro producido por la innovación energética que se ha instalado.

Al igual que en el modelo de "llave en mano", en SHIPCal se realiza un bucle para crear vectores con tantas componentes como años de vida útil tenga la inversión. El primer año (año 0), la producción es nula y no ha incremento en el precio de combustible. Sin embargo, dado que la empresa contratante no ha realizado el desembolso de la inversión, el flujo de caja de este año es 0. En el resto de los años del bucle (año 1 hasta $Nº_{años} - 1$) se calculan las siguientes variables:

- Energía producida: al igual que en el modelo "llave en mano", se considera igual todos los años y su valor es la suma de toda la energía producida durante todas las iteraciones de un año completo (8760, 52560 o 35040), ya que se recuerda que el estudio financiero se realiza solo para simulaciones de 1 año completo. Este valor se obtiene en el "bloque 2.2: integración anual".

$$Energía_{producida,\ año\ i}[kWh] = Energía_{anual}$$

- Beneficio de la ESCO: como es la empresa ESCO la que realiza la inversión y ejecución del proyecto, su beneficio vendrá definido por un porcentaje del ahorro producido por la planta solar a la empresa contratante:

$$Beneficio_{ESCO,año\ i}[€] = \left(Energía_{anual} \cdot \frac{Coste_{fuel} \cdot (1 + \Delta coste)^{i-1}}{Eficiencia_{caldera}}\right) \cdot \%reducción$$

- Ahorro bruto: se considera como el coste que supondría el uso de combustible en la caldera para obtener la energía producida mediante recurso solar y el ahorro conseguido al evitar tasas por toneladas de CO_2 producido. Se le aplica la reducción debida al beneficio que va destinado a la ESCO:

$$Ahorro_{\text{año } i}[€] = (1 - \%reducción) \cdot (Energía_{anual} \cdot \frac{Coste_{fuel} \cdot (1 + \Delta coste)^{i-1}}{Eficiencia_{caldera}}) + Ahorro_{CO_2}$$

- Ahorro neto:

$$Ahorro_{neto}[€] = Ahorro_{bruto} - Beneficio_{ESCO} - O\&M$$

- Flujo de caja (año i):

$$FCF[i][€] = ahorro_{neto}[i]$$

- Flujo de caja acumulado: flujos de caja de los años anteriores sumado al ahorro neto del año i:

$$Acum_{FCF}[€] = Acum_{FCF} \cdot [i-1] + ahorro_{neto,i}$$

Tras finalizar el bucle, se calculará la TIR y los años para rentabilizar la inversión al igual que en el modelo "llave en mano". Dado que la empresa contratante no realiza desembolso inicial, el periodo de retorno se consigue el primer año:

$$TIR = np.irr(FCF)$$

$$FCF_{\text{año } 0}[€] = 0$$

$$Inversión\ amortizada \Leftrightarrow Acum_{FCF} \cdot [i-1] + ahorro_{neto,i} > 0 \rightarrow Se\ amortiza\ en\ el\ año\ 1$$

3.2.3.5 Modificaciones en el bloque 3.

Únicamente es necesario modificar la condición de entrada a este bloque añadiendo la condición de que la variable *steps_sim* (número de pasos que tiene la simulación) sea 52.560 (diezminutal) o 35.040 (quinceminutal):

if finance_study==1 and steps_sim==8759 or steps_sim==52560 or steps_sim==35040:

3.2.4 Bloque 4: generación de gráficos.

Este bloque recibe los vectores de resultados elaborados en el bloque 2 y el bloque 3 y genera los gráficos correspondientes según se elijan con la variable *plots.* Hay gráficos que solo tiene sentido examinar si se hacen simulaciones anuales, como puede ser un gráfico con la producción de cada mes o una curva de cash-flows.

Otros gráficos solo pueden estudiarse con simulaciones de menos pasos pues no se puede apreciar con claridad el comportamiento de la simulación al haber tal cantidad de pasos de simulación en un año.

3.2.4.1 Modificaciones en el bloque 4.

Al igual que en el bloque 3, se debe establecer el condicionante que incluya la opción de paso diezminutal o quinceminutal para simulaciones de un año completo:

if steps_sim==8759 or steps_sim==52560 or steps_sim==35040:

Para simulaciones de duraciones menores que 1 año, se establece que el número de iteraciones sea distinto del número anual de iteraciones:

if steps_sim!=8759 and steps_sim!=52560 and steps_sim!=35040:

Dentro de las funciones generadoras de gráficos, es necesario implementar el cambio de leyenda del eje de abscisas que marca los pasos de simulación, en función de la variable *itercontrol*. En todas las funciones se escribirá el código siguiente:

```
if itercontrol=='paso_10min':
    ax2.set_xlabel('simulación: pasos diezminutales')
elif itercontrol=='paso_15min':
    ax2.set_xlabel('simulación: pasos quinceminutales')
```

Las funciones *prodSummerPlot*, *prodWinterPlot*, *storageWinter* y *storageSummer* solo muestran datos de la primera semana de enero y de junio por lo que es necesario cambiar la longitud de los vectores que se leerán y representarán: dado que antes la primera semana de enero eran las 168 primeras iteraciones (horas) y la primera semana de junio eran las iteraciones que iban desde 3.624 hasta 3.791 en los vectores a leer, ahora se incluye un factor que valdrá 4 si es quinceminutal o 6 si es diezminutal. También es necesario realizar modificaciones de las funciones auxiliares *arraysMonth* y *arrays_Savings_Month* que se detallan en el apartado 3.3. Estas dos últimas funciones sirven para representar los gráficos con la producción y el ahorro de cada mes durante el año completo.

3.2.5 Bloque 5: Generación de informes.

Únicamente se generan informes para simulaciones anuales completas. La única modificación es implementar la condición de que el número de iteraciones anuales pueda ser 52.560 o 35.040:

if steps_sim==8759 or steps_sim==52560 or steps_sim==35040:

3.3 Funciones auxiliares a SHIPCal.

Las funciones auxiliares son aquellas externas a la función SHIPCal que son necesarias para calcular y obtener los distintos vectores y parámetros de la simulación. Algunas de ellas son auxiliares entre sí. En este apartado se detallan las funciones que es necesario modificar explicando su funcionamiento para entender los cambios que se introducen. También se explicarán las funciones que se usan para calcular los resultados de la simulación en el bloque 2, aunque solo sufren pequeños cambios. Los códigos de Python se encuentran en el anexo.

3.3.1 Función demandCreator2.

La función *demandCreator2* se crea para elaborar un vector cuyas componentes tienen el valor de la demanda de energía en cada paso del año completo. A continuación, se explica en que consiste la función.

La función recibe como argumentos de entrada la demanda de kWh en todo el año en la variable *totalConsumption* y los vectores ponderados de demanda que son: *dayArray*, *weekArray*, *monthArray* y *step_minArray*. El vector *step_minArray* será *ten_min_array* o *fifteen_minArray* en función del modo de simulación que se esté usando. Por ello, recibe la variable *itercontrol* para controlar qué modo de paso se va a usar.

Esta función recorre tres bucles anidados para crear un único vector con tantas componentes como iteraciones anuales haya según el tipo de paso de simulación. Así, para el modo de paso diezminutal y quinceminutal el vector resultante que devolverá tras finalizar los tres bucles debe contener 52.560 y 35.040 respectivamente.

El primer bucle recorre los doce meses del año, siendo cada mes una iteración del bucle. En el segundo bucle se recorren los días de cada mes. Para un día determinado, se accede al tercer bucle que recorrerá las 24 horas del día y multiplicará cada hora por el vector *step_minArray.* Se creará pues, una lista de 144 componentes (diezminutal) o 96 (quinceminutal) cuyos valores son el producto del peso de cada porción diezminutal o quinceminutal respecto a una hora y el peso de una hora respecto a un día, establecidos por los vectores ponderados de demanda. Esta lista se añade en el segundo bucle (que recorre todos los días del mes) obteniendo un vector, cuyo número de componentes será 144*Nº días del mes (diezminutal) o 96*Nº de días del mes (quinceminutal).

Al terminar el mes, en el primer bucle se añade lo calculado hasta ahora para recoger todos los vectores mensuales y los agrupa en un vector de tantas componentes como pasos tenga el año (52.560 o 35.040).

Para poder multiplicar la energía total del año (*totalConsumption*) con el vector resultante de los bucles, es necesario dividir dicho valor de energía por un factor de conversión para pasar de semanas a mes.

$$1\ año = 8.760\ horas \cdot \frac{1\ día}{24\ horas} \cdot \frac{1\ semana}{7\ días} \cdot \frac{1\ mes}{x\ semanas} \cdot \frac{1\ año}{12\ meses} \rightarrow x = 4{,}3496$$

$$annual\ \left[\frac{kWh}{hora}\right] = totalConsumption\left[\frac{kWh}{año}\right] \cdot \frac{año}{mes} \cdot \frac{mes}{semanas} \cdot \frac{semana}{días} \cdot \frac{hora}{día} \rightarrow$$

$$annual\ \left[\frac{kWh}{hora}\right] = totalConsumption\left[\frac{kWh}{año}\right] \cdot \frac{1}{x} \cdot\ vector\ resultante\ de\ bucles$$

El vector *annual* que devuelve la función resulta de multiplicar el vector que devuelven los bucles por esa energía normalizada.

- **Ejemplo.**

Datos de entrada a la función:

1. Se supone que hay demanda de energía todos los días de la semana, por tanto, el vector temporal semanal *weekArray* será un vector cuyas componentes sean 1/7:

$$weekArray = [0{,}143;\ 0{,}143;\ 0{,}143;\ 0{,}143;\ 0{,}143;\ 0{,}143;\ 0{,}143]$$

2. Se supone que no hay demanda en febrero y en marzo se requiere el doble. El vector temporal *monthArray* es un vector cuyas componentes tienen un valor de 1/11, pues el número de meses con demanda es 11, la segunda componente (febrero) tendrá valor 0 y la tercera (marzo) tendrá valor 2/11:

$$monthArray = [\frac{1}{11}, 0, \frac{2}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}]$$

3. Se requiere demanda de energía las 24 horas del día e igual en todas las horas, por tanto:

$$dayArray = [\frac{1}{24}, \frac{1}{24}]$$

4. Se quiere hacer simulación quinceminutal, por tanto, *itercontrol* valdrá *'paso_15min'*.

5. El segundo cuarto de hora de cada hora no hay demanda de energía. El resto de los cuartos de hora requieren la misma demanda. Este vector entrará en la función como *step_minArray.*

   Cabe recordar que esto sólo sucede en este ejemplo, pues para el resto de las simulaciones, se ha establecido que la demanda sea igual en cada parte de una hora, ya sea porción diezminutal o cuarto de hora.

$$fifteen\_minArray = [\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}]$$

6. La energía anual demandada *totalConsumption* es 6.000 kWh por cada hora, multiplicada por 8.832 horas nominales de demanda resultando 5.299.200 kWh al año.

   Las horas nominales de demanda resultan de restar a 8.760 horas las del mes de febrero (no hay demanda) y sumar las de marzo (es el doble).

El vector *annual* que devolverá la función será un vector de 35.040 componentes quinceminutales que hay en un año. Todos los meses excepto febrero y marzo tendrán una estructura como la siguiente:

$$[2.000, 0, 2.000, 2.000, 2.000, 0, 2.000, 2.000 \ldots]$$

Cada hora son 4 componentes, donde el segundo cuarto es 0 y suma 6.000 $\frac{kWh}{hor}$. Febrero tendrá componentes con valor 0 y marzo de valor 4.000 menos el segundo cuarto de hora que será 0.

3.3.2 Función calc_min_year.

La función *calc_min_year* es una versión modificada de *calc_hour_year*. Ambas funciones tienen como tarea transformar las variables de comienzo y final de simulación en pasos de simulación. Así, cuando recibe las variables (*month_ini_sim, hour_ini_sim,* etc.), devuelve el número de pasos respecto al año que esas variables representan.

La función recibe *mes*, *día*, *hora*, porción minutal (*minute*) y el control de simulación *itercontrol.* Se empieza inicializando las siguientes variables:

- *Mes_days*: vector de 12 componentes con el valor del número de días de cada mes.
- *Num_days*: variable que indica la cantidad de días transcurridos en el año hasta llegar al día que se introduce en la función. Se inicializa como 0.
- *Cont_mes*: contador de mes que decrece con cada paso del bucle desde el mes introducido en la función hasta llegar al mes 0 (enero). Se inicializa como el mes introducido menos 1.

Tras la definición de estas variables se entra en un bucle *while* en el que la condición impuesta para entrar es que el contador de mes sea mayor que 0. Una vez dentro, en cada iteración se resta 1 al contador y se actualiza la variable *num_days* sumando los días de *mes_days* del mes que marque el contador *cont_mes*. Tras finalizar en este bucle, se suman los días del mes introducido en *mes* a la variable *num_days*.

Seguidamente, es necesario añadir el número de horas del día. Para ello, se resta 1 al número de días, se multiplica por 24 para pasar a horas y se añade el número de horas introducida en la variable *hora*, almacenándose dicho valor en *hour_year*.

Para tener el número de pasos diezminutales o quinceminutales se debe multiplicar el número de horas por 4 (quinceminutal) o 6 (diezminutal) y sumar la variable *minute.* Este valor se almacena en *min_year*.

Dado que los en los ficheros de datos, los pasos minutales son 0, 10, 20, 30, 40 y 50 minutos para diezminutal y 0, 15, 30 y 45 para quinceminutal, los números que se pueden introducir en *minute* será: 0, 1, 2, 3, 4, 5 ó 0, 1, 2 y 3 según qué modo de paso se escoja. Por este motivo, es necesario sumar 1 a *min_year* y poder obtener el contador real de pasos.

- **Ejemplo.**

Se hará el ejemplo con las variables de inicio de simulación. Se supondrá que la simulación comienza en febrero, el día 5 a las 0.50 H. Dado que se utilizará paso diezminutal, la variable *itercontrol* valdrá *'paso_10min'*.

$\boldsymbol{Month\_ini\_sim = 2}$
$\boldsymbol{Day\_ini\_sim = 5}$
$\boldsymbol{Hour\_ini\_sim = 0}$
$\boldsymbol{Ten\_min\_ini\_sim = 5}$
$\boldsymbol{Itercontrol = 'paso\_10min'}$

Tabla 3: valores de las variables que recibe la función.

Al ser el mes 2, se le resta 1 para que se corresponda con la componente del vector en Python y se entra en el bucle. Al entrar se vuelve a restar 1 al contador, resultando este 0 (enero en el vector). Se actualiza pues el número de días, que actualmente tiene valor 0, resultando así 31 días de enero porque se lee el valor del contador 0 en *mes_days.* Cuando se va a volver a entrar en el bucle el contador vale 0, por tanto, no entra en él. A continuación, se le suma el día introducido y se resta 1. El total de días se multiplica por 24 y se le suman las horas introducidas en la función, que en este caso es 0. *Hour_year* será pues 840 horas que van desde el 1 de enero a las 00.00 horas hasta el 5 de febrero a las 00.00 horas.

La variable *itercontrol* obliga a paso diezminutal y hará que el número de horas se multiplique por 6 y se le añadan las porciones diezminutales introducidas en *minute.* En este supuesto, *minute* tiene el valor de 5 y de 0 a 5 la suma es 6 pasos. *Min_year* resultará 5046 pasos diezminutales desde las 00.00 horas del 1 de enero hasta el 5 de febrero a las 00.50 horas.

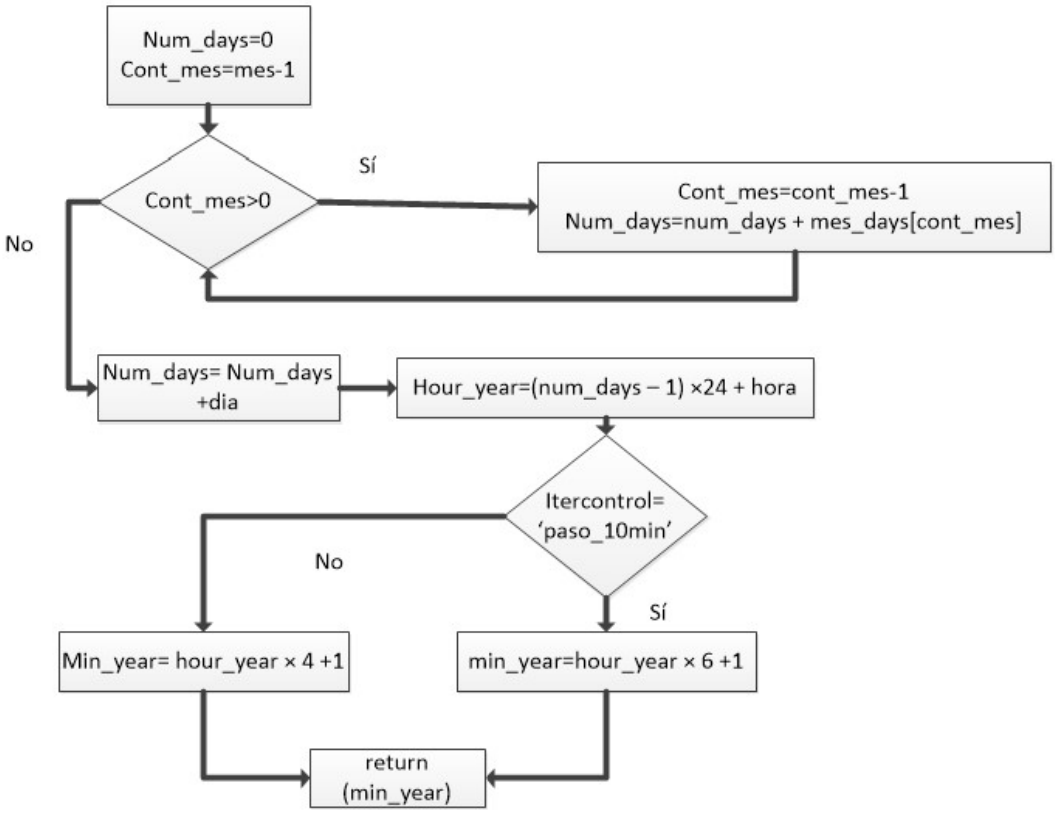

Figura 15: diagrama de flujo de *calc_min_year*.

3.3.3 Calc_day_year.

La función *calc_day_year* se crea para poder calcular el número total de días desde el inicio del año y poder calcular la hora solar en *SolarEQ_simple2*.

Recibe un mes y un día funcionando de manera similar a *calc_min_year* con dos bucles que actualizan una variable según el valor de un contador y un vector con los valores de los días que contiene cada mes. Se declara una variable vectorial (*mes_days*) con 12 componentes y el número de días del mes correspondiente. Seguidamente se declara una variable *num_days* con valor inicial 0 y que será la que devuelva la función. Por último, se declara *cont_mes* que resulta de restar 1 al mes que recibe la función. Mientras *cont_mes* sea mayor que 0, entrará en un bucle *while* y restará nuevamente 1 a *cont_mes*. Seguidamente *num_days* se actualizará con el valor correspondiente a *cont_mes* en *mes_days*. Cuando se sale del bucle, se añaden el número de días de la variable *dia*.

- **Ejemplo.**

Se establece que la variable *mes* vale 3 (marzo) y la variable *día* vale 5. Inicialmente *num_days* vale 0 y *cont_mes* es igual a mes menos 1, resultando 2. Como 2 es mayor que 0, se entra en el bucle y *cont_mes* vuelve a decrecer en 1. Se coge la segunda componente (1 en Python) de *mes_days* y se actualiza *num_days* que ahora valdrá 28. Como *cont_mes* ahora vale 1, sigue siendo mayor que 0 y vuelve a entrar en el bucle y restándose nuevamente 1 y tomando el valor 0. Se accede pues a la primera componente (0 en Python) de *mes_days* que vale 31 y se le suma a *num_days* que valía 28. Cuando se sale del bucle se suman los días introducidos en la función que son 5. La función devolverá *num_days* con el valor de 64 días.

3.3.4 Función DemandData2.

Esta función se crea como una versión modificada de *DemandData* para habilitar a SHIPCal a utilizar un vector de demanda de energía con únicamente el número de iteraciones diezminutales o quinceminutales que tenga la simulación que se esté llevando a cabo. La función recibe el vector *file_demand* de demanda anual que elabora la función *demandCreator2,* las variables de inicio y finalización de simulación introducidas en la terminal de introducción de variables y la variable de control *itercontrol*. Esta función requiere de la función auxiliar *calc_min_year.*

La función comienza pasando la lista *file_demand* a formato tipo *array.* Seguidamente se calcula el número de pasos acumulado durante el año hasta el instante de comienzo de simulación mediante la función *calc_min_year* y se almacena en *min_year_ini*. Se hace el mismo proceso para calcular el número de pasos hasta el final de simulación y se almacena en *min_year_fin*. Se hace la diferencia de ambos para obtener el número de pasos que tendrá la simulación.

Para crear el vector de demanda, se recorre este con el número de pasos como contador. Para ello se utiliza la función de Python *range* que da un rango entre dos valores, el último sin incluir. Por tanto, el bucle se recorrerá desde 0 hasta el número de pasos menos 1. La componente para leer de *file_demand* será la suma de *min_year_ini* (inicio de simulación) más el paso de esa iteración y restando 1 para acceder a la fila correcta del vector (ya que empieza en 0 en Python).

- **Ejemplo.**

Se plantea el caso de paso quinceminutal en el que *file_demand* tendrá 35.040 componentes, desde 0 a 35.039 y el valor de dichas componentes será 1.500 kWh. Presentará la siguiente forma:

$$file\_demand = [1.500, 1.500, 1.500, 1.500, 1.500\ldots]$$

Variables de inicio	Variables de fin
$\boldsymbol{month\_ini\_sim = 2}$	$month\_fin\_sim = 2$
$\boldsymbol{day\_ini\_sim = 5}$	$day\_fin\_sim = 5$
$\boldsymbol{hour\_ini\_sim = 0}$	$hour\_fin\_sim = 20$
$\boldsymbol{fifteen\_min\_ini\_sim = 1}$	$fifteen\_min\_ini\_sim = 3$

Tabla 4: variables que recibe la función.

La función auxiliar *calc_min_year* devolverá 3.362 para *min_year_ini* y devolverá 3.444 pasos para *min_year_fin*. La diferencia resulta 82 pasos, que serán los pasos de simulación. Se crea un vector con 82 componentes y en cada paso por el bucle con *range* de 0 a 82 sin entrar en el 82 se rellena la componente correspondiente de *file_demand*. Devolverá pues un vector de 82 componentes de valor 1500 kWh.

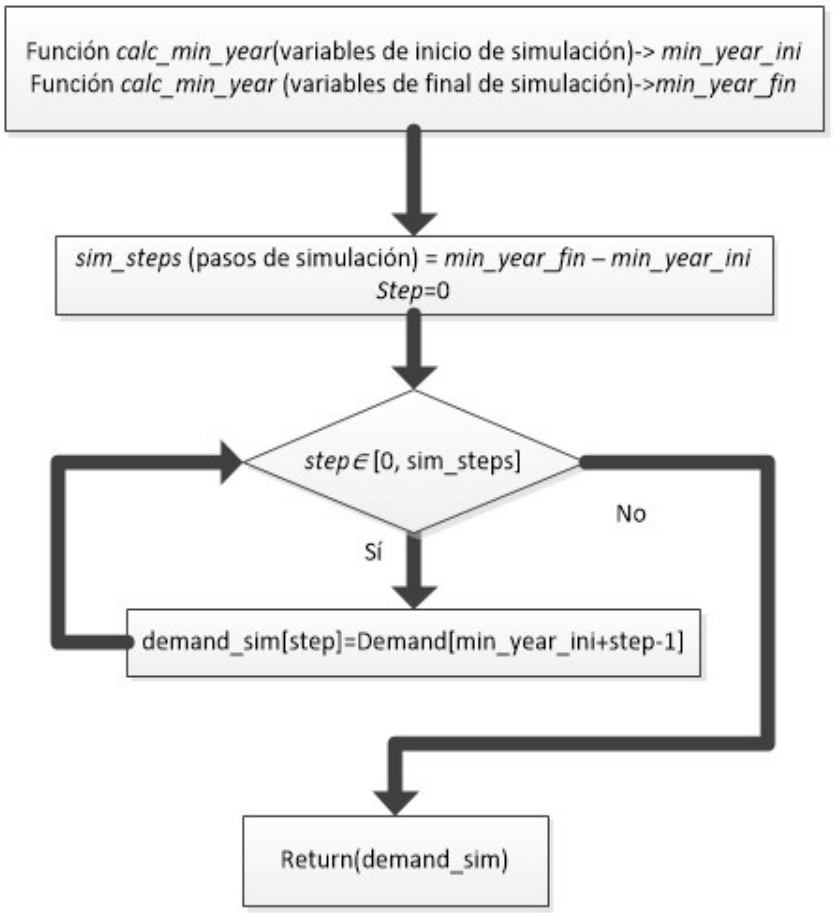

Figura 16: diagrama de flujo de función *DemanData2*.

3.3.5 SolarData2.

Su función es obtener mediante *calc_min_year*, *Meteo_data2* y *SolarEQ_simple2* los datos solares y meteorológicos para la simulación en paso diezminutal o quinceminutal.

La función recibe el fichero de datos solares importados de todo el año (*file_loc)* que se leerá para sacar la irradiancia y temperatura. Además, recibe las variables de inicio y final de simulación, la variable *itercontrol*, identidad del usuario (*sender*), la latitud, longitud y el huso horario.

El primer paso dentro de la función es llamar a la función *calc_min_year* para calcular el número de pasos totales desde el inicio de año hasta el comienzo y el final de la simulación, almacenándose dichos valores en *min_year_ini* y *min_year_fin.* Después se calcula la diferencia de ambos para calcular el número de pasos que tendrá la simulación y se almacena en *sim_steps*. Tras estos pasos, se procesará el fichero de datos solares con la función *Meteo_Data2* según la identidad de *sender* y se extrae una matriz con todas las columnas del fichero que se denominará *data* y dos vectores: la irradiancia *DNI* y la temperatura *temp.* Estas tres variables siguen conteniendo los datos de todo el año, por tanto, tendrán 8760 filas (horario), 52560 (diezminutal) o 35040 (quinceminutal).

Antes de entrar al bucle, se definen vectores con tantas componentes como pasos tenga la simulación. Así, con el número de pasos de la simulación *sim_steps* se crean 5 vectores nuevos de dimensión *sim_steps* que tienen componentes de valor 0 y corresponden a parámetros solares: hora angular (*W_sim*), elevación solar (*SUN_ELV_sim*), acimut solar (*SUN_AZ_sim*), declinación solar (*DECL_sim*) y el zenit (*SUN_ZEN_sim*).

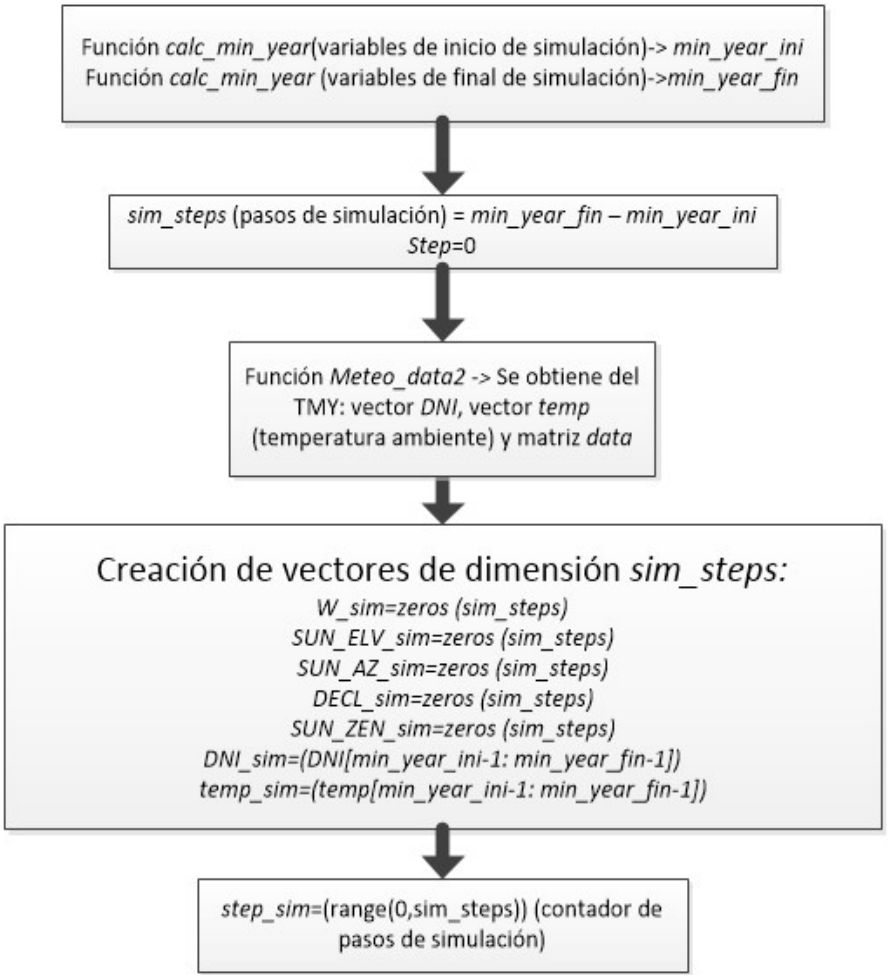

Figura 17: diagrama de flujo de *SolarData2* (parte 1).

Se crean dos vectores más leyendo *DNI* y *temp* desde *min_year_ini*-1 hasta *min_year_fin*-1 formándose *DNI_sim* y *temp_sim* para tener vectores con únicamente tantos datos como pasos haya en la simulación. Se hace lo mismo y se leen las 5 primeras columnas de la matriz *data* y se almacenan los datos de simulación en 5 vectores que recogen los meses (*month_sim*), días (*day_sim*), horas (*hour_sim*), minutos (*min_sim*) y contador de pasos de simulación dentro del año (*min_year_sim*) yendo desde *min_year_ini*-1 hasta *min_year_fin*-1.

Por último, se accede a un bucle con tantas iteraciones como pasos tenga la simulación (*sim_steps*) y en cada iteración se accede a la función *SolarEQ_simple2* para calcular los parámetros solares y almacenarlos en los vectores creados de hora angular, elevación solar, etc.

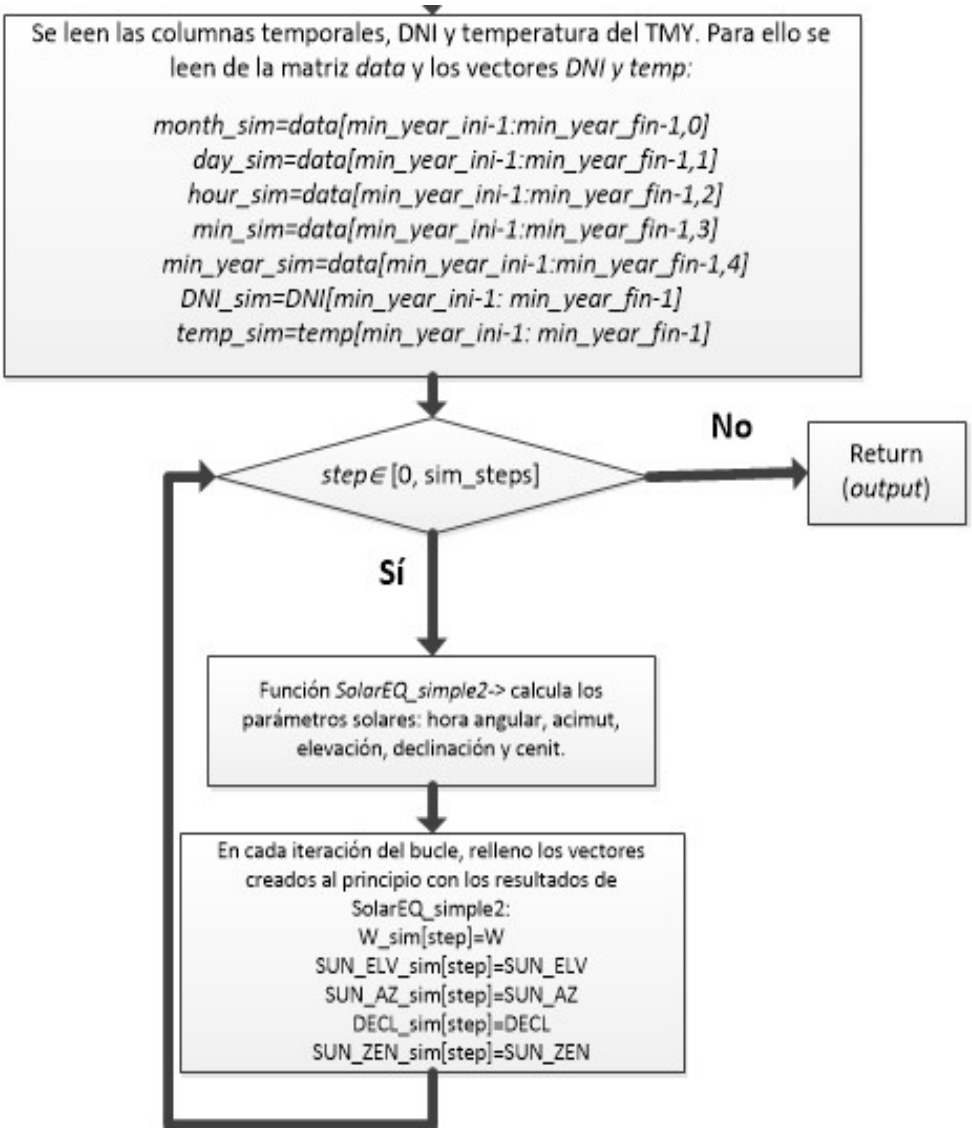

Figura 18: diagrama de flujo de *SolarData2* (parte 2).

Todos los vectores creados se unificarán y almacenarán en una variable matricial llamada *output:*

- Columna 0: meses de simulación (*month_sim*).
- Columna 1: días de simulación (*day_sim*).
- Columna 2: horas diarias de simulación (*hour_sim*).

- Columna 3: minutos de simulación (*min_sim*).
- Columna 4: contador de pasos respecto al año.
- Columna 5: hora angular.
- Columna 6: elevación solar.
- Columna 7: acimut solar.
- Columna 8: declinación solar.
- Columna 9: cenit solar.
- Columna 10: irradiancia.
- Columna 11: temperatura.
- Columna 12: contador de pasos de la simulación (*step_sim*).

3.3.6 Meteo_Data2.

La función *Meteo_data2* recibe el TMY. La única diferencia entre *Meteo_data* y *Meteo_data2* es que el TMY que recibe *Meteo_data2* tiene una columna más con los pasos minutales, por tanto, *DNI* deja de ser la columna 8 y ahora es la 9, por lo que debe cambiarse en el código. Lo mismo ocurre con la temperatura pues pasa de ser la columna 9 a la 10:

$$DNI = data[:,9]$$

$$temp = data[:,10]$$

3.3.7 SolarEQ_simple2.

Esta función recibe el mes, día, hora, paso minutal, huso, *to_solartime*, longitud (*long*) de la ubicación donde se simula y la latitud. Sirve para calcular la elevación, acimut y cenit solar.

3.3.7.1 Coordenadas solares.

Para el cálculo de los modificadores del ángulo de incidencia (IAM), es necesario saber las coordenadas que definen la posición del Sol (acimut y elevación) y el cenit de la ubicación. Para su obtención, es necesario saber la declinación y la hora angular. Las ecuaciones para calcularlos se encuentran en la función auxiliar *SolarEQ_simple* (modo horario) y *SolarEQ_simple2* (modo diezminutal o quinceminutal).

El ángulo horario es el arco de ecuador contado desde la intersección del ecuador con el meridiano local o del observador hasta el circulo horario del astro (en este caso, el Sol) recorrido en sentido horario. Para medir el desplazamiento angular del Sol en la cúpula celeste, se toma como origen el mediodía solar. Por ello, es necesario disponer de la hora solar para calcular el ángulo horario. Cada hora equivale a 15º:

$$W = (hora - 12) \cdot 15 \text{ [9]}$$

La declinación solar es el ángulo que forma la recta que une la Tierra con el Sol y el plano del ecuador terrestre. Su valor es variable a lo largo del año, valiendo 23.5º cuando se inicia el verano o el invierno (máximos) y tiene valor nulo en los equinoccios.

Así pues, una vez se ha calculado el ángulo horario y la declinación, se calculan las coordenadas solares. Dichas coordenadas describen el desplazamiento de la Tierra respecto al Sol puede mediante el trazado del movimiento del Sol en la cúpula celeste:

- Acimut: ángulo medido sobre el horizonte del observador, formado entre la dirección Norte y la proyección de la dirección observador-Sol. Se representa en grados respecto al Norte, recorriéndose en el sentido de las agujas del reloj.

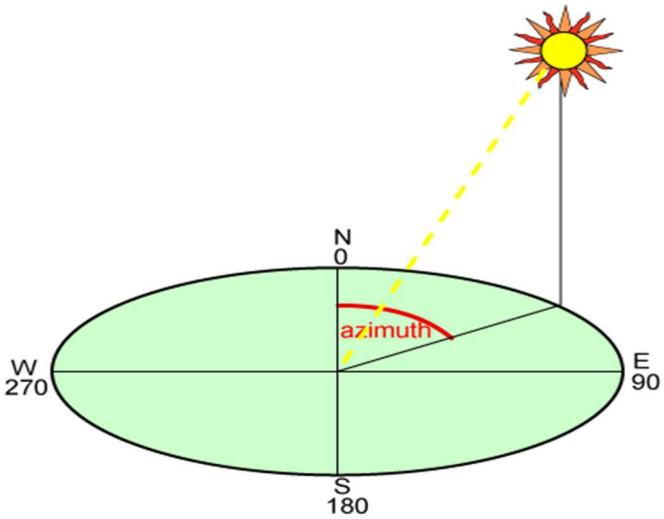

Figura 19: acimut solar. [10]

- Elevación: ángulo medido desde el horizonte del observador con la dirección observador-Sol.

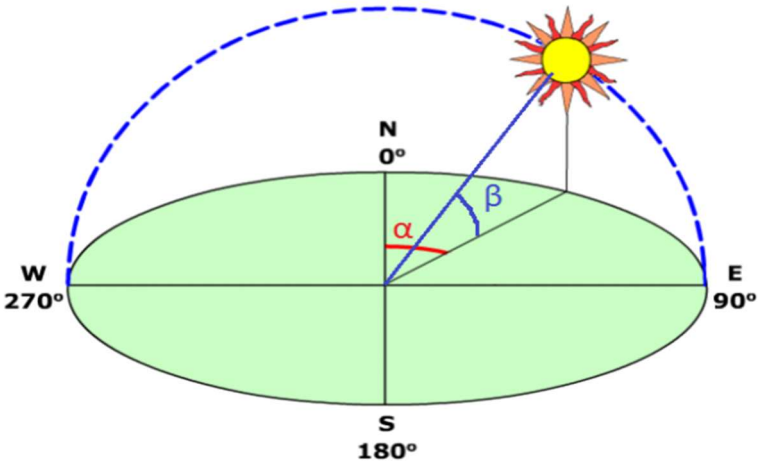

Figura 20: β representa la elevación y α el acimut.[11]

- Ángulo cenital: es el ángulo complementario de la elevación solar.

3.3.7.2 Hora solar verdadera.

La hora angular depende de la hora solar verdadera, que es distinta de la hora local y debe calcularse a partir de ésta última. Existe la posibilidad de que el TMY que se proporciona a SHIPCal ya tenga realizada esta conversión, pero deben implementarse las ecuaciones para pasar a hora solar verdadera en caso de que no se haya convertido previamente.

La hora solar verdadera es un formato horario en el cual se establece que las 00.00 H se dan cuando el Sol está en el punto más alto en su recorrido por el cielo, es decir, cuando se encuentra sobre el meridiano local de la ubicación geográfica en cuestión. Para calcular la hora solar, es necesario conocer en qué meridiano se encuentra la ubicación y la diferencia en horas entre dicho meridiano y el meridiano de Tiempo Universal Coordinado (UTC en inglés).

En SHIPCal, el mediodía se ha establecido como 12.00 H y la diferencia horaria es la que se ha definido con la variable *huso.* También se ha definido la variable *to_solartime* sirve para pedir a SHIPCal que calcule hora solar y tendrá que valer '*on'*.

En las simulaciones de ejemplo que se han hecho en este trabajo, el TMY no está en formato de hora solar verdadera, por lo que hay que calcularlo *(to_solartime* valdrá 'on'). En este caso, el TMY está en formato UTC por lo que la diferencia horaria resulta 0 (*huso* igual a 0).

3.3.7.3 Estructura de la función.

Al comenzar la función se calcula un conversor de radianes a grados, se lee el día del año de un fichero en el repositorio de SHIPCal en función del día y el mes. El día del año también puede obtenerse con la función *calc_day_year*. A continuación, se calcula el ángulo diario (*DJ*) a partir del día del año:

$$DJ = \frac{2 \cdot \pi}{365} \cdot (día\_del\_año - 1) \text{ [12]}$$

Con el ángulo diario, se calcula la declinación mediante la fórmula de Spencer (1971):

$$Declinación = 0{,}006918 - 0{,}399912 \cdot cos(DJ) + 0{,}070257 \cdot sen(DJ) - 0{,}006758 \cdot cos(2 \cdot DJ) + 0{,}000907 \cdot sen(2 \cdot DJ) - 0{,}002697 \cdot cos(3 \cdot DJ) + 0{,}00148 \cdot sen(3 \cdot DJ) \text{ [12]}$$

Antes de continuar, se encuentra la opción de calcular la hora solar en caso de ser necesario. Para ello, *to_solartime* debe tener como valor la cadena de caracteres '*on'*. El primer paso es llamar a la función *calc_day_year* para calcular el número de días desde el inicio del año hasta el día con el que se están calculando los parámetros solares y almacenar el valor en *num_days*. Después se definen las ecuaciones:

$$B = \left(\frac{360}{365}\right) \cdot (num\_days - 81) \text{ [13]}$$

Se calculan los grados de diferencia entre el meridiano de referencia (UTC) y el meridiano local (LSTM-Local Standar Time Meridian). Cada hora de diferencia son 15º.

$$LSTM = 15 \cdot huso$$

Se calcula un factor corrector en función de la excentricidad de la órbita de la Tierra y la inclinación mediante la ecuación empírica (EoT-Equation of Time) [13]:

$$EOT = 9{,}87 \cdot sen(2 \cdot B) - 7{,}53 \cdot cos(B) - 1{,}5 \cdot sen(B)$$

Se calcula el factor corrector temporal (*tc*) [13] en minutos definido como:

$$Tc = 4 \cdot (long - LSTM) + EOT$$

La hora solar será *tc* dividido por 60 para pasar a horas sumado a la hora que se quiere convertir y los minutos introducidos en la variable *minute*. Seguidamente se calcula el ángulo horario en la sección *hour* mediante la hora solar (ahora variable *hour*):

$$W = (hour - 12) \cdot 15 \text{ [9]}$$

En caso de no calcular la hora solar, como se está trabajando en pasos minutales, se debe sumar la fracción de hora que representan los minutos del paso en el que se encuentra la simulación en ese instante. La variable *minute* guarda el valor de paso minutal en ese instante, por tanto, se divide entre 60 y se suma a la variable *hour*.

$$W = \left(hour + \left(\frac{minute}{60}\right) - 12\right) \cdot 15 \text{ [9]}$$

Para el cálculo de la elevación solar y del ángulo cenital se recurre a las siguientes fórmulas:

$$sen(\theta) = cos(decl) \cdot cos(lat) \cdot cos(W) + sen(decl) \cdot cos(decl) \text{ [14]}$$

Decl: declinación.
Lat: latitud.
W: ángulo horario
θ: elevación solar.

Tabla 5: variables para el cálculo de la elevación solar.

El ángulo cenital se puede calcular a partir de la elevación solar, ya que son complementarios:

$$ZEN = \frac{\pi}{2} - \theta \text{ [15]}$$

Por último, la función calcula el acimut:

$$SUN\_AZ = arcsen(\frac{cos(DECL) \times sen(W)}{\cos(\theta)})$$

Todos los parámetros calculados se devolverán en radianes.

3.3.8 Función waterFromGrid_v3_min.

Esta función es modificación de *waterFromGrid_v3* elaborada por CIMAV y recibe la variable de ubicación *file_meteo* para leerla con el comando *np.loadtxt*. También recibe la variable *itercontrol*. Su funcionamiento se basa en leer la columna de temperatura del TMY que en caso de ser paso diezminutal o quinceminutal será la 10. A continuación se calcula la media de la temperatura anual se almacena en la variable *TambAverage* al igual que se calcula el máximo (*TambMax*). Se simulará con agua de la red a una misma temperatura para cada día, definida por la siguiente ecuación:

$$T\_in\_C\_AR = [(TambAverage + offset) + ratio \cdot \left(\frac{TambMax}{2}\right) \cdot sen(\frac{\pi}{180} \cdot (-90 + (day - 15 - lag) \cdot \frac{360}{365}))]$$

$$offset = 3$$

$$ratio = 0.22 + 0{,}0056 \cdot (TambAverage - 6{,}67)$$

$$lag = 1{,}67 - 0{,}56 \cdot (TambAverage)$$

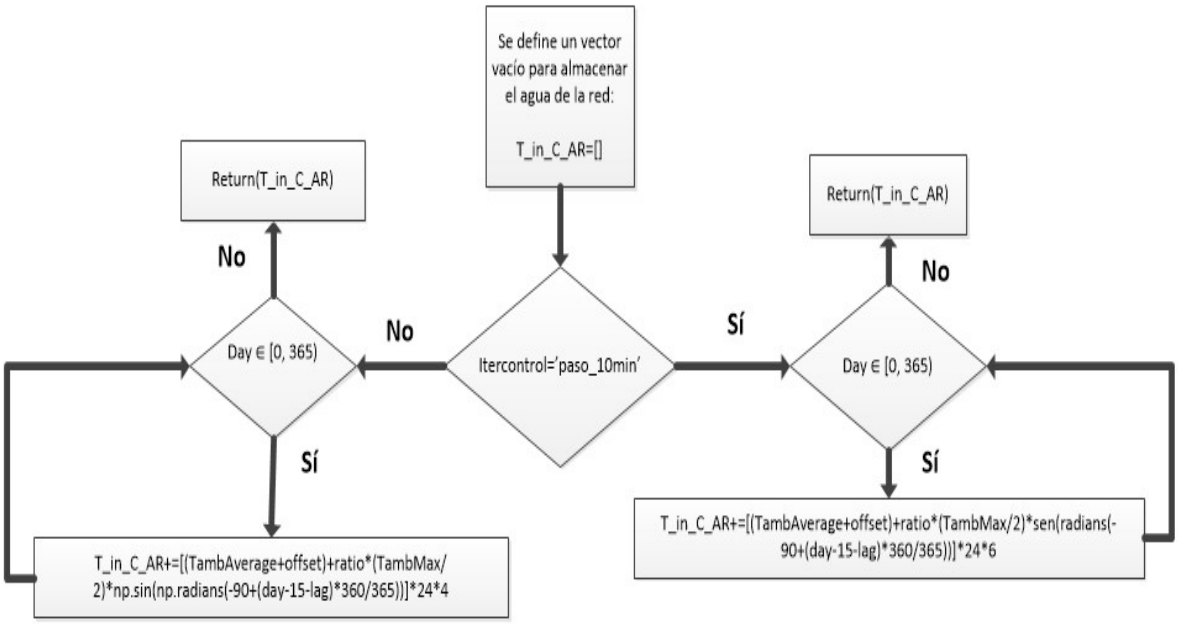

Figura 21: diagrama de flujo de la función *waterFromGrid_v3_min*.

La temperatura resultante se multiplicará por 24 horas y por 6 si es diezminutal o por 4 si es quinceminutal para cada día del año obteniéndose un vector de 52560 para diezminutal o 35040 para quinceminutal.

3.3.9 Función waterFromGrid_trim2.

Esta función es modificación de *waterFromGrid_trim* elaborada por CIMAV. Recibe el vector *T_in_C_AR* con la temperatura de cada día de la red elaborada por *waterFromGrid_v3_min* y las variables de inicio y finalización de simulación. La función elabora un vector únicamente con los pasos para la simulación según cuando empiece y acabe ésta. Para ello llama a la función *calc_min_year* para saber cuántos pasos hay desde el inicio de año hasta que se empieza (*min_year_ini)* y cuantos desde el inicio de año hasta que se acaba (*min_year_fin)*. La diferencia de ambos valores será el número de pasos que tenga la simulación (*sim_steps*) que se usará para fabricar un vector llamado *T_in_C_AR_trim* que constará de tantas componentes como pasos marque *sim_steps* y leerá *T_in_C_AR* desde la iteración de comienzo (*min_year_ini)* hasta el final de simulación.

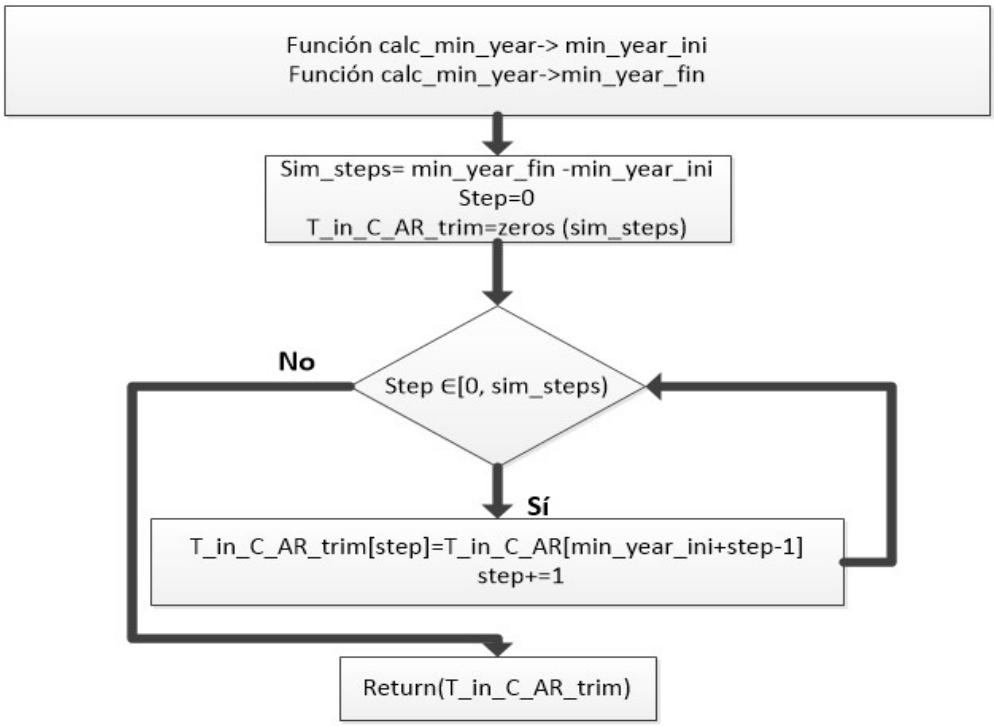

Figura 22: diagrama de flujo de la función *waterFromGrid_trim2.*

3.3.10 Función arraysMonth2.

Esta función auxiliar es una modificación de *arraysMonth*. Esta función únicamente se usa para simulaciones anuales completas y es llamada por otra función auxiliar que genera la gráfica de producción anual mostrando la producción total de cada mes, *prodMonths2*. Recibe los vectores anuales de energía solar producida total, la energía útil solar, irradiancia, demanda y la variable *itercontrol* y devuelve 4 vectores. Cada una de las 12 componentes de cada vector es la suma de la energía demandada, irradiancia o energía producida de todas las iteraciones de cada mes. Para realizar esta tarea, se crea la variable *steps* en función de *itercontrol* que será el número de pasos que tiene la simulación anual: 52.560 o 35.040. Según *itercontrol*, también se define una variable llamada *factor* que servirá para pasar de horas a pasos diezminutales si es 6 o a quinceminutal si es 4. A continuación, se definen 4 vectores para cada mes cuyas longitudes dependerá del tipo de paso escogido. Así, los vectores de paso diezminutal tendrán dimensión 52.560 y los quinceminutales tendrán 35.040. A continuación, se recorre un bucle anual que recorre los vectores importados y en función de del número de iteraciones recorridas se almacenarán los datos de dichos vectores en los vectores de un mes determinado. Por ejemplo, si en el bucle se han recorrido un número de iteraciones que sea mayor que 31×24×factor y menor que

59×24×factor se almacenarán en los vectores de febrero. Por último, se suman las componentes de cada vector y se devuelven los 4 vectores (demanda, energía producida, energía útil e irradiancia).

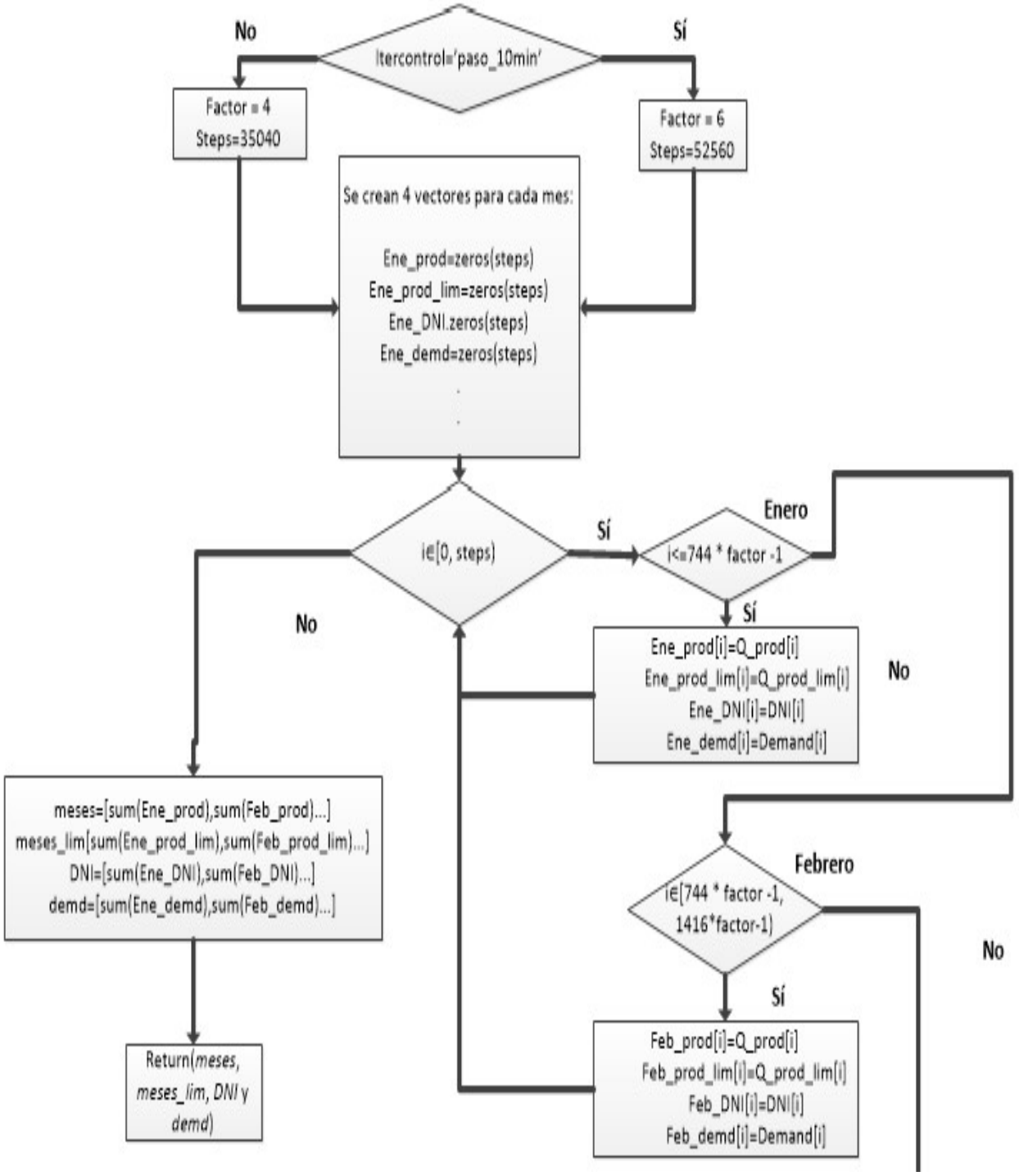

Figura 23: diagrama de flujo de *arraysMonth2*.

3.3.11 Arrays_Savings_Month2.

Esta función es similar a *arraysMonth2* y es llamada por la función generadora de gráfico *savingsMonths2*. Únicamente se usa en simulaciones anuales completas. Recibe el vector de demanda (*Demand*), *itercontrol*, vector de energía útil solar (*Q_prod_lim*), precio del kWh de combustible (*Fuel_price*) y le eficiencia de la caldera (*Boiler_eff*). Las modificaciones respecto a *arrays_savings_Month* son similares a las detalladas en

arraysMonth2. La tarea de esta función es calcular el coste que supone los kWh del vector demanda si se obtuviera la energía demandada directamente con combustible de caldera. Además, se calcula el dinero ahorrado por la energía consumida mediante recurso solar y no mediante combustible. Para ello, después de definir 3 vectores para cada mes similares a los declarados en *arraysMonths2*, se recorrerá un bucle y dependiendo del número de iteraciones recorridas se almacenará el coste y el ahorro de cada iteración según la energía útil y la demanda de dicha iteración en el vector del mes correspondiente. Para ello en cada iteración se almacenará:

$$coste\ evitado\ (€) = precio_{combustible} \cdot Q_{útil} \cdot \frac{1}{eficicencia_{caldera}}$$

$$Demanda\ (€) = precio_{combustible} \cdot demanda\ (kWh)$$

$$ahorro = \frac{coste\ evitado}{demanda}$$

Se devolverán 3 vectores cuyas 12 componentes sumen las iteraciones del mes correspondiente.

3.3.12 Funciones auxiliares para producción y recirculación.

Estas funciones auxiliares se utilizan cuando se cumplen las condiciones necesarias para que el proceso entre en producción o recirculación: elevación solar mayor que 0 e irradiancia mayor que el mínimo impuesto. A continuación, se explica en funcionamiento de las más importantes y después se explican las modificaciones a realizar.

3.3.12.1 Función operationSimple.

Esta función se usa en todos los modelos de integración excepto en los modelos de suministro directo de vapor SL_S_PD, SL_S_PD_OT y SL_S_PDS. En esta función se calcula para cada iteración determinada la potencia producida, las pérdidas térmicas al ambiente y el caudal de fluido que circula por cada lazo.

El primer paso es establecer la temperatura de entrada al lazo según haya habido recirculación en la iteración anterior o no. El requisito para que los lazos entren en recirculación es que el caudal de fluido resulte menor que el mínimo impuesto por la bomba (fijado en el subbloque 1.2). La temperatura de entrada al campo solar se fija en un principio como la temperatura de salida del proceso industrial, que es fijada por el usuario en el subbloque 1.3:

$$T_{entrada\ del\ campo\ solar} = T_{salida\ del\ proceso}(fijada\ por\ usuario)$$

Sin embargo, en función de la integración que se esté usando, esta temperatura puede cambiar debido a existencia de almacenamiento, intercambiador, etc. Dado que existe la posibilidad de recirculación en el lazo y la posibilidad de importar agua de la red externa en el caso de usar agua o vapor, se establece dos opciones para la temperatura de entrada al campo solar:

- Proceso con fluido cualquiera (agua, vapor, sales fundidas o aceite térmico) sin importar de la red: en este supuesto se consideran dos posibilidades. Si hay recirculación y la temperatura de salida del campo

solar en la iteración anterior es mayor que la temperatura de entrada del campo solar fijada por el usuario (diseño), se tomará el valor de la temperatura de salida de la iteración anterior como nueva temperatura de entrada. En caso de no cumplirse estos dos requisitos, la temperatura de entrada seguirá siendo la fijada:

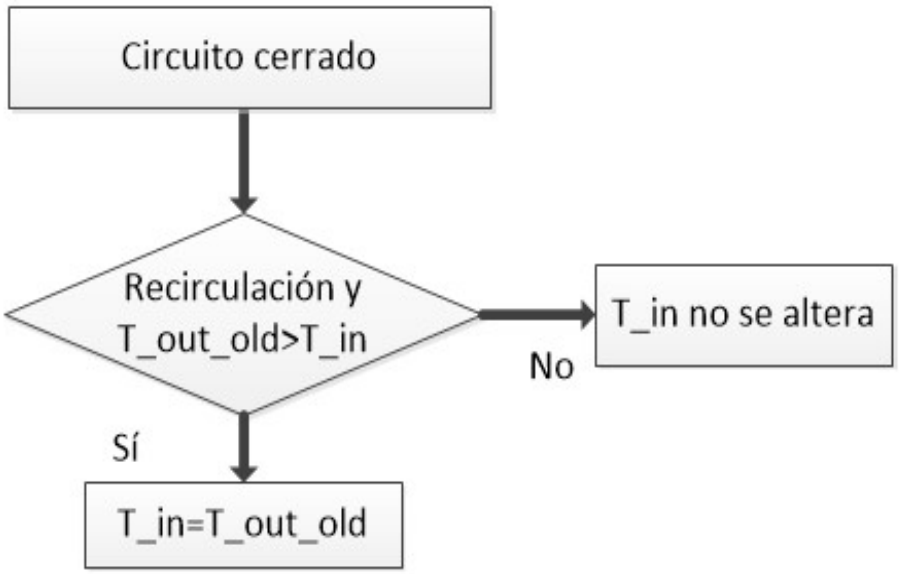

Figura 24: temperatura de entrada al campo solar

- Proceso con agua importada de la red (circuito abierto): en este supuesto el agua que entra al campo solar tiene la temperatura del agua de la red si no se cumplen las dos condiciones expuestas en el punto anterior:

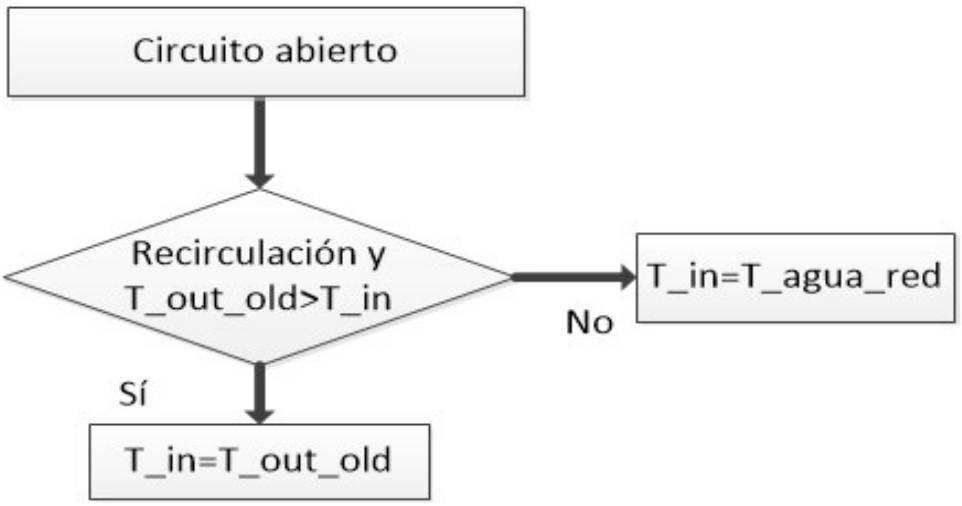

Figura 25: temperatura de entrada al campo solar con agua extraída de la red.

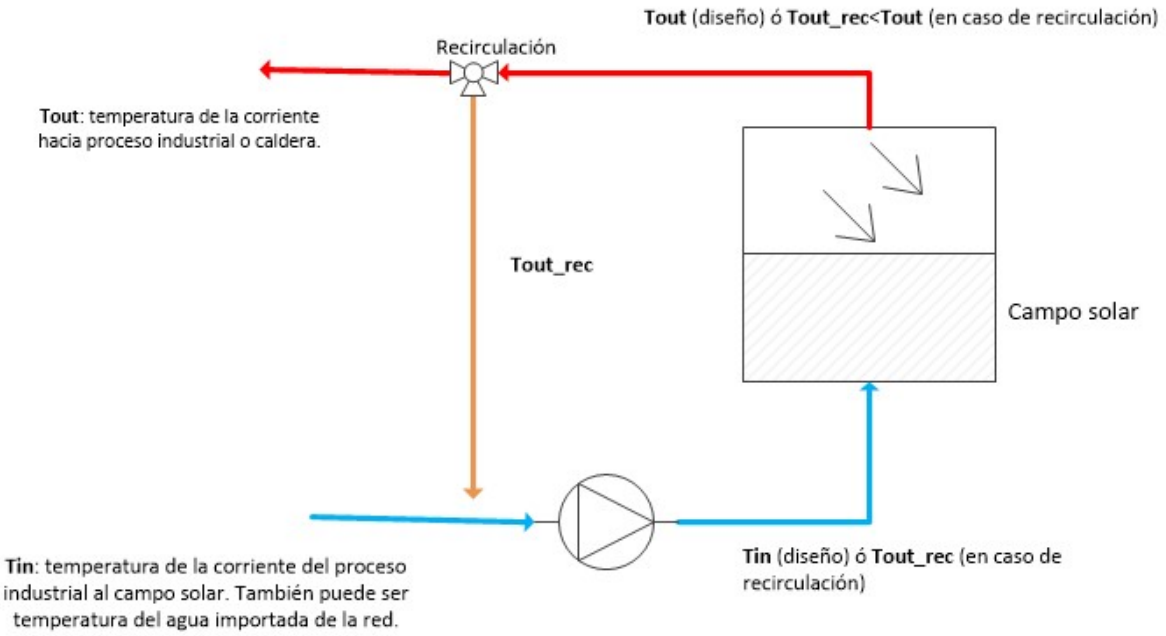

Figura 26: temperaturas del campo solar.

Tras establecer la temperatura de entrada al campo solar, se calculan las propiedades del fluido a su paso por los colectores para una temperatura media entre la entrada y la salida. Así, en cada iteración del bucle se usarán las siguientes ecuaciones según el fluido:

- Aceite:

$$Densidad\ \left(\frac{kg}{m^3}\right) = -0{,}6388 \cdot T + 885{,}61$$

$$C_P\ \left(\frac{kg}{kg \cdot K}\right) = 0{,}0038 \cdot T + 1{,}8074$$

$$Conductividad\ \left(\frac{W}{m \cdot K}\right) = 9 \cdot 10^{-5} \cdot T + 0{,}1376$$

$$Viscosidad\ dinámica\ \left(\frac{kg}{m \cdot s}\right) = (23.428{,}38511 \cdot T^{-1{,}89020}) \cdot 10^{-3}$$

$$Difusividad\ térmica\ \left(\frac{m^2}{s}\right) = 8{,}20353 \cdot e^{(-0{,}00135 \cdot T)} \cdot 10^{-8}$$

$$Prandtl\ \left(\frac{m^2}{s}\right) = 177.506{,}92794 \cdot T^{-1.68808}$$

- Sales fundidas:

$$Densidad\ \left(\frac{kg}{m^3}\right) = -0{,}636 \cdot T + 2.090$$

$$C_P\ \left(\frac{kg}{kg \cdot K}\right) = 0{,}000172 \cdot T + 1{,}44$$

$$Conductividad\ \left(\frac{W}{m \cdot K}\right) = 1{,}89 \cdot T + 0{,}441$$

$$Viscosidad\ dinámica\ \left(\frac{kg}{m \cdot s}\right) = (23.428{,}38511 \cdot T^{-1.89020}) \cdot 10^{-3}$$

- Agua: únicamente es necesario calcular el C_p que se obtiene de la librearía IAPWS97, introduciendo la temperatura media y la presión de operación definida en el subbloque 1.3.

Tras calcular las propiedades del fluido, se calculan las perdidas en el tupo receptor de cada colector. Las pérdidas serán, como se mencionó en la introducción, térmicas, ópticas y geométricas.

Las pérdidas térmicas se deben principalmente a la diferencia de temperatura con el ambiente:

$$Q \sim \sigma \cdot (T^4_{superficie} - T^4_{ambiente})$$

Las pérdidas geométricas se cuantifican con el ya explicado IAM, calculado con los ángulos de incidencia. Y, por último, las pérdidas ópticas se cuantifican con el parámetro denominado "*rendimiento óptico pico*". Para calcular dicho parámetro se debe tener en cuenta 4 factores:

- Las superficies de los espejos de captación reflejarán la radiación dependiendo de un parámetro llamado reflectividad (ρ)**,** que dependerá del material del espejo y se ve disminuido por el ensuciamiento del espejo.
- El tubo receptor cuenta con una cubierta externa de protección, de un material transparente. Esta cubierta disminuye la radiación que llega al tubo que se encuentra en el interior. Este efecto se cuantifica mediante un factor de transmisividad (τ) del material de esta cubierta externa.
- El material del tubo receptor debe tener una absortividad (α) elevada para que la radiación pueda ser transferida al fluido.
- Al conjunto de factores externos a los materiales, como pueden ser imperfecciones, errores de ubicación y obstáculos entre los espejos y el tubo se cuantifican mediante un factor de interceptación (γ).

Todos estos factores afectan al "*rendimiento óptico pico"*:

$$\eta_{opt,0^o} = (\rho_{espejo} \cdot ensuciamiento) \cdot \alpha_{tubo} \cdot \tau_{cubierta\ externa} \cdot \gamma_{factores\ externos\ a\ materiales}$$

El ángulo de incidencia también tiene efecto sobre las propiedades que afectan al $\eta_{opt,0^o}$, que está considerado para un ángulo de incidencia de 0º.

Así pues, en SHIPCal las pérdidas se reflejarán con dos ecuaciones. La primera refleja para un tubo receptor Schott PTR70, las pérdidas térmicas y geométricas, en función de la diferencia de temperatura del fluido a la salida del tubo con la temperatura del ambiente, la irradiancia incidente y ángulo de incidencia longitudinal:

$$Pérdidas_{tubo}\ [\frac{W}{m}] = 0{,}00154 \cdot AT^2 + 0{,}02021 \cdot AT - 24{,}899 + \left((0{,}00036 \cdot AT^2) + 0{,}2029 \cdot AT + 24{,}899\right) \cdot \left(\frac{DNI}{900}\right) \cdot \cos\left(theta\_long\right)$$

$$\Delta T = T_{salida\ del\ tubo} - T_{ambiente}$$

La segunda ecuación en la que se reflejan las pérdidas es la ecuación que calcula la radiación que llega realmente al fluido, influenciada por el IAM y el rendimiento óptico pico (pérdidas geométricas y ópticas):

$$Q_{real\ incidente\ en\ fluid}\ \ [kW] = DNI \cdot IAM \cdot Area_{colector} \cdot \eta_{opt,0}$$

Las pérdidas totales en cada lazo serán:

$$Perdida_{total,lazo}\,[W] = Pérdidas \cdot longitud_{tubo} \cdot N^{\underline{o}}{}_{colectores/lazo}$$

Por último, es necesario calcular el caudal del fluido a través de cada lazo:

$$Flow_{lazo}\ [\frac{kg}{s}] = \frac{DNI \cdot IAM \cdot Area_{lazo} \cdot \eta_{opt,0} - Perdida_{total,lazo}}{(T_{salida} - T_{entrada}) \cdot Cp \cdot 1.000}$$

Donde $Area_{lazo}$es el área de captación de todos los colectores en un lazo y $\eta_{opt,0}$ el rendimiento óptico pico, ambos valores definidos en el subbloque 1.3.

A partir del valor del caudal, se establece si el estado de operación es producción o recirculación según sea dicho caudal mayor o menor que el mínimo, respectivamente:

- En caso de trabajar con estado de recirculación, el flujo recirculado viene definido por un coeficiente de recirculación impuesto en el subbloque 1.2 por el usuario. Como en este caso el caudal es mayor que el calculado originalmente ($Flow$) para poder cumplir el mínimo impuesto por la bomba.

  $$flow_{rec} = coef_{recirculación} \cdot caudal_{mínimo\ bomba}$$

  Se debe recalcular las pérdidas térmicas y la temperatura de salida (que será menor), mediante un proceso iterativo. Una vez hecho esto, se procede a calcular el calor absorbido. Para ello se usa un balance de energía en el fluido a su paso por un lazo.

Para calcular el calor absorbido, la temperatura de salida será la calculada en el proceso iterativo y la temperatura de entrada será la de diseño o la de salida de la iteración anterior en caso de que ésta estuviera también en estado de recirculación, como se ha explicado al principio de este apartado. En el caso de que la iteración anterior también estuviera en recirculación, se debe sumar el calor absorbido en dicha iteración. El calor absorbido será:

$$Q_{lazo}\,[kW] = Flow_{rec,lazo} \cdot Cp_{medio} \cdot \left(T_{salida,rec} - T_{entrada,rec}\right) + Q_{lazo,i-1}$$

- En caso de estar en modo de producción, la temperatura de entrada se establecerá según el mismo criterio que en el caso de recirculación, tal como se ha explicado al principio del apartado. La temperatura de salida será la de diseño y el caudal será el calculado ($Flow$). Si en la iteración anterior hubo recirculación, se sumará a la potencia absorbida en la iteración actual:

$$Q\,[kW] = Flow_{lazo} \cdot Cp_{medio} \cdot (T_{salida} - T_{entrada}) \cdot N^{\underline{o}}{}_{lazos} \cdot MOF_{prod} + Q_{lazo.i-1} \cdot N^{\underline{o}}{}_{lazos} \cdot MOF_{prod}$$

Esta potencia será la que llevará el fluido al proceso industrial desde el campo solar. Como se observa, aquí se aplica el modificador de producción definido en el diccionario *modificators* que recibe SHIPCal. Para tener las pérdidas totales del campo solar, las pérdidas totales de cada lazo se multiplican por el número de lazos y se le suma el calor producido que se pierde por el modificador de producción para tener las pérdidas totales del campo solar.

$$Perdida_{total}\,[kW] = \frac{Perdida_{total,lazo}}{1.000} \cdot N^{\underline{o}}{}_{lazos} + Q \cdot \left(1 - MOF_{prod}\right)$$

3.3.12.2 Función outputStorageSimple.

Tras calcular el calor absorbido en *OperationSimple*, se usa esta función en caso de que el campo solar disponga de almacenamiento. Dispone de tres modos de funcionamiento:

- Descarga completa: la suma del calor absorbido en los lazos y la energía almacenada en el acumulador es menor que la demanda. En este caso, la energía que se llevará al proceso industrial será la suma de la energía absorbida y el total de energía almacenada en el acumulador y, por tanto, la energía almacenada pasará a ser 0.

$$Q[kWh] + Energía_{almacenada}[kWh] < Demanda[kWh]$$

$$Q_{útil}[kWh] = Q[kWh] + Energía_{almacenada}[kWh]$$

- Descarga parcial: el calor absorbido en los lazos es menor que la demanda, pero la suma de dicho calor y de la energía almacenada es mayor que la demanda. Para este caso, la descarga de energía acumulada

será la diferencia entre la demanda y el calor absorbido en los lazos. El calor útil que va al proceso será la demanda, pues se puede suplir ésta completamente:

$$Q[kWh] + Energía_{almacenada}[kWh] > Demanda[kWh] \ \& \ Q[kWh] < Demanda[kWh]$$

$$Energía_{almacenada,2}[kWh] = Energía_{almacenada}[kWh] - (Demanda - Q)[kWh]$$

$$Q_{útil}[kWh] = Demanda[kWh]$$

- Carga de energía en acumulador: el calor absorbido en los lazos es mayor que la demanda. En caso de que haya espacio para almacenar más energía, se almacenará la diferencia entre el calor absorbido en los lazos y la demanda.

$$Q[kWh] \geq Demanda\ [kWh] \ \& \ (Q - Demanda) + Energía_{almacenada} < Capacidad_{max}$$

$$Energía_{almacenada,2}[kWh] = Energía_{almacenada}[kWh] - (Q - Demanda)[kWh]$$

$$Q_{cargado}\ [kWh] = (Q - Demanda)[kWh]$$

$$Q_{útil}[kWh] = Q[kWh] - Q_{cargado}\ [kWh]$$

En caso de no haber espacio suficiente, lo que sobra se considera energía desenfocada (pérdidas). El calor que va al proceso será la diferencia entre el calor absorbido en los lazos y el calor almacenado y desenfocado:

$$Q[kWh] \geq Demanda\ [kWh] \ \& \ (Q - Demanda) + Energía_{almacenada} > Capacidad_{max}$$

$$Energía_{almacenada,2}[kWh] = \ Capacidad_{max}$$

$$Q_{cargado}\ [kWh] = Capacidad_{max} - \ Energía_{almacenada}$$

$$Q_{desenfocado}[kWh] = Q[kWh] - Demanda\ [kWh] - Q_{cargado}\ [kWh]$$

$$Q_{útil}[kWh] = Q[kWh] - Q_{cargado}\ [kWh] - Q_{desenfocado}[kWh]$$

3.3.12.3 Función outputWithoutStorageSimple.

Esta función se usa cuando no hay almacenamiento, tras haber usado *operationSimple*. Simplemente sirve para establecer si hay calor desenfocado o no:

$$Q\,[kWh] < Demanda[kWh]$$

$$Q_{útil}[kWh] = Q[kWh]$$

Si el calor absorbido en los alzos es mayor que la demanda, habrá desenfoque:

$$Q[kWh] > Demanda[kWh]$$

$$Q_{útil}[kWh] = Demanda[kWh]$$

$$Q_{desenfoque}[kWh] = Q[kWh] - Demanda[kWh]$$

3.3.12.4 Función operationDSG.

Esta función se utiliza para el modo de integración SL_S_PD y SL_S_PDS, en el que se trabaja con vapor. Tiene una estructura similar a *operationSimple*. La temperatura de entrada al campo solar se fija en un principio como la temperatura de salida del proceso industrial, que es fijada por el usuario en el subbloque 1.3.

En función del tipo de integración, se mantendrá así, o tomará otro valor inicial, debido a acumulador, intercambiador, etc.

$$T_{entrada\ del\ campo\ solar} = T_{salida\ del\ proceso}(fijada\ por\ usuario)$$

En caso de que haya recirculación, se establece dicha temperatura de entrada como la temperatura de salida del campo solar en la iteración anterior. El título de vapor de la corriente de salida se ha fijado en 0.8:

$$\boxed{x_{salida} = 0{,}8}$$

Con este título de vapor y la presión de operación, se establece la entalpía a la salida para poder calcular el caudal. Así, se calcula la potencia pérdida por metro de tubo receptor para poder obtener el caudal, igual que en *operationSimple*:

$$Pérdidas\,[\tfrac{W}{m}] = 0.00154 \cdot AT^2 + 0.02021 \cdot AT - 24.899 + \left((0.00036 \cdot AT^2) + 0.2029 \cdot AT + 24.899\right) \cdot \left(\tfrac{DNI}{900}\right) \cdot \cos\left(theta\_long\right)$$

$$AT = T_{salida} - T_{ambiente}$$

$$Perdida_{total,lazo}\,[W] = Pérdidas \cdot longitud_{tubo} \cdot N^{o}{}_{\frac{colectores}{lazo}}$$

Ahora se procede a calcular el caudal del fluido a través de cada lazo:

$$Flow_{lazo}\,[\frac{kg}{s}] = \frac{DNI \cdot IAM \cdot Area_{lazo} \cdot \eta_{opt,0} - Perdida_{total,lazo}}{(h_{salida} - h_{entrada}) \cdot 1000}$$

- Igual que en *operationSimple*, si el caudal obtenido es menor que el impuesto por la bomba se debe recircular. En ese caso se utiliza un caudal de recirculación y se calcula la nueva entalpía de salida:

$$flow_{recirculación} = coef_{recirculación} \cdot caudal_{mínimo\ bomba}$$

$$h_{salida,rec}[\frac{kJ}{kg}] = (\left(\frac{DNI \cdot IAM \cdot Area_{lazo} \cdot \eta_{opt,0} - Perdida_{total,lazo}}{Flow_{lazo}}\right) \cdot \frac{1}{1000}) + h_{entrada}$$

  Con esta nueva entalpía de salida y la presión de operación, que no ha variado, se calcula el nuevo título de vapor para la recirculación. En función del título de vapor se procederá de una manera u otra. Si el fluido es líquido, las pérdidas, la temperatura de salida y el calor producido en la recirculación se calculan igual que en *operationSimple.* Si el fluido es bifásico, debe recalcularse el caudal, refrigerando para que el fluido pase a líquido subenfriado, pues en estado bifásico puede dañar la bomba al recircular. Para ello, el calor producido se calcula con la fórmula de la radiación incidente y las pérdidas:

$$Q_{rec}[kW] = \left(DNI \cdot IAM \cdot Area \cdot rho_{optic_0} - Q_{loss_{rec}} \cdot n_{coll_{loop}} \cdot Long\right) \cdot \frac{mofProd}{1000}$$

  Y tras ello, se calcula el caudal, dividiendo esta energía entre la diferencia de entalpías de líquido subenfriado y la entalpía de entrada:

$$h_{liq.subenfriado}[\frac{kJ}{kg}] = h_{agua}(P = P_{operación}, T = T_{x=0} - T_{subenfriamiento})$$

$$flow_{recirculación}[\frac{kg}{s}] = \frac{Q_{rec}}{h_{liq.subenfriado}}$$

  El calor total será el absorbido en esta iteración sumado al calor absorbido en las iteraciones anteriores de esta recirculación. Si en la iteración anterior el estado de operación era en producción, no se suma nada.

$$Q_{rec.total}[kW] = Q_{rec} + Q_{rec,i-1} \rightarrow Si\ en\ i-1\ no\ hubo\ recirculación, Q_{rec,i-1} = 0$$

- En caso de ser en modo producción, el calor total será el producido más el acumulado en caso de haber recirculación en la iteración anterior. El caudal, las pérdidas y las temperaturas serán las calculadas originalmente al principio de la función:

$$Q_{total}[kW] = Flow_{lazo} \cdot (h_{salida} - h_{entrada}) \cdot Nº_{lazos} \cdot MOF_{prod} + Q_{rec.total,i-1} \cdot Nº_{lazos}$$

Las pérdidas totales serán:

$$Perdida_{total}\,[kW] = \frac{Perdida_{total,lazo}}{1000} \cdot Nº_{lazos} + Q_{total} \cdot (1 - MOF_{prod})$$

3.3.12.5 Modificaciones en las funciones auxiliares de producción.

Las funciones creadas como modificación son *operationSimple2, operationDSG2.*

```
330            Q_prod=0 #No h
331            Q_p    ec=Q_prod_rec/factor
332            Q_prod_rec=Q_prod_rec+Q_prod_rec_old #The Q_prod_re is per serie or loop and it will be mutiplied by 
333            bypass.append("REC")
334            newBypass="REC"
335            Perd_termicas = Perd_termicas*num_loops/factor #Tak into account all the loops, before it was only t
336
337        else:
338            #PRODUCCION
339            if sender == 'CIMAV':
340                Q_prod+=Q_prod_rec_old*num_loops
341                Q_prod*=mofProd
342            else:
343                if fluidInput=="water" or fluidInput=="steam":
344                    outlet=IAPWS97(P=P_op_Mpa, T=T_out_K)
345                    h_out_kJkg=outlet.h
346                    #Cp_av_kJkgK = IAPWS97(P=P_op_Mpa, T=0.5*(T_out_K+T_in_K)).cp
347                    if bypass_old=="REC" and Q_prod_rec_old>0:
348                        Q_prod=flow_rate_kgs*(h_out_kJkg-h_in_kJkg)*num_loops*mofProd+Q_prod_rec_old*num_loops*mofP
349                        #Q_prod=flow_rate_kgs*(Cp_av_kJkgK)*(T_out_K-T_in_K)*num_loops*mofProd+Q_prod_rec_old*num_
350                    else:
351                        Q_prod=flow_rate_kgs*(h_out_kJkg-h_in_kJkg)*num_loops*mofProd #In kW
352                        #Q_prod=flow_rate_kgs*(Cp_av_kJkgK)*(T_out_K-T_in_K)*num_loops*mofProd
353
354                if fluidInput=="oil":
355                    if bypass_old=="REC":
356                        if Q_prod_rec_old>0:
357                            Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd+Q_prod_rec_old*num_loops
358                        else:
359                            Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd#In kWh
360                    else:
361                        Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd #In kW
362
363                if fluidInput=="moltenSalt":
364                    if bypass_old=="REC":
365                        if Q_prod_rec_old>0:
366                            Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd+Q_prod_rec_old*num_loops
367                        else:
368                            Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd#In kWh
369                    else:
370                        Q_prod=flow_rate_kgs*Cp_av*(T_out_K-T_in_K)*num_loops*mofProd #In kW
371
372            bypass.append("PROD")
373            newBypass="PROD"
374            Q_prod=Q_prod/factor
375            Perd_termicas = Perd_termicas*num_loops/factor + (1-mofProd)*Q_prod #Takes into account all the loops,
376            flow_rate_rec=0
377            Q_prod_rec=0
378            #Perd_termicas [kW]
```

Figura 27: se añade la variable factor a *operationSimple2.*

Las modificaciones en las funciones de producción se basan únicamente en dividir las variables de energía producida, recirculada, etc., entre un factor que será 6 si es paso diezminutal o 4 si es quinceminutal para poder pasar de unidades de potencia a kWh. En todas las funciones se declara inicialmente la variable *factor* que tendrá un valor u otro en función de *itercontrol.* Tras esto, las funciones calcularán primero las pérdidas al ambiente y el caudal necesario para cumplir la diferencia de temperaturas impuesta en el bloque 1.3 para después calcular la energía producida según el modo de operación (producción o recirculación).

```
                else:
                    flow_rate_rec=0

                Q_prod=0
                T_out_K=outlet.T
                Q_prod_rec=Q_prod_rec*num_loops*mofProd # Total Q_prod_rec in the field
            Q_prod_rec=Q_prod_rec/factor
            Q_prod_rec=Q_prod_rec+Q_prod_rec_old
            bypass.append("REC")
            #newBypass="REC" #Not used
        else:
            if bypass_old=="REC":
                if Q_prod_rec_old>0:
                    Q_prod=flow_rate_kgs*(outlet.h-h_in_kJkg)*num_loops*mofProd+Q_prod_rec_old #In kW
                else:
                    Q_prod=flow_rate_kgs*(h_out_kJkg-h_in_kJkg)*num_loops*mofProd
            else:
                x_out=x_desing
                outlet=IAPWS97(P=P_op_Mpa, x=x_out)
                Q_prod=flow_rate_kgs*(outlet.h-h_in_kJkg)*num_loops*mofProd #In kW
            Q_prod=Q_prod/factor
            x_out=x_desing
            T_out_K=IAPWS97(P=P_op_Mpa, x=x_out).T
            Q_prod_rec=0
            flow_rate_rec=0
            bypass.append("PROD")
            #newBypass="PROD" #Not used
        if Perd_termicas==0:
            Perd_termicas=Q_loss_rec*n_coll_loop*Long/1000
        Perd_termicas=Perd_termicas/factor
        return [flow_rate_kgs,Perd_termicas,Q_prod,T_in_K,x_out,T_out_K,flow_rate_rec,Q_prod_rec,bypass]
```

Figura 28: variable *factor* en *operationDSG2*.

3.3.13 Funciones auxiliares para generar gráficos.

Además de *Arrays_Savings_Month2* y *arraysMonth2*, hay otras funciones generadoras de gráficos que deben modificarse introduciendo un factor (6 diezminutal o 4 quinceminutal) para modificar la dimensión de los vectores a crear tras leer los resultados del bloque 2.

```
def storageSummer2(itercontrol,sender,origin,lang,Q_prod,Q_charg,Q_prod_lim,Q_useful,Demand,Q_defocus,Q_discha
    fig = plt.figure(figsize=(14, 3.5))
    #np.array(in list) is because Django need it since Q_prod, Q_prod_lim,.. are passed as lists
    if origin==-2 or origin == -3:
        fig.patch.set_alpha(0)
    if itercontrol=='paso_10min':
        factor=6
    elif itercontrol=='paso_15min':
        factor=4
    if lang=="spa":
        fig.suptitle('Almacenamiento primera semana Junio', fontsize=14, fontweight='bold',y=1)
        ax1 = fig.add_subplot(111)

        plt.fill_between( np.arange(3624*factor,3624*factor+167*factor,1), Demand[3624*factor:3791*factor], c

        plt.bar((np.arange(3624*factor,3624*factor+167*factor,1)), np.array(Q_prod[3624*factor:3791*factor])-
        #ax1 .plot((np.arange(3624,3624+167,1)), Q_prod_lim[3624:3791],color = 'blue',label="Energía suministr
        #ax1 .plot((np.arange(3624,3624+167,1)), Q_useful[3624:3791],color = 'green',label="Energía útil",lin

        ax1 .plot((np.arange(3624*factor,3624*factor+167*factor,1)), Demand[3624*factor:3791*factor],color =
        if sender =='CIMAV':
            plt.bar((np.arange(3624*factor,3624*factor+167*factor,1)), Q_defocus[3624*factor:3791*factor],col
        else:
            plt.bar((np.arange(3624*factor,3624*factor+167*factor,1)), Q_defocus[3624*factor:3791*factor],col

        plt.bar((np.arange(3624*factor,3624*factor+167*factor,1)), Q_charg[3624*factor:3791*factor],color = '

        plt.bar((np.arange(3624*factor,3624*factor+167*factor,1)), Q_discharg[3624*factor:3791*factor],color

        ax1.set_ylabel('Producción & Demanda - kWh')
        ax1.set_ylim([0,np.max([np.max(Q_prod[3624*factor:3791*factor]),np.max(Demand[3624*factor:3791*factor
        ax1.set_xlim([3624*factor,3624*factor+167*factor])
        ax1.legend(loc='upper left', borderaxespad=0.).set_zorder(99)



        ax2 = ax1.twinx()
        if type_integration=="SL_L_S" or type_integration=="SL_L_S_PH":
            ax2 .plot((np.arange(3624*factor,3624*factor+167*factor,1)), np.array(T_alm_K[3624*factor:3791*fa
        ax2 .plot((np.arange(3624*factor,3624*factor+167*factor,1)), SOC[3624*factor:3791*factor],color='oran
        if itercontrol=='paso_10min':
            ax2.set_xlabel('simulación: pasos diezminutales')
```

Figura 29: función *storageSummer2* como ejemplo de aplicación del factor de conversión.

Por ejemplo, si se quiere realizar una simulación anual, la función *prodSummerPlot* genera un gráfico que sirve para analizar la demanda, irradiancia y producción en la primera semana de junio. Para ello antes leía los vectores resultados del bloque 2 desde la componente 3624 hasta 3791, pero ahora se debe multiplicar esos límites por el factor según sea el modo de paso elegido (figura siguiente). Como se ha dicho anteriormente, hay que añadir la opción de cambiar la leyenda de los ejes según el tipo de paso elegido. Las funciones creadas como modificación son *demandVsRadiation2, thetaAnglesPlot2, IAMAnglesPlot2, flowRatesPlot2, prodWinterPlot2, prodSummerPlot2, productionSolar2, storageWinter2, storageSummer2, storageNonAnnual2, storageNonAnnualSL_S_PDR2, savingsMonths2, SL_S_PDR_Plot2* y *prodMonths2.*

4 APLICACIÓN WEB.

4.1 Funcionamiento y modificaciones.

En este apartado se explica en que se basan los cambios introducidos en el front-end para simular en SHIPCal. Esta aplicación web funciona con Django, un framework web desarrollado para crear aplicaciones web de manera sencilla.

El front-end opera llamando al script de SHIPCAL.py, por lo que las variables de entrada a SHIPCal se definen externamente y no desde la terminal de introducción de variables. Opera como si se tratara de una función que trabaja de manera independiente por lo que deben definirse las variables que se envían a SHIPCal y las que dicha función devuelve al front-end. Por ello, *Django* cuenta con varias carpetas (script.py):

4.1.1 Carpeta de modelos.

El script *models.py* es la base de datos del front-end. Se definen las variables que se usan para llamar a SHIPCal. También se debe crear una base de datos para las variables que devuelva el front-end por pantalla como resultado. Por tanto, habrá dos carpetas de *models.py*:

- Carpeta de *models.py* guardado en carpeta de *simforms*: en esta carpeta de modelos se almacenan las variables que introduce el usuario para llamar a SHIPCal desde el formulario de simulación.
- Carpeta de *models.py* guardado en la carpeta de *results*: se almacenan las variables que se devolverán por pantalla.

```
8   class Simulation(models.Model):
9
10      #Who is simulating
11      name = models.CharField(_("Name"), max_length=30)
12      email = models.EmailField("e-mail",)
13      #What is simulating
14      industry = models.CharField(_("Industry"), max_length=30)
15      sectorIndustry = models.CharField(_("Industry sector"), max_length=30)
16      process = models.TextField(_("Process description"), blank=True, null=True)
```

Figura 30: definición de un modelo.

En la figura anterior se define una *clase* (modelo) denominado *simulation* que será la base de datos que almacene las variables que intervienen en la simulación y que se importan a SHIPCal, por tanto, está en la carpeta de *simforms*. Aquí también se guardarán datos identificativos, que no entran en SHIPCal, como el correo electrónico del usuario o el sector industrial. Además de éste, hay 4 modelos (bases de datos) más definidos en la carpeta de *simforms*:

-*Fuels*: sirve para agregar nuevos combustibles por parte del usuario a la base de datos.

-*FuelUnits*: almacena el factor de conversión para el combustible en caso de que el precio no venga en €/kWh. Si, por ejemplo, el coste del combustible se conoce en €/kg, se almacena aquí el factor de conversión (previamente introducido) para pasar de kg a kWh.

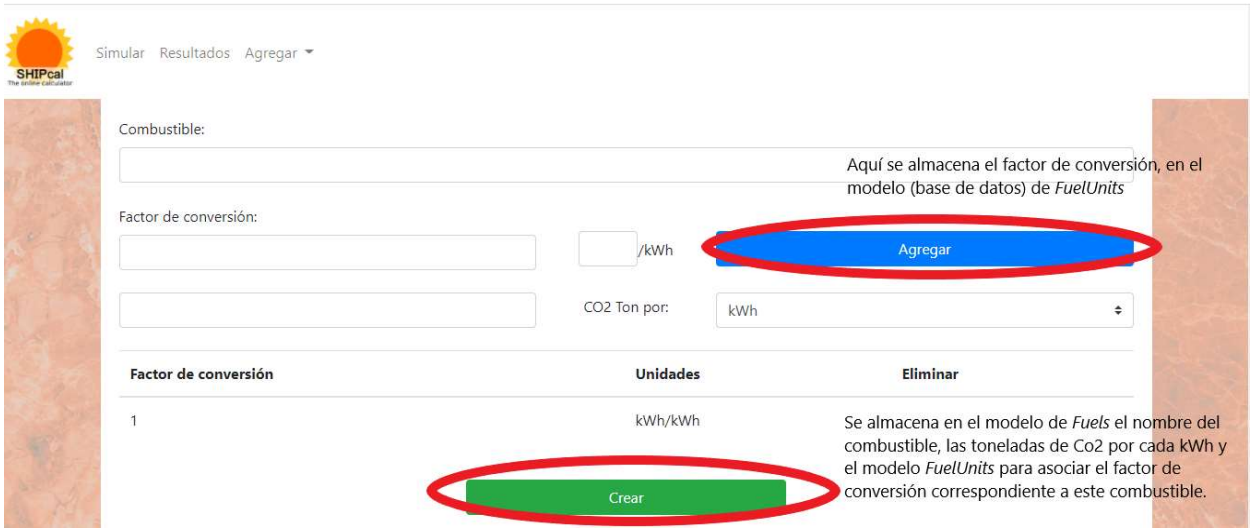

Figura 31: uso de modelos *Fuel* y *FuelUnits* para agregar un nuevo combustible.

-*Locations*: almacena las ubicaciones que se almacenen por parte del usuario, con su correspondiente TMY asociado.

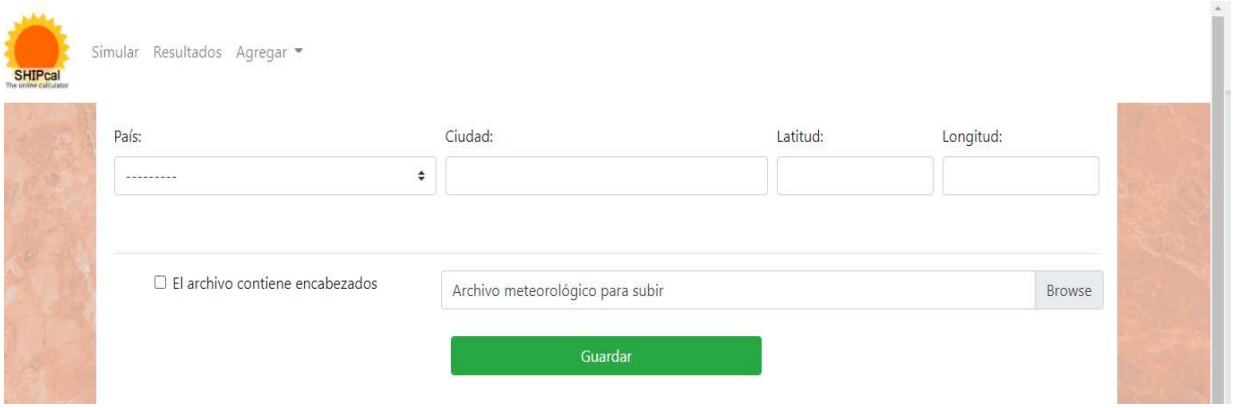

Figura 32: vista de la ventana para añadir ubicaciones con TMY.

-*MeteoData*: aquí se crean los vectores que almacenarán cada columna del TMY introducido.

De la misma forma, en la carpeta de *results* se almacenan dos modelos que recogen las variables que se representan en los gráficos e informe generados y se muestran como resultado de la simulación por pantalla. Tiene 5 modelos, los dos más importantes y que se deben modificar son:

-*PlotVars*: se almacenan las variables que devuelve SHIPCal para generar gráficos.

-*ReportVars*: se almacenan las variables que devuelve SHIPCal para generar el informe. Se recuerda que el informe solo se genera para simulaciones de un año completo.

4.1.1.1 Modificaciones en los modelos.

Se deben realizar cambios en la carpeta de modelos en la carpeta de formulario (*simforms)* y en la carpeta de resultados (*results)*.

4.1.1.1.1 Carpeta de simforms.

En esta carpeta se deben incluir las nuevas variables para paso diezminutal y quinceminutal, al igual que debe añadirse la columna de minutos del TMY. Para ello se deben modificar los modelos *Simulation*, *Locations* y *MeteoData.*

En el modelo de *Simulation* se añaden las variables *to_solartime*, *huso*, *itercontrol*, *annual*, *month_ini_sim*, *month_fin_sim*, *day_ini_sim*, *day_fin_sim*, *hour_fin_sim* y *hour_ini_sim*. La variable *annual* es una clave que se ha establecido para simular durante un año completo o durante un tiempo menor. Si dicha variable vale *yes,* simulará desde el 1 de enero hasta el 31 de diciembre. Si vale *no,* simulará en el periodo marcado por las variables *month_ini_sim*, *month_fin_sim*, *day_ini_sim*, *day_fin_sim*, *hour_fin_sim* y *hour_ini_sim*.

- Las variables *to_solartime, itercontrol*, *annual* se definirán en la base de datos como cadena de caracteres (*models.Charfield*). Se declaran con únicamente 3 valores posibles (diezminutal, quinceminutal u horario) en el caso de *itercontrol* o dos valores posibles en caso de *annual* y *to_solartime* (sí o no).

- Las variables *huso, month_ini_sim, month_fin_sim, day_ini_sim, day_fin_sim* se declaran como número enteros (*models.IntegerField*).

- Las variables *hour_fin_sim* y *hour_ini_sim* se han definido como valores con formato tiempo (*models.TimeField*). Con esto se almacenarán en estas dos variables la hora y minutos que se introducen por parte del usuario en la aplicación web:

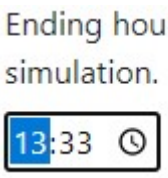

Figura 33: vista final por pantalla de *models.TimeField*.

En el modelo de *MeteoData* únicamente es necesario añadir la columna *min_sim.* En el modelo *Locations*, se debe declarar la variable de longitud de la ubicación geográfica en cuestión para calcular la hora solar. Se declara como una variable flotante (*models.FloatField*) y se almacena en la variable *lon*.

4.1.1.1.2 Carpeta de results.

Únicamente deben declararse las variables que devuelva SHIPCal. No solo se deben declarar las variables nuevas, que únicamente será *itercontrol*, sino que también es necesario declarar las variables que intervienen en los gráficos de simulaciones de duración menor de 1 año que antes no devolvía SHIPCal pues solamente se mostraban en el front-end las simulaciones de un año completo.

4.1.2 Carpeta de formularios.

Corresponde a la carpeta de *forms.py*. En esta carpeta se definen los formularios que aparecen por pantalla para que el usuario introduzca los datos. Únicamente se declaran las variables que van a aparecer, pero no como se ordenan y la disposición visual que tendrá finalmente el formulario en pantalla. Para ello, primero se asocia el modelo de *simforms* que contiene las variables, luego se definen las variables que aparecerán y después se asocia cada variable con un tipo de formulario:

```
    class Meta:
        model = Simulation
        fields = [
            'name','email','industry','sectorIndustry', 'process',
            'location', 'surface', 'distance',
            'to_solartime', 'huso',
            'co2TonPrice', 'businessModel', 'fuel_price', 'fuel', 'fuel_price_unit',
            'fluid','tempOUT','tempIN','pressureUnit', 'pressure',
            'demand', 'demandUnit', 'hourEND', 'hourINI', 'itercontrol','annual',
            'month_ini_sim', 'month_fin_sim', 'day_ini_sim', 'day_fin_sim', 'hour_fin_sim', 'hour_ini_sim',
            'Mond', 'Tues', 'Wend', 'Thur', 'Fri', 'Sat', 'Sun',
            'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec',
            'num_loops', 'n_coll_loop','connection', 'type_integration', 'almVolumen',
            'mofINV', 'mofDNI', 'mofProd',
            'orientation', 'inclination', 'shadows', 'terrain',
        ]
        widgets = {
            'orientation':forms.Select(attrs={'class':'custom-select'}),
            'inclination':forms.Select(attrs={'class':'custom-select'}),
            'shadows':forms.Select(attrs={'class':'custom-select'}),
            'terrain':forms.Select(attrs={'class':'custom-select'}),

            'itercontrol':forms.Select(attrs={'class':'custom-select'}),
            'to_solartime':forms.Select(attrs={'class':'custom-select'}),
            'huso':forms.NumberInput(attrs={'class':'form-control'}),
            'annual':forms.Select(attrs={'class':'custom-select'}),

            'num_loops':forms.NumberInput(attrs={'class':'form-control'}),
            'n_coll_loop':forms.NumberInput(attrs={'class':'form-control'}),
            'connection':forms.Select(attrs={'class':'custom-select'}),
            'type_integration':forms.Select(attrs={'class':'custom-select'}),
            'almVolumen':forms.NumberInput(attrs={'class':'form-control'}),
            'mofINV':forms.NumberInput(attrs={'class':'form-control'}),
            'mofDNI':forms.NumberInput(attrs={'class':'form-control'}),
```

Figura 34: estructura para definir un formulario que se mostrará por pantalla.

Como se observa en la figura, el formulario de simulación recogerá las variables que se introducen en él para simular. Para ello, se recoge el modelo de la carpeta de *simforms*, en este caso el modelo es *Simulation* y se almacena en *model*. Así se grabará en la base de datos los valores que se vayan introduciendo en el formulario. Seguidamente se definen las variables que aparecerán en el formulario, almacenadas en el vector *fields*, sin importar el orden con el que se definan pues esto simplemente sirve para declarar qué variables aparecen, pero no cómo se disponen en la pantalla. Por último, cada variable se asocia a un tipo de formulario. Los tres tipos de formularios que aparecen son:

- *Forms.Select*: el usuario deberá elegir entre unas opciones que se le mostrarán. Por ejemplo, deberá elegir entre los tipos de integración disponibles o el tipo de modelo de negocio.
- *Forms.NumberInput*: el usuario deberá introducir un valor numérico.
- *Forms.TextInput*: el usuario deberá introducir un texto.

Además del formulario de simulación, hay 3 formularios más. Uno sirve para agregar una nueva localización con su TMY, que usará el modelo *locations.* Los otros dos aparecen juntos, pues sirven para agregar un nuevo combustible (utilizará el modelo de *fuels*) y factor de conversión (modelo *FuelUnits*).

4.1.2.1 Modificaciones en el formulario.

Se añaden las variables que se han declarado previamente en las bases de datos (modelos). Excepto la variable *lon* que se declara en el formulario de *Locations*, el resto se define en el formulario de simulación:

- Las variables *itercontrol*, *to_solartime* y *annual* serán del tipo *forms.Select*.

- Las variables *lon*, *huso*, *month_ini_sim*, *month_fin_sim*, *day_ini_sim* y *day_fin_sim* serán del tipo *forms.NumberInput.*

- Se añade un nuevo tipo de formulario para las variables *hour_fin_sim* y *hour_ini_sim*: *forms.TimeInput.* Este tipo de formulario mostrará por pantalla el display que se muestra e la figura 25.

4.1.3 Carpeta de views.

Esta carpeta contiene la "lógica" del front-end, pues aquí es donde se utilizan las variables introducidas en el formulario y las variables devueltas por la función SHIPCal. Al igual que con la carpeta de modelos, si la carpeta de *views.py* está guardada en *simforms*, será la encargada de procesar las variables que vienen del formulario. En caso de que *views.py* esté en la carpeta de *results*, se procesarán los resultados que vienen de SHIPCal y que se mostrarán por pantalla al usuario.

En el caso de la carpeta *simforms*, cuando se han introducido las variables en el formulario, estas se graban en la base de datos (modelos) y se llevan a la carpeta de *views*, almacenándose en los diccionarios ya explicados anteriormente en esta memoria: *inputsDjango, confReport*, *modificators*, *SimControl* y *desginDict*. Como se ha dicho, estos diccionarios serán los que se introduzcan en SHIPCal.

```
43          new_sim = sim_form.cleaned_data
44          sim = Simulation(**new_sim)
45          sim.save()
46
47          lang_code=translation.get_language()
48          if lang_code=='es':
49              lang='spa'
50          else:
51              lang='eng'
52
53          inputsDjango = model_to_dict(sim)
54          inputsDjango.update({
55              'pressure':inputsDjango['pressure']*inputsDjango['pressureUnit'], #Converts into kWh
56              'demand':inputsDjango['demand']*inputsDjango['demandUnit'], #Converts into bar
57              'fuelPrice':inputsDjango['fuel_price']*sim.fuel_price_unit.conversion_factor, #converts into EUR/kWh
58              'co2factor':sim.fuel.co2factor*sim.fuel.co2units.conversion_factor, # [CO2tons/kWh]
59              'date':sim.date,'fuel':sim.fuel.fuel,'location_aux':sim.location.location_aux,
60              'hour_ini_sim':sim_form.cleaned_data['hour_ini_sim'].hour,
61              'hour_fin_sim':sim_form.cleaned_data['hour_fin_sim'].hour,
62
63          })
64          inputsDjango.update({'pressureUnit':'bar','demandUnit':'kWh','fuelUnit':'kWh',})
65
66          confReport={'lang':lang,'sender':'SHIPcal','cabecera':_('Resultados de la <br> simulación'),'mapama':0}
67          modificators={'mofINV':sim.mofINV,'mofDNI':sim.mofDNI,'mofProd':sim.mofProd}
68          desginDict={'num_loops':sim.num_loops,'n_coll_loop':sim.n_coll_loop,'type_integration':sim.type_integration,'almVolum
69
70          if inputsDjango['itercontrol']=='paso_10min' or inputsDjango['itercontrol']=='paso_15min':
71              if inputsDjango['annual']=='yes':
72                   if inputsDjango['itercontrol']=='paso_10min':
73                      simControl={'finance_study':1,'mes_ini_sim':1,'dia_ini_sim':1,'hora_ini_sim':0,'mes_fin_sim':12,'dia_fin_
74                   elif inputsDjango['itercontrol']=='paso_15min':
75                      simControl={'finance_study':1,'mes_ini_sim':1,'dia_ini_sim':1,'hora_ini_sim':0,'mes_fin_sim':12,'dia_fin_
76              else:
```

Figura 35: recogida de las variables procedentes de la base datos y creación de los vectores.

Como se observa en la ilustración, las variables que se han introducido en el formulario y se han grabado en el modelo de *Simulation*, ahora se guardan en *sim*. A partir de aquí, se actualizan determinadas variables, por ejemplo, aplicando los factores de conversión de la presión o del combustible. También se crean los vectores que se enviaran a SHIPCal y se ejecuta dicha función:

template_vars,plotVars,reportsVar,version
=SHIPcal(1,inputsDjango,[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],200,confReport,modificators,desginDict,simCo
ntrol,sim.id)

Como se observa, la primera variable correspondiente a *origin* vale 1, porque se ejecuta SHIPCal desde el front-end. El vector de *plots* contiene valores de 0 pues los gráficos se generan con el front-end y fuera de SHIPCal. La calidad de imagen será 200.

Si la carpeta de views se encuentra en la carpeta de *results*, en esta carpeta se cogen las variables almacenadas en los diccionarios que devuelve SHIPCal (*plotVars* y *reportsVar*, pues *template_vars* únicamente contiene valores cuando se ejecuta desde RESSSPI) y se importan las funciones generadoras de gráficos para generar los gráficos con las variables del diccionario *plotVars*. Las variables del diccionario *reportsVar* se muestran por pantalla directamente.

4.1.3.1 Modificaciones en views.

La carpeta de *views.py* se modificará en la carpeta del formulario (*simforms*) y en la de resultados (*results*).

4.1.3.1.1 Modificaciones en carpeta de simforms.

En el fragmento de código en el que se definen los diccionarios, es necesario incluir en el diccionario *SimControl* las variables nuevas que necesita SHIPCal. Para ello, según sean los valores introducidos en el formulario para *itercontrol* y la variable *annual*, las variables de inicio y final de simulación tendrán unos valores u otros:

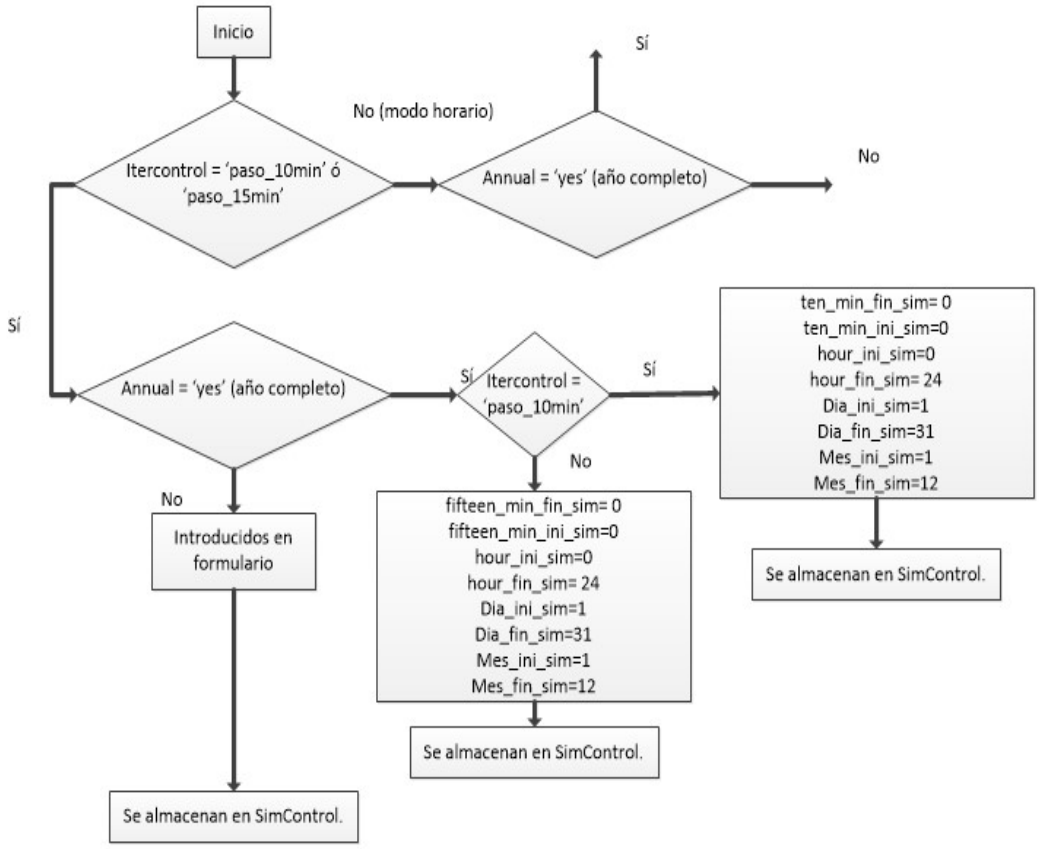

Figura 36: variables de inicio y final de simulación según *itercontrol* y *annual* (parte 1).

Si se escoge el modo de simulación de año completo, las variables de inicio y final de simulación tendrán valores por defecto. Estos valores dependerán de si se simula con paso horario, paso diezminutal o quinceminutal.

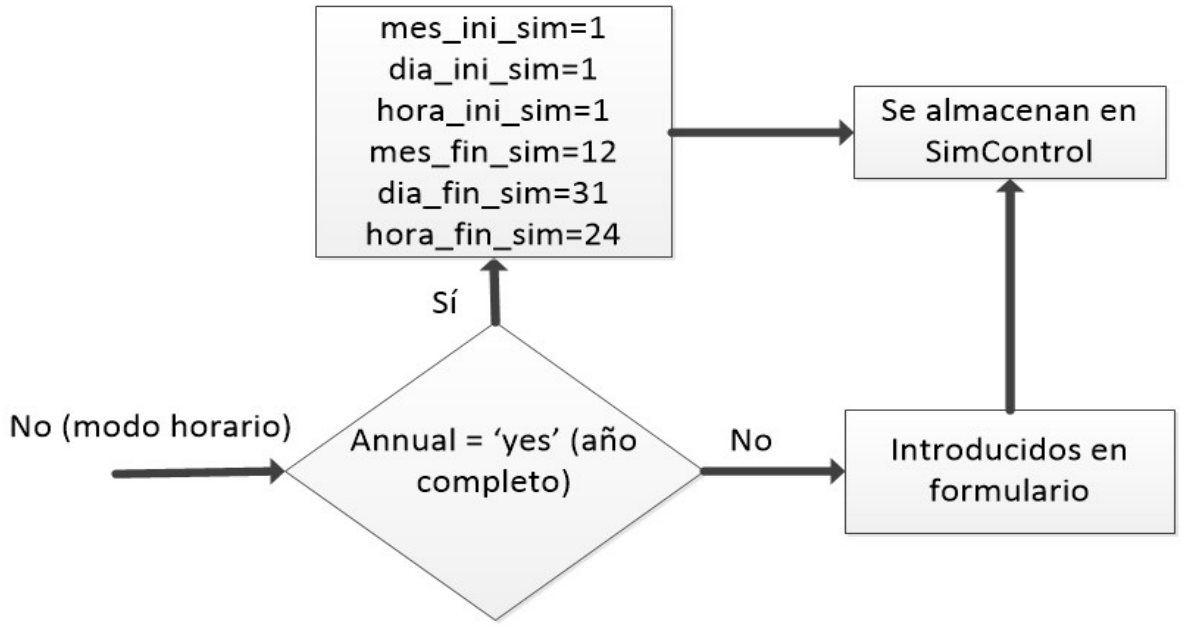

Figura 37: variables de inicio y final de simulación para modo horario (parte 2).

4.1.3.1.2 Modificaciones en carpeta de results.

El cambio más significativo es que según sea el valor de *itercontrol* y *steps_sim* (número de iteraciones que tiene la simulación), se ejecutarán unas funciones auxiliares de gráficos determinadas. Así, si *steps_sim* es 8759, 52560 o 35040, ejecutará las funciones que generan gráficas de estudio de 1 año completo: diagrama de Sankey, gráfico financiero, etc. Si *itercontrol* vale '*paso_10min'* o '*paso_15min*' ejecutará las funciones creadas en este trabajo y en caso contrario ejecutará las ya existentes para paso horario.

En el caso de simulaciones menores de 1 año completo, los gráficos que se mostrarán será el de demanda y radiación, gráfico de caudales y temperaturas y en caso de ser un modelo de integración con almacenamiento, también mostrará el gráfico de estado del almacenamiento.

Independientemente de si se simula durante todo el año o durante un tiempo menor, siempre se mostrarán las gráficas de propiedades del fluido de trabajo.

4.1.4 Carpetas de HTML.

HTML, siglas en inglés de HyperText Markup Language ('lenguaje de marcas de hipertexto'), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. [16]

Así, según se disponga la estructura dentro del código de estas carpetas, se mostrará por pantalla una disposición determinada del front-end. Por tanto, es estas carpetas dónde se organiza lo que el usuario verá por pantalla, tanto en el formulario, como en los resultados.

4.1.4.1 HTML para formularios.

Estas carpetas de HTML son tres:

- Carpeta *simulation_form.html*: aquí se construye la visual de la interfaz del formulario.
- Carpeta de *new_locations.html*: esta carpeta contiene el código que organiza el formulario para añadir una nueva localización y un nuevo TMY.
- Carpeta de *new_fuel.html*: esta carpeta contiene el código que estructura el formulario para añadir un nuevo combustible.

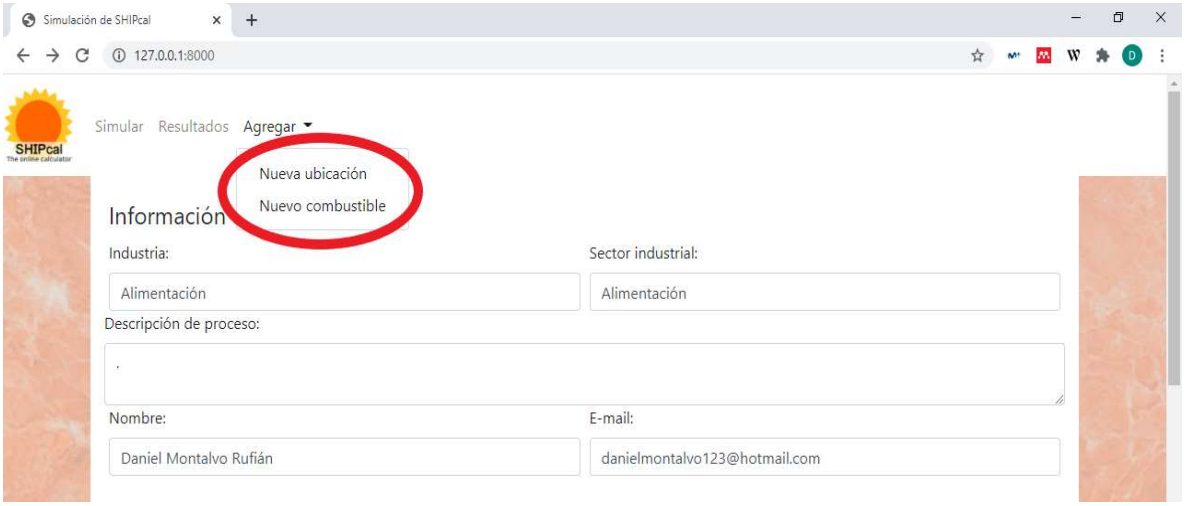

Figura 38: cabecera del formulario principal (*simulation_form.html*).

Como se observa en la ilustración, por pantalla se observa la cabecera del formulario definido y estructurado en *simulation_form.html.* La zona resaltada indica los otros dos formularios: nueva ubicación (*new_locations.html)* y nuevo combustible (*new_fuel.html).*

4.1.4.1.1 Modificaciones en los HTML de formularios.

En el formulario principal es necesario añadir las variables *to_solartime, huso, itercontrol, annual.* Dado que inicialmente el front-end solo permitía realizar simulaciones anuales, se debe añadir *month_ini_sim, month_fin_sim, day_ini_sim, day_fin_sim, hour_ini_sim* y *hour_fin_sim* para que las introduzca el usuario.

4.1.4.2 HTML para resultados.

Como se ha mencionado, el front-end inicialmente estaba habilitado únicamente para simulaciones anuales, por lo que originalmente los resultados solo corresponden a gráficos y resultados que se obtienen si se simula durante un año completo (como se mencionó en "Bloque 4: generación de gráficos" y "Bloque 5: generación de informes"):

- Carpeta *imp.html*: esta carpeta contiene la estructura de la primera ventana de resultados que se muestra tras simular, que es la ventana "resumen". Esta ventana muestra el resumen financiero y el resumen de producción.

- Carpeta *imp_instalation.html*: esta carpeta contiene la estructura que da forma a la ventana de "instalación", que muestra el resumen de los parámetros de operación de la planta: colectores por lazo y número de lazos, área utilizada de captación, tipo de colector, esquema de integración seleccionado y caudal promedio.

- Carpeta *imp_prod.html:* esta carpeta contiene la disposición de la ventana de "producción solar". Aquí se muestra la producción a lo largo de todo el año, en la primera semana de enero y de verano. También se muestran las propiedades promedio del fluido de trabajo en el campo solar.

- Carpeta *imp_finance.html*: se muestra el gráfico financiero de flujos de caja y la progresión de costes y ahorros durante el número de años de vida útil de la planta.

Figura 39: cabecera de la ventana de resultados financieros (*imp_finance.html*).

Como se observa en la figura, en la cabecera se encuentran las 4 ventanas de resultados: resumen (*imp.html*), instalación solar (*imp_instalacion.html*), producción solar (*imp_prod.html*) y resumen financiero (*imp_finace.html*).

4.1.4.2.1 Modificaciones en los HTML de resultados.

Cómo inicialmente estaba habilitada solamente para simulaciones de 1 año completo de paso horario, hay que introducir la opción de mostrar los gráficos de simulaciones diezminutales o quinceminutales si *itercontrol* así lo indica. También, en caso de que las simulaciones sean menores de un año completo, no se mostrarán las ventanas descritas en el apartado anterior, sino que se mostrarán únicamente dos ventanas: ventana de resultados y ventana de propiedades. La ventana de resultados también usará la carpeta de *imp.html*, que mostrará el gráfico de producción y demanda de cada paso de la simulación y el gráfico de temperaturas y caudales en cada paso de simulación. La ventana de propiedades usará *imp_prod.html,* y mostrará los gráficos de propiedades.

5 EJEMPLO DE SIMULACIÓN.

En este apartado se reflejarán varias simulaciones de ejemplo, en las que se compara el modo horario con el modo diezminutal. Como se ha mencionado, el TMY que se usará para estas simulaciones no está en formato de hora solar, por lo que hay que calcularla. También se ha mencionado que la franja horaria de dicho TMY es UTC. El TMY será de Sevilla. Para todas las simulaciones que se reflejen aquí, ciertos parámetros permanecerán fijos:

Parámetros variables.	**El usuario los puede cambiar en cada simulación**
$to\_solartime = 'on'$	Se calculará la hora solar.
$huso = 0$	El TMY de estas simulaciones está en formato UTC.
$mofINV = 1$	No se modifican inversión, DNI o producción en las simulaciones, por lo que los parámetros de ajuste de simulación valdrán 1.
$mofDNI = 1$	
$mofPROD = 1$	
num_loops (Nº de lazos) $= 3$	Parámetros de dimensionado de planta.
n_coll_loop ($Nº$ $colectores$ por $lazo$) $= 20$	
$almVolumen = 8.000\ L$	
$BusinessModel = "turnkey"$	El análisis financiero se hará con el modelo de "llave en mano"
$Fuel = "NG"$	El combustible de la caldera será gas natural
$dayarray = [0,0,0,0,0,0,0,0,0,\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},\frac{1}{10},0,0,0,0,0,0]$	La demanda será desde las 9.00 hasta las 18.00, todos los días del año. Resultan 3650 horas al año.
$T\_process\_in = 115°C$	Temperatura de entrada al proceso.
$T\_process\_out = 45°C$	Temperatura de salida del proceso.
$P\_op = 15\ bar$	Presión del fluido.
Subbloque 1.2	**Variables fijas del subbloque 1.2. Sólo se pueden cambiar desde el código fuente.**
$CPI = 2,5$	IPC, para calcular el incremento de costes.
$FuelCostRaise = 3,5$	Incremento del coste del combustible.
$n\_years\_sim = 25$	Número de años de funcionamiento de la instalación
$lim\_inf\_DNI = 200\ [\frac{W}{m^2}]$	Irradiancia mínima para que la instalación solar funcione.
$m\_dot\_min\_kgs = 0,06\ [\frac{kg}{s}]$	Caudal mínimo de la bomba.
$coef\_flow\_rec = 1$	Coeficiente de recirculación.
$Boiler\_eff = 0,8$	Rendimiento de la caldera.
$heatFactor = 0,8$	Porcentaje de precalentamiento respecto al salto de temperaturas de entrada y salida del proceso, en caso de haber intercambiador.
$HX\_eff = 0,9$	Eficiencia del intercambiador
$DELTA\_ST = 30$	Diferencia de temperatura sobre la temperatura de diseño del almacenamiento, en caso de haberlo.

$\boldsymbol{DELTA\_HX = 5}$	Diferencia de temperatura entre las corrientes del intercambiador, en caso de haberlo.
Colector	**Variables del colector estándar de SHIPCal.**
Tubo receptor	SCHOTT
Área de captación de cada colector	$26{,}4\ m^2$
Rendimiento óptico pico	0,75583
Longitud de cada colector	$5{,}28\ m$

Tabla 6: variables fijadas para las simulaciones.

5.1 Aspectos que se deben considerar.

Para comparar los resultados, primero se analizará los resultados horarios y diezminutales en horas concretas, para cuantificar la diferencia entre ambos modos de simulación. Como se ha explicado en las funciones de producción, la energía que se produce en cada tubo receptor depende principalmente de 2 parámetros: irradiancia y ángulo de incidencia longitudinal. La variación de ambos hará que en ocasiones la producción solar no sea la misma en ambos modos de simulación. Por último, debe incluirse el efecto de la recirculación. En caso de haber recirculación por caudal demasiado pequeño, las producciones no serán iguales.

5.1.1 Irradiancia estable y sin recirculación.

Si se simula durante una hora en la que la irradiancia varía lentamente (mediodía, primeras horas de la tarde y media mañana) o no sufre cambios acentuados, ambos modos tendrán resultados similares, aunque generalmente se produce un poco más de energía bruta en el modo horario. La producción neta (útil que se consume y no se desecha), será similar en ambos casos. Se mostrará dos ejemplos, uno un día de verano donde la irradiancia es muy similar durante toda la hora. En el otro ejemplo, una hora con variación moderada. En ambos, la demanda de energía para el proceso será 800 kWh.

- Irradiancia estable:

$\boldsymbol{DNI - TMY\ horario\ , \theta_{long}}$	$\boldsymbol{DNI - TMY.10min\ , \theta_{long}}$
$719\ \frac{\mathbf{W}}{\mathbf{m^2}}/\ 13{,}96°$	$712{,}1\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 12{,}95°$
	$716{,}1\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 13{,}34°$
	$716{,}8\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 13{,}86°$
	$718{,}9\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 14{,}23°$
	$723{,}1\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 14{,}54°$
	$726{,}5\ \frac{\mathrm{W}}{\mathrm{m^2}}/\ 14{,}78°$

Tabla 7: irradiancia y ángulos de incidencia longitudinal.

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Pérdidas (kWh)*	*Producción/demanda*
					producción neta/ demanda
Modo horario	*SL_L_RF*	728	640	9,77	91 %
					80 %
Modo diezminutal	*SL_L_RF*	727,415	640	9,764	90,92 %
					80 %

Tabla 8: resultados para una hora con irradiancia estable.

- Variación moderada de la irradiancia.

$DNI - TMY\ horario\ , \theta_{long}$	$DNI - TMY.10min\ , \theta_{long}$
$380\ \frac{W}{m^2}/\ 42{,}66°$	$312{,}9\ \frac{W}{m^2}/\ 43{,}12°$
	$405{,}7\ \frac{W}{m^2}/\ 43{,}19°$
	$429{,}7\ \frac{W}{m^2}/\ 43{,}17°$
	$370{,}4\ \frac{W}{m^2}/\ 43{,}06°$
	$382{,}8\ \frac{W}{m^2}/\ 42{,}86°$
	$379{,}9\ \frac{W}{m^2}/\ 42{,}58°$

Tabla 9: irradiancia y ángulos de incidencia longitudinal.

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Pérdidas (kWh)*	*Producción/demanda*
					Producción neta/ demanda
Modo horario	*SL_S_FW*	280,34	206,340	25,752	35,04%
					25,79 %
Modo diezminutal	*SL_S_FW*	278,128	206,340	25,717	34,77 %
					25,79 %

Tabla 10: resultados para una hora con irradiancia estable.

5.1.2 Irradiancia inestable y sin recirculación.

En este caso, la irradiancia es inestable o presenta valores bajos (primeras horas del amanecer y últimas horas del atardecer). Para esta situación se puede presentar el problema de que la irradiancia para una hora sea menor que el mínimo impuesto (se ha impuesto 200 $\frac{W}{m^2}$) para realizar la producción en el campo solar. También puede darse el caso de que, para el modo horario, la irradiancia sí sea mayor, pero alguna porción diezminutal en el TMY diezminutal tenga irradiancia menor que el mínimo. La demanda por parte del proceso será de 800 kWh.

5.1.2.1 Irradiancia insuficiente en modo horario.

La irradiancia para una hora determinada en el TMY horario es menor que la mínima y no se producirá o consumirá recurso solar, por estar la planta solar parada. Sin embargo, en las porciones minutales del TMY diezminutal sí se produce y consume. En este caso, la producción en el modo diezminutal será mayor que el horario, que será 0. En el gráfico siguiente se observa que en la tercera hora (punto 3) la irradiancia es un poco menor de 200 $\frac{W}{m^2}$ y la instalación solar entra en modo parada.

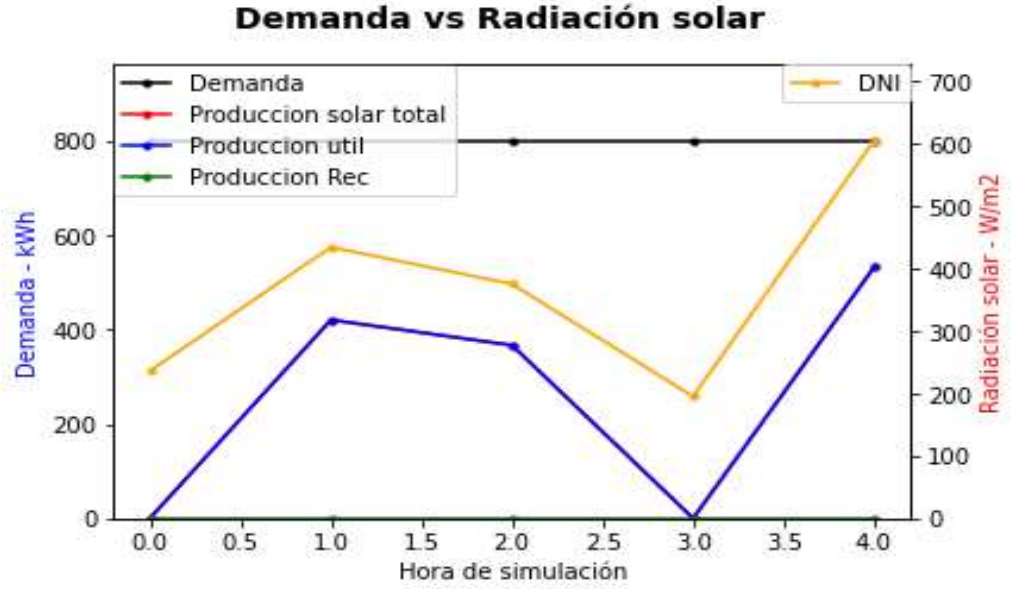

Figura 40: demanda y producción con irradiancia insuficiente en modo horario.

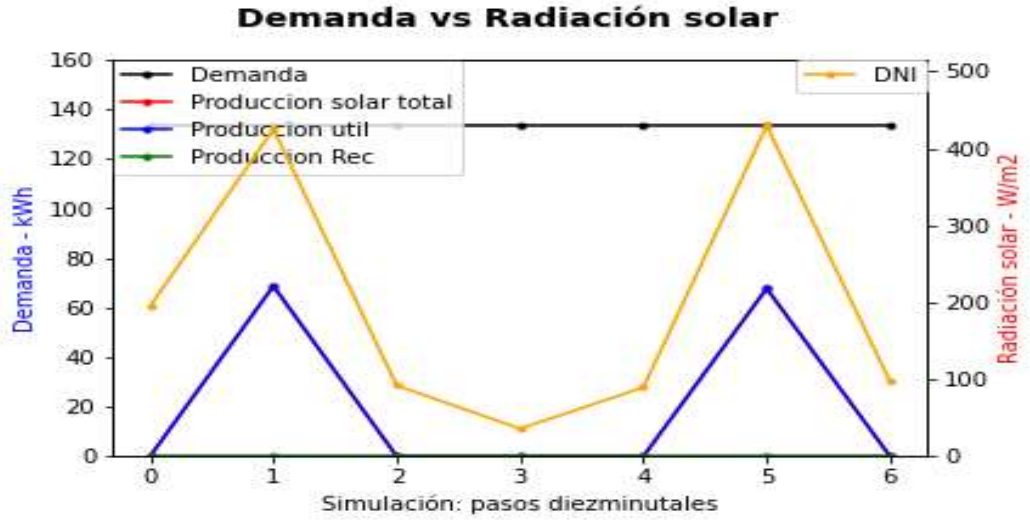

Figura 41: demanda, producción e irradiancia en modo diezminutal.

En la gráfica anterior, desde el punto 1 al punto 6 se representa el punto 3 (de 14.00 a 15.00 del mediodía) de la primera gráfica, viéndose que en el modo diezminutal no sufre parada durante toda la hora, sólo en algunas porciones. Se recuerda que los primeros diez minutos (para simulaciones diezminutales) o la primera hora (para simulaciones horarias) se consideran con producción nula (punto 0, de 13.50 a 14.00 de la tarde). Se observa como en los primeros 10 minutos de la hora estudiada (punto 1) y en la porción 40-50 minutos (punto 5) de dicha hora sí habrá producción (línea roja tapada por línea azul, pues son coincidentes) y consumo (línea azul).

5.1.2.2 Irradiancia insuficiente en modo diezminutal.

En este supuesto, sucede el fenómeno contrario al apartado anterior. Aquí la irradiancia será mayor que el mínimo en el modo horario, pero en el modo diezminutal habrá franjas que no tengan producción. Como resultado, la producción en el modo horario será mayor que en el modo diezminutal.

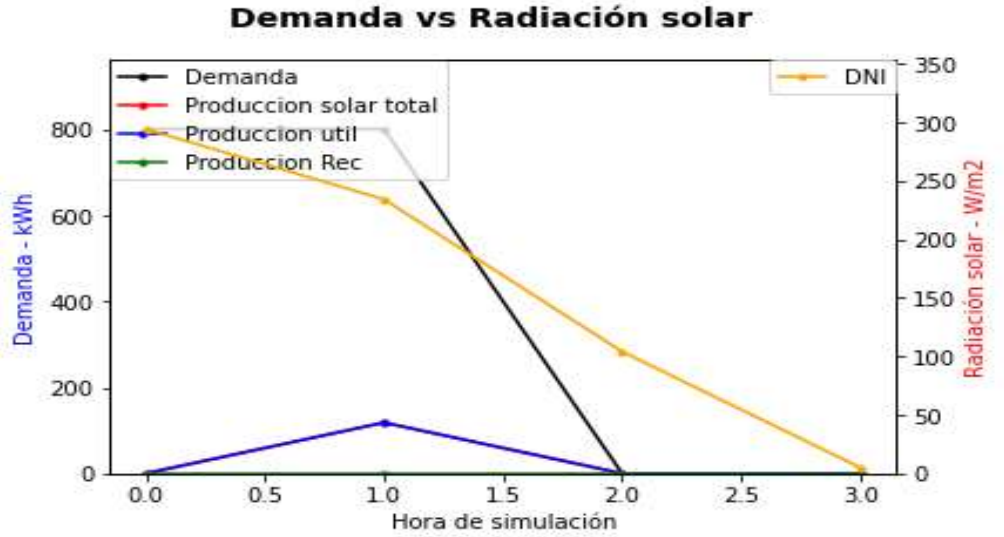

Figura 42: demanda, producción e irradiancia en modo horario.

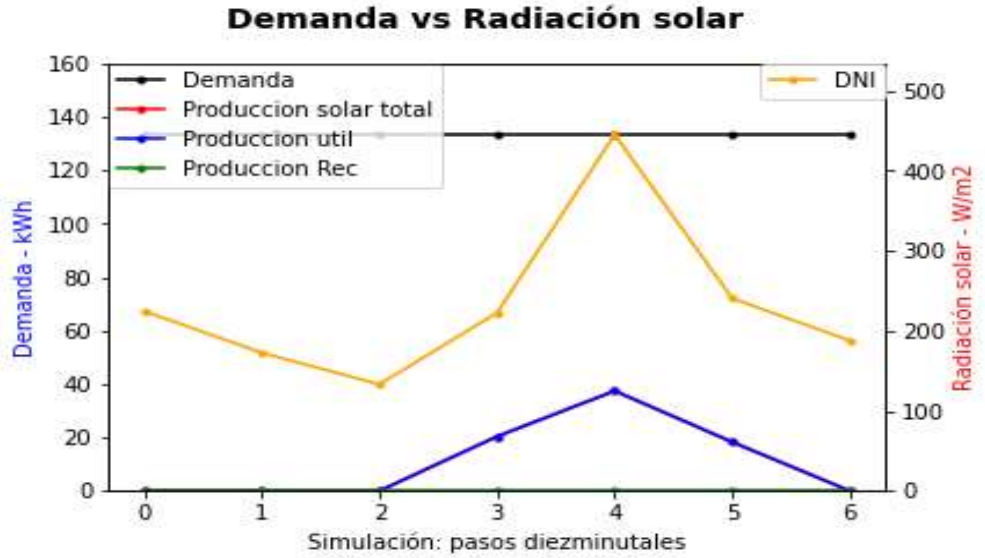

Figura 43: demanda y producción con irradiancia insuficiente en modo diezminutal.

En el punto 1 de la gráfica de modo horario, la irradiancia es baja, pero suficiente para entrar en modo producción. En la segunda gráfica se muestran los 60 minutos de dicha hora (punto 1 a 6). Se observa que los puntos 1, 2 y 6 tienen irradiancia insuficiente, por tanto, habrá 30 minutos en los que el campo esté parado si se simula en modo diezminutal.

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Pérdidas (kWh)*	*Producción/demanda*
					Producción neta/demanda

Modo horario	*SL_L_PS*	129,664	129,664	5,54	16,21%
					16,21 %
Modo diezminutal	*SL_L_PS*	83,799	83,799	3,323	10,50 %
					10,50 %

Tabla 11: resultados para una hora con insuficiente irradiancia para modo diezminutal.

5.1.3 Disminución debido a recirculación.

En caso de que haya recirculación en alguna porción en modo diezminutal, pero no en el modo horario, la producción será mayor en el modo horario. Sin embargo, el fenómeno de recirculación afecta más negativamente al modo horario que al modo diezminutal. Entonces, si en una hora determinada hay recirculación en el modo horario y en algunas porciones en el modo diezminutal, la producción generalmente será mayor en el modo diezminutal.

Modo de simulación.	***Modelo de integración***	***Producción (kWh)***	***Producción neta(consumo) (kWh)***	***Pérdidas (kWh)***	***Producción/demanda***
					Producción neta/demanda
Modo horario	*SL_L_P*	21,666	0	10,988	2,7%
					0 %
Modo diezminutal	*SL_L_P*	50,800	50,800	11,013	6,35 %
					6,35 %

Tabla 12: variación de resultados en una hora con recirculación.

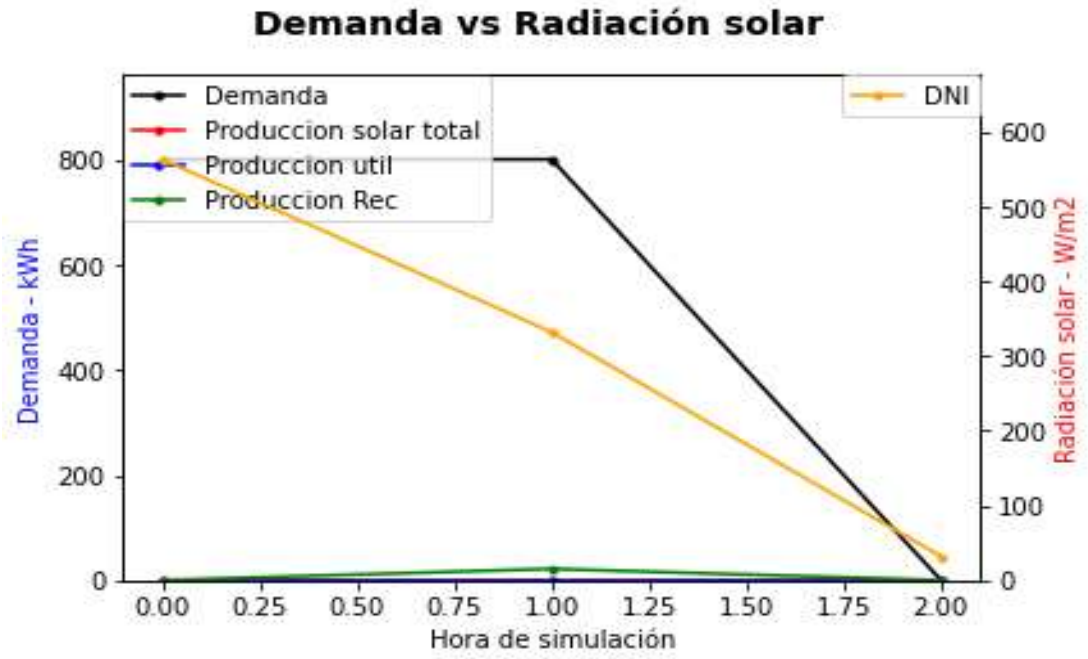

Figura 44: recirculación (verde) en modo horario.

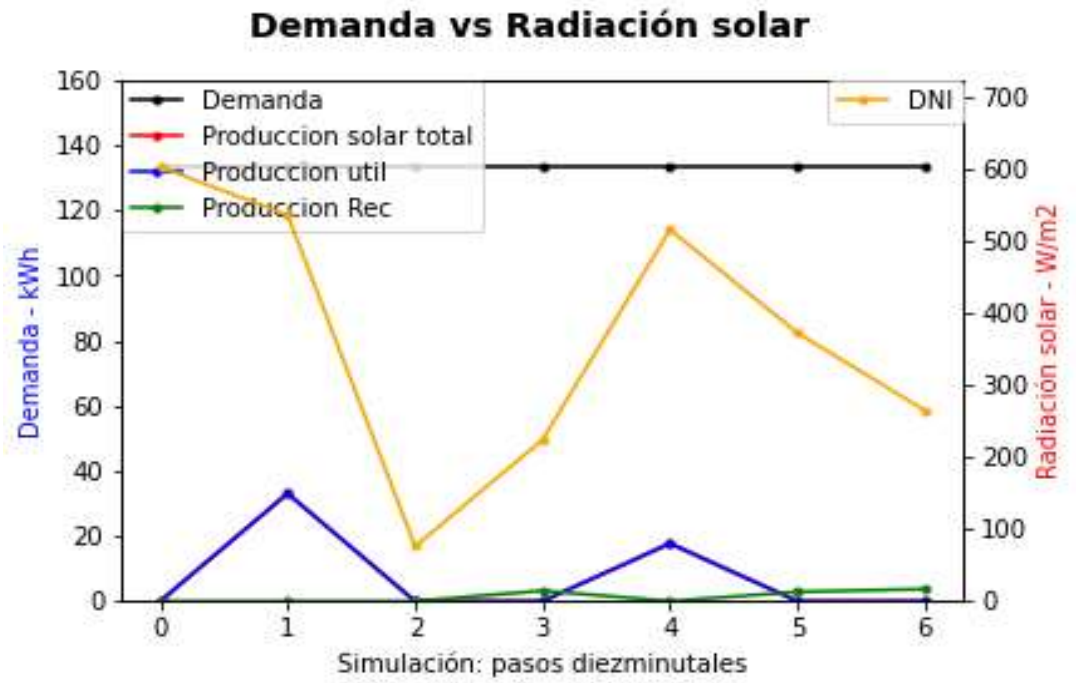

Figura 45: recirculación (verde) en modo diezminutal.

En las gráficas se observa el fenómeno de recirculación. En la primera gráfica, el punto 1 es una hora en la que es necesario realizar recirculación y dado que no hay almacenamiento, no se puede responder a la demanda. En la segunda gráfica se representa desde el punto 1 al 6 la hora correspondiente al punto 1 de la primera gráfica. Como se observa, la recirculación (verde) conlleva menos energía producida que si se estuviera en estado de producción normal. Dado que hay franjas de 10 minutos que sí producen, en general, la producción y el consumo para suplir la demanda en modo diezminutal será mayor que la producción en modo horario.

5.2 Simulaciones comparativas.

A continuación, tras haber explicado los motivos que pueden hacer diferir ambos modos de simulación, se procede a realizar 4 comparaciones:

- Día con irradiancia estable (verano).
- Día con irradiancia variable (primavera).
- Meses del año.
- Simulación anual.

Estas simulaciones son comparativas entre ambos modos de simulación y no pretenden demostrar la viabilidad de cada modelo de integración. Habrá integraciones con peores prestaciones respecto a otras por las variables de operación que se han fijado para todas en común. En todas estas comparaciones, la demanda de energía por parte del proceso se establece como 600 kWh cada hora.

5.2.1 Día con irradiancia estable.

Se simulará el 6 de junio, en el que la irradiancia no sufra variaciones bruscas. Por tanto, el único efecto que hará diferir ambos modos de simulación serán posibles recirculaciones y valores de irradiancia por debajo del mínimo al amanecer y en las últimas horas del atardecer. Se recuerda que las integraciones con almacenamiento acumulan energía sobrante de los puntos con mayor producción, complementando la energía que falte para cubrir la demanda en puntos con menos producción.

Dado que hay 10 horas de demanda al día, la demanda total resulta 6.000 kWh en total en las horas en las que hay demanda (también impuestas en la tabla 6). En todas las integraciones de la IEA la curva roja (producción) y curva amarilla (irradiancia) serán estables y sin variaciones bruscas, tal como se muestra en las imágenes.

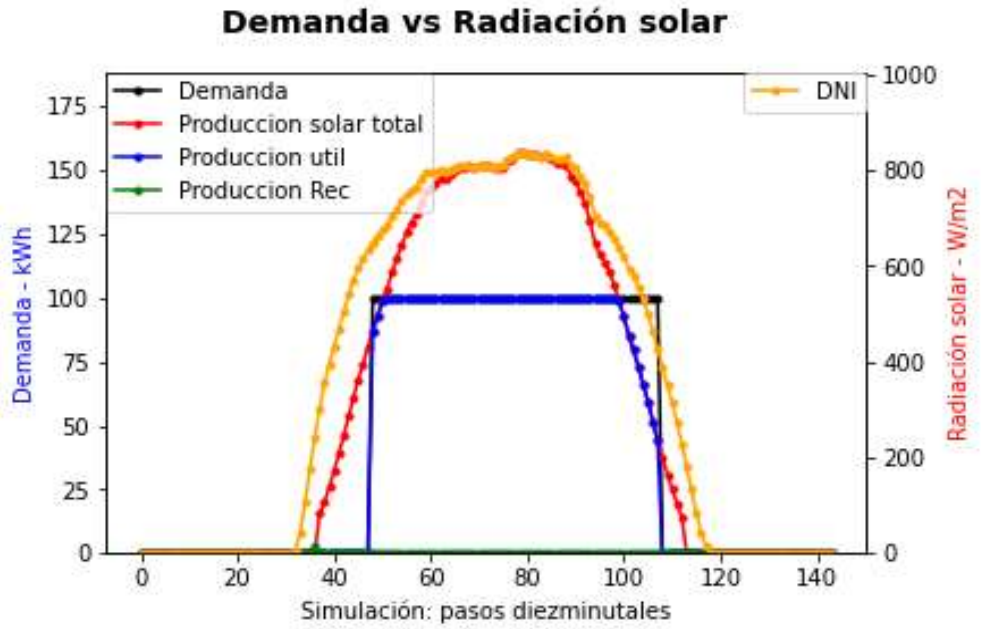

Figura 46: ilustración de ejemplo. Simulación diezminutal (SL_L_P).

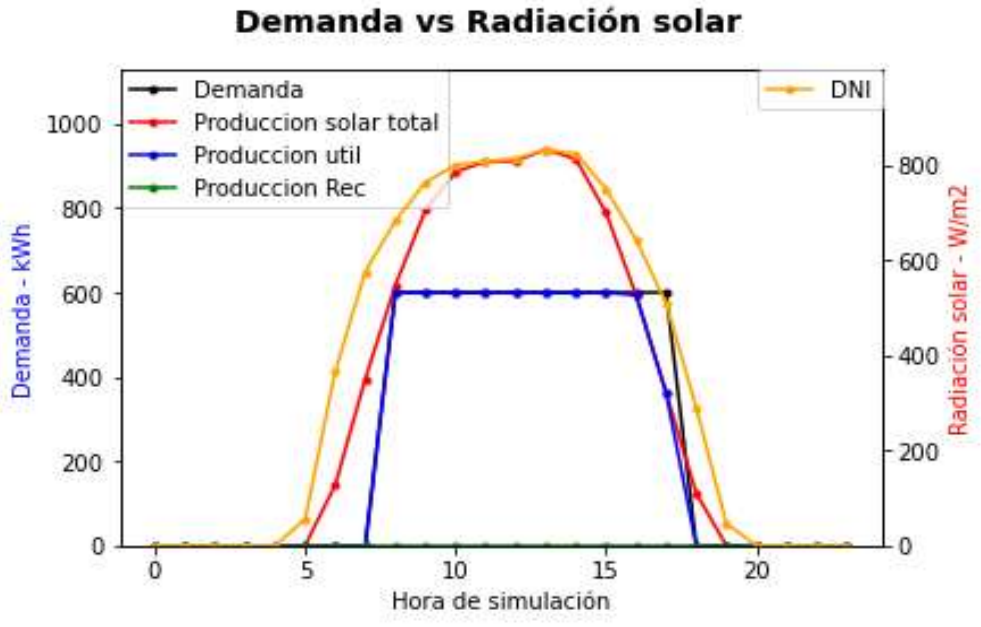

Figura 47: ilustración de ejemplo. Simulación horaria (SL_L_P).

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Pérdidas térmicas (kWh)*	*Producción/demanda*
					Producción neta/ demanda
Modo horario	*SL_L_P* (aceite)	8.382,716	5.755,286	142,029	139,70 %
					95,92 %
Modo diezminutal	*SL_L_P(aceite)*	8.367,947	5.729,691	141,514	139,47 %
					95,50 %
Modo horario	*SL_L_PS (aceite) (almacenamiento)*	8.374,344	6.000,000	142,029	139,57 %
					100,00 %
Modo diezminutal	*SL_L_PS (aceite) (almacenamiento)*	8.366,775	6.000,000	141,549	139,45 %
					100,00 %
Modo horario	*SL_L_RF (aceite)*	7.557,980	4.646,801	118,617	125,97 %
					77,45 %
Modo diezminutal	*SL_L_RF (aceite)*	7.550,960	4.653,604	118,366	125,58 %
					77,56 %
Modo horario	*SL_L_S (aceite) (almacenamiento)*	6.379,996	4.107,286	142,107	106,33 %
					68,46 %
Modo diezminutal	SL_*L_S (aceite) (almacenamiento)*	6.394,673	4.406,664	142,107	106,58 %
					73,44 %
Modo horario	*SL_L_S_PH (aceite) (almacenamiento)*	5.796,697	4.383,006	46,87	96,61 %
					73,05 %
Modo diezminutal	*SL_L_S_PH (aceite) (almacenamiento)*	6742,111	5300,000	87,1	112,37 %
					88,33 %
Modo horario	*SL_S_FW (agua)*	8.058,006	1.511,828	395,56	134,30 %
					25,20 %
Modo diezminutal	*SL_S_FW (agua)*	8.128,169	1.511,828	390,097	135,47 %
					25,20 %
Modo horario	*SL_S_FWS (agua) (almacenamiento)*	8.044,356	1.511,829	413,649	134,07 %
					25,20 %
Modo diezminutal	*SL_S_FWS (agua) (almacenamiento)*	8.119,015	1.511,829	409,276	135,32 %
					25,20 %
Modo horario	*SL_S_PD_OT (vapor)*	7.651,896	5.367,017	416,345	127,32 %
					89,45 %
Modo diezminutal	*SL_S_PD_OT (vapor)*	7.658,029	5.546,377	409,979	127,63 %
					92,44 %
Modo horario	*PL_E_PM (vapor)*	8.382,716	5.755,286	142,029	139,71 %
					95,92 %

Modo diezminutal	*PL_E_PM (vapor)*	8.367,947	5.729,69	141,514	139,47 %
					95,50 %
Modo horario	*SL_S_MW (agua)*	8.064,456	1.511,828	392,117	134,41 %
					25,20 %
Modo diezminutal	*SL_S_MW (agua)*	8.141,990	1.511,828	389,413	135,69 %
					25,20 %
Modo horario	*SL_S_MWS (agua) (almacenamiento)*	8.052,127	1.637,386	408,842	134,20 %
					27,28 %
Modo diezminutal	*SL_S_MWS (agua) (almacenamiento)*	8.128,171	1.637,376	407,914	135,47 %
					27,29 %
Modo horario	*SL_S_PD (vapor)*	8.100,029	5.456,051	-	135,00 %
					90,93 %
Modo diezminutal	*SL_S_PD (vapor)*	8.095,237	5.124,344	-	134,92 %
					85,406 %
Modo horario	*SL_S_PDS (vapor) (almacenamiento)*	7.993,959	6.000,000	413,649	133,23 %
					100,00%
Modo diezminutal	*SL_S_PDS (vapor) (almacenamiento)*	7.907,38	6.000,000	409,979	131,79 %
					100,00%

Tabla 13: resultados para un día con irradiancia estable.

- La variación de SL_L_S_PH se debe a que hay iteraciones en las que el fluido almacenado en el acumulador ya tiene la temperatura necesaria para enviarlo a la caldera (PH: precalentamiento). Esto provoca que haya iteraciones en los que, aun habiendo condiciones climáticas para producir, el campo esté parado por no necesitarse un salto de temperaturas en el fluido. Los dos modos de simulación progresarán de manera diferente:

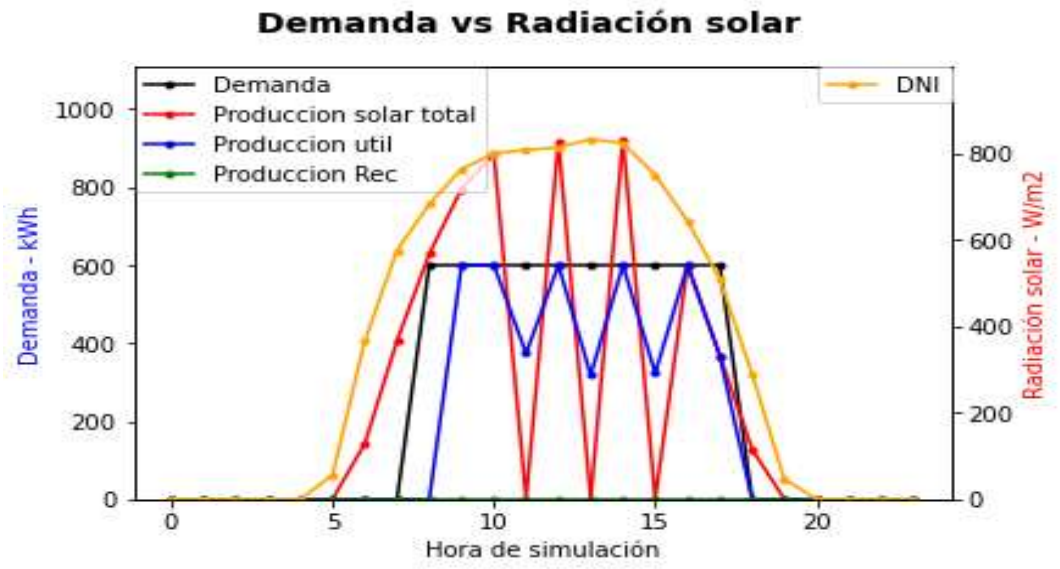

Figura 48: simulación del 6 de junio de SL_L_S_PH, modo horario.

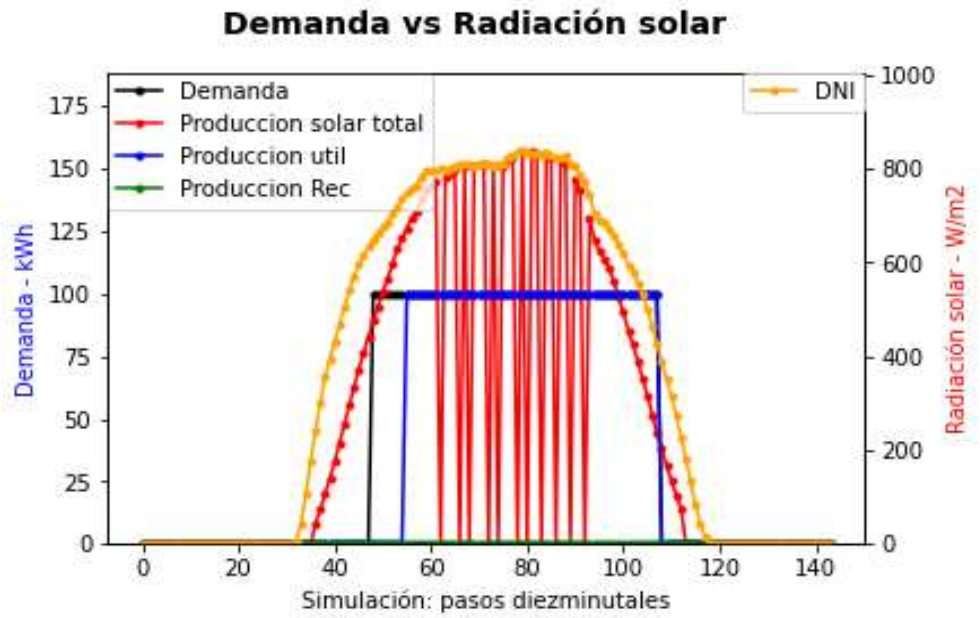

Figura 49: simulación del 6 de junio de SL_L_S_PH, modo diezminutal.

- El modelo SL_S_PD aún no tiene implementado el cálculo de pérdidas.

En general, los resultados son similares y en algunas integraciones se consume más energía neta en el modo diezminutal y en otras más en el modo horario. Por tanto, no se observa cual modo de simulación resulta más favorable para las integraciones.

Aparentemente, las integraciones que utilizan líquido térmico (aceite) y recogen el fluido del proceso para después suministrarlo en paralelo a la caldera tienen mayor producción bruta en modo horario. Éstas son SL_L_P, SL_L_PS y SL_L_RF. Ocurre lo mismo con las integraciones en paralelo que trabajan con vapor.

Sin embargo, las integraciones que funcionan con agua, que son SL_S_FW, SL_S_FWS, SL_S_PD_OT y SL_S_MW tienen mayor producción bruta en el modo diezminutal. Este fenómeno también se da en las dos integraciones que utilizan líquido térmico, pero que no recogen fluido del proceso para después llevarlo directamente tras salir del campo solar, sino que lo suministran siempre desde un acumulador: SL_L_S y SL_L_S_PH.

La integración PL_E_PM sirve para calentar un proceso terciario con intercambiador intermedio y, aunque se ha simulado con vapor, no tiene un fluido de trabajo establecido. Por tanto, esta integración se puede simular con cualquier fluido. Al igual que en la tabla, si se simula con aceite sigue resultando mejor el modo horario que el modo diezminutal pues la producción bruta y consumo en modo horario son 8.271,87 kWh y 5.739,45 kWh respectivamente, mientras que en el modo diezminutal son 8.265,321 kWh y 5.713,681 kWh. Esto también se ha comprobado con las integraciones que trabajan con líquido térmico y trabajan en paralelo (SL_L_P, SL_L_PS y SL_L_RF), debido a que en lugar de simular con aceite se ha simulado con sales fundidas y el modo horario sigue teniendo mayor producción bruta.

Respecto a qué modo de integración varía más, el SL_L_S_PH es el que más varía de manera notable tanto en producción bruta como en producción neta, por el motivo ya expuesto anteriormente. Además de esta integración, las integraciones que varían en más porcentaje respecto al modo diezminutal ($\frac{horar}{diezminutal}$) en producción neta son SL_L_S (93,2%), SL_S_PD_OT (96,8%) y SL_S_PD (106,5%).

5.2.2 Día con irradiancia inestable.

En este caso, se simulará en un día de primavera (13 de abril), con irradiancia más variable. Esto provocará que las prestaciones disten más entre el modo horario y el diezminutal, en comparación con el caso de un día con irradiancia estable. La demanda será igual que en el caso de irradiancia estable. La irradiancia (amarillo) será como en la gráfica:

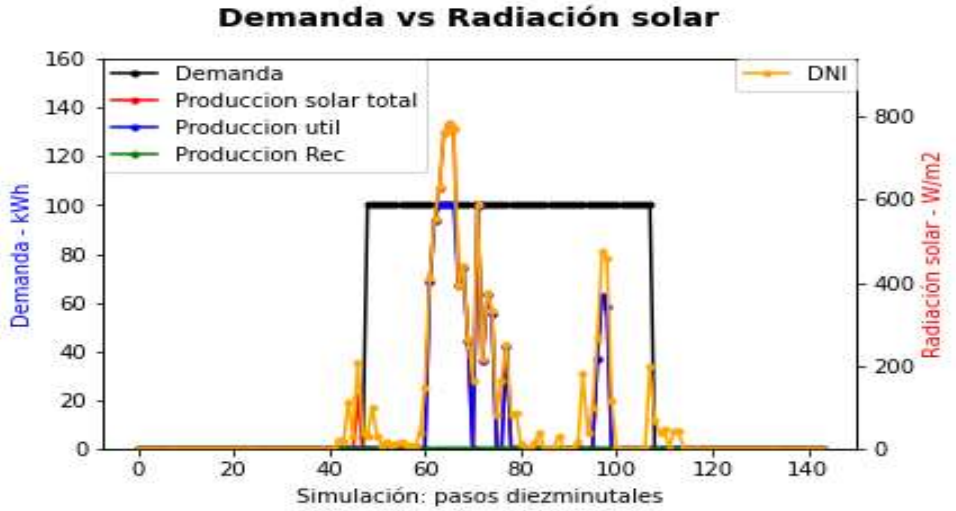

Figura 50: simulación diezminutal. Ilustración con SL_L_P de ejemplo.

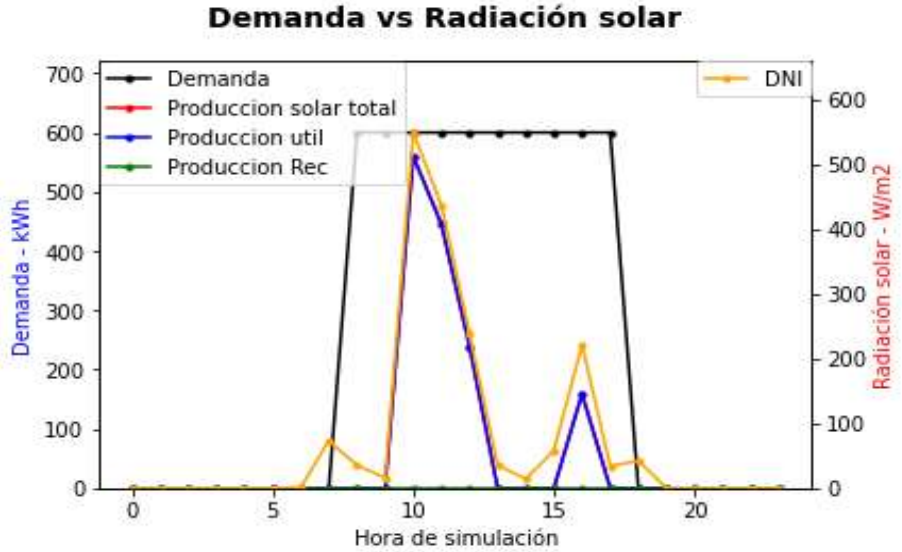

Figura 51: simulación horaria. Ilustración con SL_L_P de ejemplo.

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Pérdidas térmicas (kWh)*	*Producción/demanda*
					Producción neta/ demanda
Modo horario	*SL_L_P* (aceite)	1.401,383	1.401,383	34,691	23,36 %
					23,36 %
Modo diezminutal	*SL_L_P(aceite)*	1.324,638	1.202,507	30,525	22,08 %
					20,04 %

Modo horario	*SL_L_PS (aceite) (almacenamiento)*	1.401,383	1.401,383	34,691	23,356 % 23,356 %
Modo diezminutal	*SL_L_PS (aceite) (almacenamiento)*	1.324,638	1.324,638	30,525	22,08 % 22,08 %
Modo horario	*SL_L_RF (aceite)*	1.267,291	1.242,480	27,973	21,12% 20,71 %
Modo diezminutal	*SL_L_RF (aceite)*	1.196,913	1.031,669	25,259	19,95 % 17,19 %
Modo horario	*SL_L_S (aceite) (almacenamiento)*	1.392,563	918,36	43,51	23,21 % 15,31 %
Modo diezminutal	*SL_L_S (aceite) (almacenamiento)*	1.290,929	917,238	41,780	21,52 % 15,29 %
Modo horario	*SL_L_S_PH (aceite) (almacenamiento)*	1.426,501	228,548	9,57	23,78 % 3,81 %
Modo diezminutal	*SL_L_S_PH (aceite) (almacenamiento)*	1.351,103	131,903	4,060	22,52 % 2,18 %
Modo horario	*SL_S_FW (agua)*	1.335,914	595,715	107,780	22,27 % 9,93 %
Modo diezminutal	*SL_S_FW (agua)*	1.256,528	428,351	87,386	20,94 % 7,14 %
Modo horario	*SL_S_FWS (agua) (almacenamiento)*	1.331,093	1.209,462	113,322	22,18% 20,16 %
Modo diezminutal	*SL_S_FWS (agua) (almacenamiento)*	1.253,127	1.184,265	91,474	20,89 % 19,74 %
Modo horario	*SL_S_PD_OT (vapor)*	962,434	962,434	113,322	16,04 % 16,04 %
Modo diezminutal	*SL_S_PD_OT (vapor)*	894,473	807,912	91,474	14,91 % 13,47 %
Modo horario	*PL_E_PM (vapor)*	1.402,784	1.402,784	34,691	23,38 % 23,38 %
Modo diezminutal	*PL_E_PM (vapor)*	1.325,962	1.203,310	30,525	22,10 % 20,10 %
Modo horario	*SL_S_MW (agua)*	1.337,762	595,912	107,780	22,30 % 9,93 %

Modo diezminutal	*SL_S_MW (agua)*	1.258,268	428,35	86,698	20,97 %
					7,14 %
Modo horario	*SL_S_MWS (agua) (almacenamiento)*	1.332,980	1.332,980	113,322	22,22 %
					22,22 %
Modo diezminutal	*SL_S_MWS (agua) (almacenamiento)*	1.254,904	1.254,904	90,785	20,92 %
					20,92 %
Modo horario	*SL_S_PD (vapor)*	1.322,753	203,884	-	22,05 %
					3,40 %
Modo diezminutal	*SL_S_PD (vapor)*	1.263,307	0	-	21,10 %
					0 %
Modo horario	*SL_S_PDS (vapor) (almacenamiento)*	962,434	962,434	113,322	16,04 %
					16,04 %
Modo diezminutal	*SL_S_PDS (vapor) (almacenamiento)*	1.109,553	1.109,553	91,473	18,49 %
					18,49 %

Tabla 14: resultados para un día con irradiancia inestable.

Como se observa en las variaciones entre los ratios de la columna de la derecha de ambos modos de simulación, se refleja que sufre más variación la producción neta que la producción bruta en general.

Al igual que en la simulación anterior, se aplica el porcentaje respecto al modo diezminutal ($\frac{horario}{diezminutal}$). Exceptuando la integración SL_S_PDS, en la que el porcentaje resulta 86,74% y hay por tanto más producción neta y producción bruta en el modo diezminutal, se produce más energía bruta (105-108%) y se consume más energía neta en el modo horario. La producción neta alcanza diferencias varias, siendo la mayor en SL_L_S_PH (173%), por el motivo que se explicó en la simulación para un día con irradiancia estable. En el resto, las integraciones cuyos ratios de producción neta varían más son: SL_L_P (3,32 puntos), SL_L_RF (3,52 puntos), SL_S_FW (2,79 puntos), SL_S_PD (3,40 puntos), SL_S_MW (2,79 puntos) y PL_E_PM (3,28 puntos). Todos ellos con mayores ratios en modo horario, lo cual lo hace a priori más rentable desde el punto de vista del ahorro energético y la viabilidad de la instalación solar ante los cambios en el régimen de operación.

Podría establecerse la preferencia de esta simulación respecto a la simulación de un día con irradiancia estable si la meteorología e irradiancia media de la ubicación en cuestión son variables, además de que al presentar mayores variaciones se situaría en un escenario que refleja mejor las prestaciones en un régimen más transitorio, respecto a la simulación con irradiancia estable. Si se visualiza desde el punto de la seguridad y la fidelidad a la realidad, el modo diezminutal podría dar unos resultados más exactos que el modo horario, aun siendo estos últimos mejores a primera vista. Sin embargo, se debe evaluar las simulaciones anuales para sacar mejores conclusiones a largo plazo para cada integración.

5.2.3 Simulación mensual.

La factura energética representa el coste que supone la demanda si únicamente se usara caldera. El ahorro supone los euros de coste evitado del combustible que no se usaría al utilizar en su lugar recurso solar. Tanto la factura energética como el ahorro por recurso solar se refieren a cifras del primer año de funcionamiento, dado que en los años posteriores el precio del combustible sufrirá inflación.

SL_L_P					
Modo de simulación.	*Mes del año*	*Producción (kWh)*	*Producción neta(consumo) (kWh)*	*Demanda (kWh)*	*Demanda · Precio combustible (Factura energética) (€)* *Ahorro solar (€)*
Modo horario	*Enero*	54.907,689	54.907,689	186.000,000	11.625,000 3.431,731
Modo diezminutal	*Enero*	54.494,030	54.494,030	186.000,000	11.625,000 3.405,877
Modo horario	*Febrero*	72.097,858	71.147,560	168.000,000	10.500,000 4.446,722
Modo diezminutal	*Febrero*	73.179,276	72.178,030	168.000,000	10.500,000 4.511,127
Modo horario	*Marzo*	109.410,012	95.087,793	186.000,000	11.625,000 5.942,987
Modo diezminutal	*Marzo*	110.157,440	94.777,778	186.000,000	11.625,000 5.923,611
Modo horario	*Abril*	140.768,581	107.641,664	180.000,000	11.250,000 6.727,604
Modo diezminutal	*Abril*	143.186,021	107.049,003	180.000,000	11.250,000 6.690,563
Modo horario	*Mayo*	195.865,557	140.762,633	186.000,000	11.625,000 8.797,665
Modo diezminutal	*Mayo*	196.651,173	137.734,850	186.000,000	11.625,000 8.608,428
Modo horario	*Junio*	217.649,297	152.621,022	180.000,000	11.250,000 9.538,814
Modo diezminutal	*Junio*	218.665.222	151.577,857	180.000,000	11.250,000 9.473,616
Modo horario	*Julio*	251.312,679	167.819,453	186.000,000	11.625,000 10.488,716

Modo diezminutal	*Julio*	252.182,053	167.725,907	186.000,000	11.625,000 10.482,869
Modo horario	*Agosto*	214.706,193	157.633,370	186.000,000	11.625,000 9.852,086
Modo diezminutal	*Agosto*	215.455,294	156.106,358	186.000,000	11.625,000 9.756,647
Modo horario	*Septiembre*	144.900,958	123.523,479	180.000,000	11.250,000 7.720,217
Modo diezminutal	*Septiembre*	144.403,333	121.749,050	180.000,000	11.250,000 7.609,316
Modo horario	*Octubre*	92.529,692	87.417,726	186.000,000	11.625,000 5.463,608
Modo diezminutal	*Octubre*	92.632,271	87.498,511	186.000,000	11.625,000 5.468,657
Modo horario	*Noviembre*	55.236,048	55.154,652	180.000,000	11.250,000 3.447,166
Modo diezminutal	*Noviembre*	55.532,852	55.324,300	180.000,000	11.250,000 3.457,769
Modo horario	*Diciembre*	42.819,332	42.819,332	186.000,000	11.625,000 2.676,208
Modo diezminutal	*Diciembre*	42.171,716	42.171,716	186.000,000	11.625,000 2.635,730

Tabla 15: resultados para cada mes del año con la integración SL_L_P

Ambos modos de simulación tendrán unas gráficas similares, tal que así:

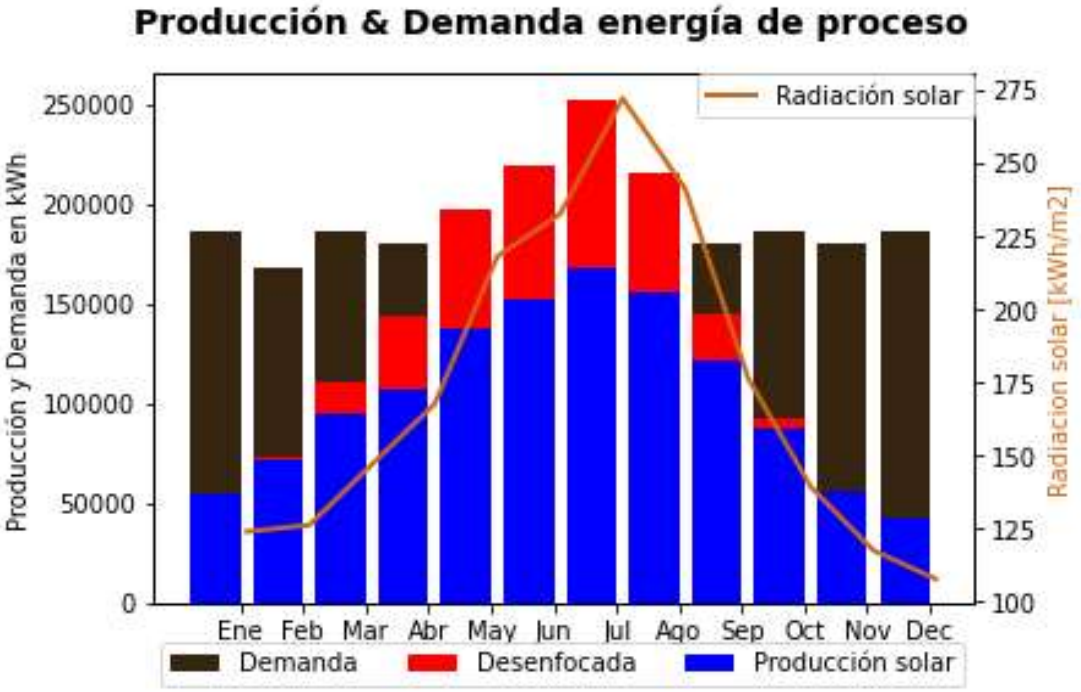

Figura 52: producción y demanda en cada mes del año.

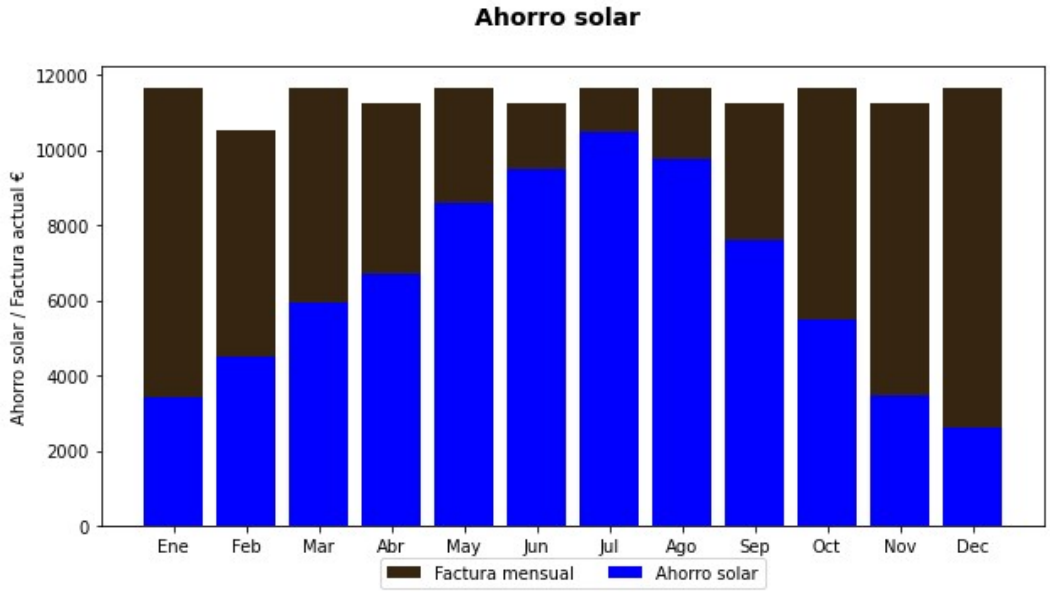

Figura 53: ahorro solar en cada mes del año.

Únicamente en enero, septiembre y diciembre la producción bruta es mayor en el modo horario que en el modo diezminutal. Con respecto a la producción neta, ocurre lo contrario: solamente en febrero, octubre y noviembre la producción neta es mayor en el modo diezminutal. Esto difiere de las conclusiones que se podrían deducir a priori de las simulaciones para días concretos.

El hecho de que la producción neta sea mayor en el modo horario conlleva que el ahorro resulte mayor en dicho modo de simulación. Así, el ahorro en el primer año en el modo horario será 83.622 €, mientras que para el

modo diezminutal será 83.080€. Las mayores variaciones en la producción neta (ahorro) a favor del modo diezminutal se dan únicamente en febrero (61,41€). Mientras que, para el modo horario, las variaciones serán mayores que el modo diezminutal, aunque siguen siendo cambios relativamente pequeños. Al ser mayores, se justifica que el ahorro anual sea mayor para el modo horario: mayo (189,24€), junio (65,20€), agosto (95,44€), septiembre (110,90€) y diciembre (40,48€).

Respecto a la producción bruta, los únicos tres meses en los que la diferencia es mayor del 1% respecto al modo diezminutal ($\frac{horar}{diezminutal}$) son: abril (98,31%), diciembre (101,54%) y febrero (98,52%).

El hecho de que en el modo horario se produzca menos energía bruta y se consuma más recurso solar hace que las simulaciones resulten más viables si se evaluaran de este modo, ya que es necesario producir menos energía (menos pérdidas y desenfoque) y se consume más. Aunque si se observa, las diferencias entre los modos de simulación en ambos parámetros son relativamente pequeñas por lo que convendría dar preferencia al modo diezminutal.

En el resto de las integraciones, las variaciones son similares a las que se han mostrado en la tabla, excepto en la integración SL_S_PD_OT, que presenta mayor variación en la mayoría de los meses. Esto se observa en la simulación anual.

5.2.4 Simulación anual.

Se reflejará el porcentaje que representa el modo horario respecto al modo diezminutal junto a los resultados horarios:

$$\% = \frac{Modo\ horario}{Modo\ diezminutal} \cdot 100$$

Modo de simulación.	*Modelo de integración*	*Producción (kWh)*	*Producción neta (consumo) (kWh)*	*Pérdidas térmicas (kWh)*	*LCOE (€/kWh)*	*Ahorro (€)* / *Ahorro/ factura*
Modo horario	*SL_L_P (aceite)*	1.592.203,895/ 99,59%	1.256.536,373/ 100,65%	34.179,252/ 98,97%	0,031	83.622 / 61,09 %
Modo diezminutal	*SL_L_P(aceite)*	1.598.710,678	1.248.387,380/	34.533,615/	0,031	83.080 / 60,70 %
Modo horario	*SL_L_PS (aceite) (almacenamiento)*	1.592.203,895/ 99,59%	1.313.015,256/ 99,75%	34.179,252/ 98,97%	0,026	87.381 / 63,84 %
Modo diezminutal	*SL_L_PS (aceite) (almacenamiento)*	1.598.710,678	1.316.301,556	34.533,615	0,026	87.600 / 64 %
Modo horario	*SL_L_RF (aceite)*	1.437.951,602/ 99,59%	1.054.580,962/ 100,95%	28.659,145/ 98,67%	0,037	70.182 / 51,27 %
Modo diezminutal	*SL_L_RF (aceite)*	1.443.906,476	1.044.658,166	29.046,880	0,037	69.522 / 50,79 %

Modo horario	*SL_S_FW (agua)*	1.532.489,841/ 99,55%	391.072,485/ 101,65%	92.033,357/ 99,61%	0,097	26.026 19,01 %
Modo diezminutal	*SL_S_FW (agua)*	1.539.462,695	384.740,198	92.396,387	0,098	25.604 18,71 %
Modo horario	*SL_S_FWS (agua) (almacenamiento)*	1.527.683,592/ 99,52%	491.444,482/ 99,12%	96.261,59/ 99,56%	0,068	32.706 23,89 %
Modo diezminutal	*SL_S_FWS (agua) (almacenamiento)*	1.535.023,669	495.825,212	96.689,091	0,068	32.997 24,11 %
Modo horario	*SL_S_PD_OT (vapor)*	1.397.551,814/ 103,20%	1.082.565,414/ 100,64%	97.446,725/ 99,47%	0,031	72.045 52,64 %
Modo diezminutal	*SL_S_PD_OT (vapor)*	1.354.277,203	1.075.631,503	97.967,693	0,031	71.583 52,30 %
Modo horario	*PL_E_PM (vapor)*	1.593.444,936/ 99,64%	1.257.209,771/ 101,61 %	34.144,607/ 0,10%	0,027	83.667 61,13 %
Modo diezminutal	*PL_E_PM (vapor)*	1.599.125,867	1.237.316,447	34.452.947	0,027	83.089 60,70 %
Modo horario	*SL_S_MW (agua)*	1.529.724,152/ 99,49%	385.249,703/ 100,87%	91.096,200/ 99,24%	0,087	25.638 18,73 %
Modo diezminutal	*SL_S_MW (agua)*	1.537.887,436	381.940,484	91.794,732	0,088	25.418 18,57 %
Modo horario	*SL_S_MWS (agua) (almacenamiento)*	1.525.523,981/ 99,49%	538.170,000/ 99,10%	95.226,441/ 99,11%	0,062	35.815 26,17 %
Modo diezminutal	*SL_S_MWS (agua) (almacenamiento)*	1.533.350,096	543.074,524	96.075,697	0,062	36.142 26,41 %
Modo horario	*SL_S_PDS (vapor) (almacenamiento)*	1.480.713,528/ 100,45%	1.295.512,776/ 100,31%	96.588,522/ 98,63%	0,026	86.216 62,98 %
Modo diezminutal	*SL_S_PDS (vapor) (almacenamiento)*	1.474.012,077	1.291.531,287	97.932,262	0,026	85.951 62,80 %

Tabla 16: resultados de simulación anual.

Atendiendo a la producción bruta, se observa el mismo fenómeno que en la simulación mensual: se produce más energía en modo diezminutal que en modo horario en casi todas las integraciones, exceptuando SL_S_PDS y SL_S_PD_OT. Las diferencias entre ambos modos de simulación son relativamente pequeñas (alrededor del 0,5%), excepto en SL_S_PD_OT, que es de un 3,2%. Las que menos varían (menos del 0,5%) son la integración PL_E_PM y los modos de integración en paralelo con líquido térmico (aceite): SL_L_P, SL_L_PS y SL_L_RF.

Respecto a la producción neta, hay 7 de 10 integraciones en las que el modo horario consume más energía útil, repitiéndose la tendencia que se mostraba en la simulación mensual: en el modo horario las simulaciones resultan

más eficientes, pues consumen más recurso solar y presentan menos desenfoque de energía bruta. Las integraciones con mayor variación de energía neta son SL_S_FW y PL_E_PM, con un 1,65% y 1,61% mayor en modo horario, respectivamente. El resto no sufre variaciones mayores del 1%.

Atendiendo a las pérdidas, las mayores variaciones se dan en SL_L_RF (98,67%) y SL_S_PDS (98,63%). En todas las integraciones se reflejan mayores pérdidas térmicas en el modo diezminutal.

Analizando pues la producción bruta, producción neta (ahorro) y pérdidas se deduce que simular en modo horario siempre va a resultar más favorable de cara a los resultados obtenidos. Se produce menos (resulta favorable si no se aprovecha la energía desenfocada para un uso secundario), se consume más recurso solar y se vierte menos energía al ambiente (pérdidas térmicas).

6 CONCLUSIONES

En la versión original del simulador SHIPCal se simulaba solo en paso horario. Debido a que la fuente de energía de los procesos que utilizan energía termosolar es el Sol, se necesitaba estudiar el efecto que tiene la variación de la irradiancia dentro de 1 hora, especialmente si la meteorología es muy variable. Por ello, se ha querido realizar este trabajo. El simulador SHIPCal se podrá seguir mejorando para conseguir cada vez resultados más precisos. Algunas integraciones siguen sufriendo modificaciones al igual que se pueden añadir nuevos modelos de integración. En cuanto a las pérdidas térmicas, solo se están teniendo en cuenta las pérdidas en los receptores en los lazos, pero podría mejorarse el simulador SHIPCal añadiendo funciones que calculen pérdidas en acumuladores, canalizaciones, etc.

Respecto a los resultados, comparando los modos de simulación diezminutal y horario, se observa que, para simulaciones de duración media-corta, los resultados se distancian más o menos en función del tipo de integración, estabilidad de la irradiancia y de si hay recirculación. Para simulaciones de larga duración pueden variar un poco más en términos absolutos, pero dentro de un porcentaje aceptable (exceptuando la integración SL_S_PD_OT). Las simulaciones anuales son las que mejor reflejan el comportamiento de una planta a lo largo del tiempo y los ahorros para el modo horario resultan mayores que en el modo diezminutal, pero por muy poca diferencia. A priori, la instalación solar puede parecer más rentable si se observara desde el modo horario, pues en caso de no tener almacenamiento o un proceso secundario, se desaprovecha menos energía producida (menos desenfoque), ya que se produce menos energía bruta y a la vez se consume más energía solar neta. Sucede lo mismo con las pérdidas térmicas al entorno, que son un poco mayores si se simula en modo diezminutal.

Sin embargo, si se observa desde un punto de vista más realista y fiel a la realidad, el ángulo de incidencia y la irradiancia en cada paso diezminutal muestran una simulación más detallada, mientras que en la simulación horaria el ángulo de incidencia y la irradiancia son la media de cada hora (TMY menos exacto), pudiéndose mostrar resultados menos precisos en el modo horario. Será cuestión de probar con otro modo de simulación (quinceminutal, por ejemplo) y evaluar si se repite la tendencia observada en las simulaciones anuales. De ser así, quedaría reflejado que el modo horario muestra resultados optimistas, pero menos fieles a la realidad.

Atendiendo a los objetivos establecidos al principio del trabajo, puede establecerse que:

-SHIPCal ha sido habilitado con éxito para trabajar en pasos subhorarios.

-El front-end originalmente trabajaba con simulaciones horarias anuales. Se ha conseguido modificarlo para trabajar en pasos subhorarios y realizar simulaciones de duración menor a 1 año.

-Se han comparado los resultados del modo de simulación diezminutal y el modo horario. Puede establecerse a priori que el modo horario muestra resultados “maquillados y optimistas”, pero menos exactos. Debe recordarse que el TMY utilizado ha sido de Sevilla.

-Los resultados obtenidos muestran que para simulaciones de duración media-corta la diferencia entre ambos modos de simulación se acentúa en periodos con irradiancia inestable. En simulaciones de periodos de larga duración (simulación mensual y anual), las diferencias en puntos porcentuales son pequeñas, excepto en la integración SL_S_PD_OT, que en el modo horario llega a presentar un 3,2 % más de energía bruta producida respecto al modo diezminutal.

7 ANEXO.

7.1 Funciones auxiliares a SHIPCal.

7.1.1 Función DemandCreator2.

```
def demandCreator2(totalConsumption,dayArray,weekArray,monthArray,step_minArray,itercontrol):
    days_in_the_month=[31,28,31,30,31,30,31,31,30,31,30,31]

    start_week_day=0

    if itercontrol=='paso_10min':

        weight_of_ten_min_in_day=[]
        weight_of_ten_min_in_month=[]
        weight_of_ten_min_in_year=[]
for month_number in range(12):
            for day_of_the_month in range(days_in_the_month[month_number]):
                day=(start_week_day+day_of_the_month)%7
                for hour in range(len(dayArray)):
                    weight_of_ten_min_in_day += np.multiply(dayArray[hour],step_minArray).tolist()



                weight_of_ten_min_in_month += np.multiply(weekArray[day], weight_of_ten_min_in_day).tolist()
                weight_of_ten_min_in_day=[]
            start_week_day=(day+1)%7
            weight_of_ten_min_in_year +=
np.multiply(monthArray[month_number],weight_of_ten_min_in_month).tolist()
            weight_of_ten_min_in_month=[]
```

```
    renormalization_factor=sum( weight_of_ten_min_in_year)
    totalConsumption_normailized=totalConsumption/renormalization_factor


    annual=np.multiply(totalConsumption_normailized,weight_of_ten_min_in_year)

  elif itercontrol=='paso_15min':

    weight_of_fifteen_min_in_day=[]
    weight_of_fifteen_min_in_month=[]
    weight_of_fifteen_min_in_year=[]

    for month_number in range(12):
      for day_of_the_month in range(days_in_the_month[month_number]):
        day=(start_week_day+day_of_the_month)%7
        for hour in range(len(dayArray)):
          weight_of_fifteen_min_in_day += np.multiply(dayArray[hour],step_minArray).tolist()


        weight_of_fifteen_min_in_month += np.multiply(weekArray[day],
weight_of_fifteen_min_in_day).tolist()
        weight_of_fifteen_min_in_day=[]
      start_week_day=(day+1)%7
      weight_of_fifteen_min_in_year +=
np.multiply(monthArray[month_number],weight_of_fifteen_min_in_month).tolist()
      weight_of_fifteen_min_in_month=[]
```

```
    renormalization_factor=sum( weight_of_fifteen_min_in_year)
    totalConsumption_normailized=totalConsumption/renormalization_factor

    annual=np.multiply(totalConsumption_normailized,weight_of_fifteen_min_in_year)

  return (annual)
```

7.1.2 Función Calc_min_year.

```
def calc_min_year(mes,dia,hora,minute,itercontrol):

  mes_days=(31,28,31,30,31,30,31,31,30,31,30,31)

  num_days=0

cont_mes=mes-1
  if mes<=12:
    while (cont_mes >0):
      cont_mes=cont_mes-1

      num_days=num_days+mes_days[cont_mes]


    if dia<=mes_days[mes-1]:
      num_days=num_days+dia
    else:
      raise ValueError('Day should be <=days_month')
  else:
    raise ValueError('Month should be <=12')
```

```
    if hora<=24:
      hour_year=(num_days-1)*24+hora

       raise ValueError('Hour should be <=24')

    if itercontrol=='paso_10min':

       if minute<=5 and minute>=0:

          min_year=hour_year*6+minute+1
       else:
           raise ValueError('Ten_min has to be 0, 1, 2, 3, 4 or 5')
    elif itercontrol=='paso_15min':
if  minute<=3 and minute>=0:

          min_year=hour_year*4+minute+1

       else :
          raise ValueError('Fifteen_min has to be 0, 1, 2 or 3')
    return min_year
```

7.1.3 Función DemanData2.

```
def
DemandData2(file_demand,mes_ini_sim,dia_ini_sim,hora_ini_sim,mes_fin_sim,dia_fin_sim,hora_fin_sim,
min_ini_sim, min_fin_sim, itercontrol):

    Demand=np.array(file_demand)
```

```
min_year_ini=calc_min_year(mes_ini_sim,dia_ini_sim,hora_ini_sim, min_ini_sim, itercontrol)
min_year_fin=calc_min_year(mes_fin_sim,dia_fin_sim,hora_fin_sim, min_fin_sim, itercontrol)

if min_year_ini <= min_year_fin:
    sim_steps=min_year_fin-min_year_ini

else:
    raise ValueError('End time is smaller than start time')

#Bucle de simulacion
Demand_sim=np.zeros (sim_steps)
step_sim=np.zeros (sim_steps)

step=0
for step in range(0,sim_steps):
    step_sim[step]=step

    Demand_sim[step]=Demand[min_year_ini+step-1]
    step+=1

return Demand_sim
```

7.1.4 Función SolarData2.

```
def
SolarData2(file_loc,mes_ini_sim,dia_ini_sim,hora_ini_sim,min_ini_sim,mes_fin_sim,dia_fin_sim,hora_fin_si
m,min_fin_sim, itercontrol,huso,to_solartime ,long,sender='notCIMAV',Lat=0,Huso=0, optic_type='0'):

min_year_ini=calc_min_year(mes_ini_sim,dia_ini_sim,hora_ini_sim, min_ini_sim, itercontrol)
min_year_fin=calc_min_year(mes_fin_sim,dia_fin_sim,hora_fin_sim, min_fin_sim, itercontrol)

if min_year_ini <= min_year_fin:
```

```
    sim_steps=min_year_fin-min_year_ini #Stablishes the number of steps as the hours between the starting
and ending hours
  else:
    raise ValueError('End time is smaller than start time')

  if sender == 'CIMAV':
    Lat,Huso,Positional_longitude,data,DNI,temp=Meteo_data(file_loc,sender,optic_type)
  elif sender == 'SHIPcal':
    from simforms.models import Locations, MeteoData
    data = MeteoData.objects.filter(location=Locations.objects.get(pk=file_loc)).order_by('hour_year_sim')
    temp = data.values_list('temp',flat=True)
    if optic_type=='concentrator' or optic_type=='0':
      DNI = data.values_list('DNI',flat=True)
    else:

DNI = data.values_list('GHI',flat=True)
  else:
    (data,DNI,temp)=Meteo_data2(file_loc,sender)

  W_sim=np.zeros (sim_steps)
  SUN_ELV_sim=np.zeros (sim_steps)
  SUN_AZ_sim=np.zeros (sim_steps)
  DECL_sim=np.zeros (sim_steps)
  SUN_ZEN_sim=np.zeros (sim_steps)

  step_sim=np.array(range(0,sim_steps)) #np.zeros (sim_steps)
  DNI_sim=np.array(DNI[min_year_ini-1: min_year_fin-1])
  temp_sim=np.array(temp[min_year_ini-1: min_year_fin-1])
  if sender=='SHIPcal':
    month_sim=np.array(data.values_list('month_sim',flat=True)[hour_year_ini-1:hour_year_fin-1])
```

```
    day_sim=np.array(data.values_list('day_sim',flat=True)[hour_year_ini-1:hour_year_fin-1])
    hour_sim=np.array(data.values_list('hour_sim',flat=True)[hour_year_ini-1:hour_year_fin-1])
    hour_year_sim=np.array(data.values_list('hour_year_sim',flat=True)[hour_year_ini-1:hour_year_fin-1])
  else:
    month_sim=data[min_year_ini-1:min_year_fin-1,0]
    day_sim=data[min_year_ini-1:min_year_fin-1,1]
    hour_sim=data[min_year_ini-1:min_year_fin-1,2]
    min_sim=data[min_year_ini-1:min_year_fin-1,3]
    min_year_sim=data[min_year_ini-1:min_year_fin-1,4]

  step=0
  for step in range(0,sim_steps):

    W,SUN_ELV,SUN_AZ,DECL,SUN_ZEN=SolarEQ_simple2(month_sim[step],day_sim[step],hour_sim[step], min_sim[step],huso,to_solartime,long,Lat,Huso)

W_sim[step]=W
    SUN_ELV_sim[step]=SUN_ELV  #rad
    SUN_AZ_sim[step]=SUN_AZ    #rad
    DECL_sim[step]=DECL        #rad
    SUN_ZEN_sim[step]=SUN_ZEN  #rad

    step+=1

output=np.column_stack((month_sim,day_sim,hour_sim,min_sim,min_year_sim,W_sim,SUN_ELV_sim,SUN_AZ_sim,DECL_sim,SUN_ZEN_sim,DNI_sim,temp_sim,step_sim))

  """
```

```
    Output key:
    output[0]->month of year
    output[1]->day of month
    output[2]->hour of day
    output[3]-> minutes portions of hour
    output[4]-> minutes portions of the year
    output[5]->W - rad
    output[6]->SUN_ELV - rad
    output[7]->SUN AZ - rad
    output[8]->DECL - rad
    output[9]->SUN ZEN - rad
    output[10]->DNI  - W/m2
    output[11]->temp -C
    output[12]->step_sim
    """

    if sender == 'CIMAV':

return Lat,Huso,Positional_longitude,output
    else:
        return[output,min_year_ini,min_year_fin]
```

7.1.5 Función Meteo_data2.

```
def Meteo_data2 (file_meteo,sender='notCIMAV', optic_type='0')
      data = np.loadtxt(file_meteo, delimiter="\t")
       DNI=data[:,9]
       temp=data[:,10]
       return [data,DNI,temp]
```

7.1.6 Función SolarEQ_simple2.

```
def SolarEQ_simple2 (Month,Day,Hour,minute,huso,to_solartime,long,Lat,Huso):

   gr=np.pi/180; #Just to convert RAD-DEG

"JUL_DAY=np.loadtxt(os.path.dirname(os.path.dirname(__file__))+'/Solar_modules/Julian_day_prueba.txt',
delimiter='\t');"

   JUL_DAY=np.loadtxt(os.path.dirname(__file__)+'/Solar_modules/Julian_day_prueba.txt',delimiter='\t');

   Jul_day=JUL_DAY[int(Day-1),int(Month-1)];

   DJ=2*np.pi/365*(Jul_day-1); #Julian day in rad

   DECL=(0.006918-0.399912*np.cos(DJ)+ 0.070257*np.sin(DJ)-
0.006758*np.cos(2*DJ)+0.000907*np.sin(2*DJ)-0.002697*np.cos(3*DJ)+0.00148*np.sin(3*DJ));

   DECL_deg=DECL/gr;

   if to_solartime=='on':

      num_days=calc_day_year(int(Month),int(Day))

B=(360/365)*(num_days-81)

      LSTM=15*huso

      EOT=9.87*np.sin(2*B)-7.53*np.cos(B)-1.5*np.sin(B)

      tc=4*(long-LSTM)+EOT

      Hour=tc/60+Hour+((minute)/60)+ 3.5/60

      if Hour==0:

         W_deg=-1*(Hour-12)*15;

         W=W_deg*gr;

      else:

         W_deg=(Hour-12)*15;
```

```
        W=W_deg*gr;
    else:
      if Hour==0:
        W_deg=-1*(Hour+(minute/60)-12)*15;
        W=W_deg*gr;
      else:
        W_deg=(Hour+(minute/60)-12)*15;
        W=W_deg*gr;



    XLat=Lat*gr;
    sin_Elv=np.sin(DECL)*np.sin(XLat)+np.cos(DECL)*np.cos(XLat)*np.cos(W);
    SUN_ELV=np.arcsin(sin_Elv);
    SUN_ELV_deg=SUN_ELV/gr;

    SUN_ZEN=(np.pi/2)-SUN_ELV;

    SUN_AZ=np.arcsin(np.cos(DECL)*np.sin(W)/np.cos(SUN_ELV));

    verif=(np.tan(DECL)/np.tan(XLat));
    if np.cos(W)>=verif:
      SUN_AZ=SUN_AZ;
    else:
        if SUN_AZ >0:
          SUN_AZ=((np.pi/2)+((np.pi/2)-abs(SUN_AZ)));
        else:
          SUN_AZ=-((np.pi/2)+((np.pi/2)-abs(SUN_AZ)));

    return [W,SUN_ELV,SUN_AZ,DECL,SUN_ZEN]
```

7.1.7 Función calc_day_year.

```
def calc_day_year(mes,dia):

    mes_days=(31,28,31,30,31,30,31,31,30,31,30,31)

    num_days=0
    cont_mes=mes-1
    if mes<=12:
        while (cont_mes >0):
            cont_mes=cont_mes-1

            num_days=num_days+mes_days[cont_mes]

        if dia<=mes_days[mes-1]:
            num_days=num_days+dia
        else:
            raise ValueError('Day should be <=days_month')

else:
        raise ValueError('Month should be <=12')

    return (num_days)
```

7.1.8 Función waterFromGrid_v3_min.

```
def waterFromGrid_v3_min(file_meteo, itercontrol, sender='CIMAV'):
    if sender=='CIMAV':
        Tamb = np.loadtxt(file_meteo, delimiter="\t", skiprows=4)[:,7]#Reads the temperature of the weather.
The TMYs are a bit different.
    elif sender=='SHIPcal':
```

```
        from simforms.models import Locations, MeteoData
        meteo_data = MeteoData.objects.filter(location=Locations.objects.get(pk=file_meteo))
        Tamb = meteo_data.order_by('hour_year_sim').values_list('temp',flat=True)
    else:
        Tamb = np.loadtxt(file_meteo, delimiter="\t")[:,10]#Reads the temperature of the weather
    TambAverage=np.mean(Tamb) #Computes the year average
    TambMax=np.amax(Tamb) #Computes the maximum temperature


    offset = 3 #A defined offset of 3 °C
    ratio = 0.22 + 0.0056*(TambAverage - 6.67)
    lag = 1.67 - 0.56*(TambAverage - 6.67)
#The offset, lag, and ratio values were obtained by fitting data compiled by Abrams and Shedd [8], the FloridaSolar Energy Center [9], and Sandia National Labs


    T_in_C_AR=[] #It is easier to work with this array as a list first to print 24 times the mean value of the water temperature for every day


    if itercontrol=='paso_10min':
        for day in range(365):


#The ten-minute year array is built by the temperature calculated for the day printed 144 times for each day
            T_in_C_AR+=[(TambAverage+offset)+ratio*(TambMax/2)*np.sin(np.radians(-90+(day-15-lag)*360/365))]*24*6 #This was taken from TRNSYS documentation.
    elif itercontrol=='paso_15min':
        for day in range(365):
            #The fifteen-minute year array is built by the temperature calculated for the day printed 96 times for each day
            T_in_C_AR+=[(TambAverage+offset)+ratio*(TambMax/2)*np.sin(np.radians(-90+(day-15-lag)*360/365))]*24*4
    return np.array(T_in_C_AR)
```

7.1.9 Función waterFromGrid_trim2.

```
def waterFromGrid_trim2(T_in_C_AR,mes_ini_sim,dia_ini_sim,hora_ini_sim,mes_fin_sim,dia_fin_sim,hora_fin_sim,min_ini_sim,min_fin_sim):


    min_year_ini=calc_min_year(mes_ini_sim,dia_ini_sim,hora_ini_sim, min_ini_sim,itercontrol)

    min_year_fin=calc_min_year(mes_fin_sim,dia_fin_sim,hora_fin_sim, min_fin_sim, itercontrol)


    if min_year_ini <= min_year_fin:

        sim_steps=min_year_fin-min_year_ini

    else:

        raise ValueError('End time is smaller than start time')


    T_in_C_AR_trim=np.zeros (sim_steps)


    step=0

    for step in range(0,sim_steps):

        T_in_C_AR_trim[step]=T_in_C_AR[min_year_ini+step-1]

        step+=1


return (T_in_C_AR_trim)
```

7.1.10 Función ArraysMonth2.

```
def arraysMonth2(itercontrol,Q_prod,Q_prod_lim,DNI,Demand,**kwargs):

 #Para resumen mensual

    if itercontrol=='paso_10min':

        steps=52560

        factor=6

    elif itercontrol=='paso_15min':

        steps==35040
```

```
    factor=4
    Ene_prod=np.zeros(steps)
    Feb_prod=np.zeros(steps)
    Mar_prod=np.zeros(steps)
    Abr_prod=np.zeros(steps)
    May_prod=np.zeros(steps)
    Jun_prod=np.zeros(steps)
    Jul_prod=np.zeros(steps)
    Ago_prod=np.zeros(steps)
    Sep_prod=np.zeros(steps)
    Oct_prod=np.zeros(steps)
    Nov_prod=np.zeros(steps)
    Dic_prod=np.zeros(steps)
    Ene_prod_lim=np.zeros(steps)
    Feb_prod_lim=np.zeros(steps)
    Mar_prod_lim=np.zeros(steps)
    Abr_prod_lim=np.zeros(steps)
    May_prod_lim=np.zeros(steps)
    Jun_prod_lim=np.zeros(steps)

Jul_prod_lim=np.zeros(steps)
    Ago_prod_lim=np.zeros(steps)
    Sep_prod_lim=np.zeros(steps)
    Oct_prod_lim=np.zeros(steps)
    Nov_prod_lim=np.zeros(steps)
    Dic_prod_lim=np.zeros(steps)
    Ene_DNI=np.zeros(steps)
    Feb_DNI=np.zeros(steps)
    Mar_DNI=np.zeros(steps)
```

```
Abr_DNI=np.zeros(steps)
May_DNI=np.zeros(steps)
Jun_DNI=np.zeros(steps)
Jul_DNI=np.zeros(steps)
Ago_DNI=np.zeros(steps)
Sep_DNI=np.zeros(steps)
Oct_DNI=np.zeros(steps)
Nov_DNI=np.zeros(steps)
Dic_DNI=np.zeros(steps)
Ene_demd=np.zeros(steps)
Feb_demd=np.zeros(steps)
Mar_demd=np.zeros(steps)
Abr_demd=np.zeros(steps)
May_demd=np.zeros(steps)
Jun_demd=np.zeros(steps)
Jul_demd=np.zeros(steps)
Ago_demd=np.zeros(steps)
Sep_demd=np.zeros(steps)
Oct_demd=np.zeros(steps)
Nov_demd=np.zeros(steps)
Dic_demd=np.zeros(steps)

for i in range(0,steps):
    if (i<=744*factor-1):
        Ene_prod[i]=Q_prod[i]
        Ene_prod_lim[i]=Q_prod_lim[i]
        Ene_DNI[i]=DNI[i]
        Ene_demd[i]=Demand[i]
    if (i>744*factor-1) and (i<=1416*factor-1):
        Feb_prod[i]=Q_prod[i]
```

```
        Feb_prod_lim[i]=Q_prod_lim[i]
        Feb_DNI[i]=DNI[i]
        Feb_demd[i]=Demand[i]
    if (i>1416*factor-1) and (i<=2160*factor-1):
        Mar_prod[i]=Q_prod[i]
        Mar_prod_lim[i]=Q_prod_lim[i]
        Mar_DNI[i]=DNI[i]
        Mar_demd[i]=Demand[i]
    if (i>2160*factor-1) and (i<=2880*factor-1):
        Abr_prod[i]=Q_prod[i]
        Abr_prod_lim[i]=Q_prod_lim[i]
        Abr_DNI[i]=DNI[i]
        Abr_demd[i]=Demand[i]
    if (i>2880*factor-1) and (i<=3624*factor-1):
        May_prod[i]=Q_prod[i]
        May_prod_lim[i]=Q_prod_lim[i]
        May_DNI[i]=DNI[i]
        May_demd[i]=Demand[i]
    if (i>3624*factor-1) and (i<=4344*factor-1):
        Jun_prod[i]=Q_prod[i]
        Jun_prod_lim[i]=Q_prod_lim[i]

Jun_DNI[i]=DNI[i]
        Jun_demd[i]=Demand[i]
    if (i>4344*factor-1) and (i<=5088*factor-1):
        Jul_prod[i]=Q_prod[i]
        Jul_prod_lim[i]=Q_prod_lim[i]
        Jul_DNI[i]=DNI[i]
        Jul_demd[i]=Demand[i]
```

```
    if (i>5088*factor-1) and (i<=5832*factor-1):
        Ago_prod[i]=Q_prod[i]
        Ago_prod_lim[i]=Q_prod_lim[i]
        Ago_DNI[i]=DNI[i]
        Ago_demd[i]=Demand[i]
    if (i>5832*factor-1) and (i<=6552*factor-1):
        Sep_prod[i]=Q_prod[i]
        Sep_prod_lim[i]=Q_prod_lim[i]
        Sep_DNI[i]=DNI[i]
        Sep_demd[i]=Demand[i]
    if (i>6552*factor-1) and (i<=7296*factor-1):
        Oct_prod[i]=Q_prod[i]
        Oct_prod_lim[i]=Q_prod_lim[i]
        Oct_DNI[i]=DNI[i]
        Oct_demd[i]=Demand[i]
    if (i>7296*factor-1) and (i<=8016*factor-1):
        Nov_prod[i]=Q_prod[i]
        Nov_prod_lim[i]=Q_prod_lim[i]
        Nov_DNI[i]=DNI[i]
        Nov_demd[i]=Demand[i]
    if (i>8016*factor-1):
        Dic_prod[i]=Q_prod[i]
        Dic_prod_lim[i]=Q_prod_lim[i]

Dic_DNI[i]=DNI[i]
        Dic_demd[i]=Demand[i]

array_de_meses=[np.sum(Ene_prod),np.sum(Feb_prod),np.sum(Mar_prod),np.sum(Abr_prod),np.sum(May_
prod),np.sum(Jun_prod),np.sum(Jul_prod),np.sum(Ago_prod),np.sum(Sep_prod),np.sum(Oct_prod),np.sum(
Nov_prod),np.sum(Dic_prod)]
```

```
array_de_meses_lim=[np.sum(Ene_prod_lim),np.sum(Feb_prod_lim),np.sum(Mar_prod_lim),np.sum(Abr_prod_lim),np.sum(May_prod_lim),np.sum(Jun_prod_lim),np.sum(Jul_prod_lim),np.sum(Ago_prod_lim),np.sum(Sep_prod_lim),np.sum(Oct_prod_lim),np.sum(Nov_prod_lim),np.sum(Dic_prod_lim)]


array_de_DNI=[np.sum(Ene_DNI),np.sum(Feb_DNI),np.sum(Mar_DNI),np.sum(Abr_DNI),np.sum(May_DNI),np.sum(Jun_DNI),np.sum(Jul_DNI),np.sum(Ago_DNI),np.sum(Sep_DNI),np.sum(Oct_DNI),np.sum(Nov_DNI),np.sum(Dic_DNI)]


array_de_demd=[np.sum(Ene_demd),np.sum(Feb_demd),np.sum(Mar_demd),np.sum(Abr_demd),np.sum(May_demd),np.sum(Jun_demd),np.sum(Jul_demd),np.sum(Ago_demd),np.sum(Sep_demd),np.sum(Oct_demd),np.sum(Nov_demd),np.sum(Dic_demd)]

   array_de_fraction=np.zeros(12)


   return array_de_meses,array_de_meses_lim,array_de_DNI,array_de_demd,array_de_fraction
```

7.1.11 Función arrays_Savings_Month2.

```
def arrays_Savings_Month2(itercontrol,Q_prod_lim,Demand,Fuel_price,Boiler_eff,**kwargs):
 #Para resumen mensual
   if itercontrol=='paso_10min':
      steps=52560
      factor=6
   elif itercontrol=='paso_15min':
      steps==35040
      factor=4
   Ene_sav_lim=np.zeros(steps)
   Feb_sav_lim=np.zeros(steps)
   Mar_sav_lim=np.zeros(steps)
   Abr_sav_lim=np.zeros(steps)


May_sav_lim=np.zeros(steps)
   Jun_sav_lim=np.zeros(steps)
```

Jul_sav_lim=np.zeros(steps)

Ago_sav_lim=np.zeros(steps)

Sep_sav_lim=np.zeros(steps)

Oct_sav_lim=np.zeros(steps)

Nov_sav_lim=np.zeros(steps)

Dic_sav_lim=np.zeros(steps)

Ene_demd=np.zeros(steps)

Feb_demd=np.zeros(steps)

Mar_demd=np.zeros(steps)

Abr_demd=np.zeros(steps)

May_demd=np.zeros(steps)

Jun_demd=np.zeros(steps)

Jul_demd=np.zeros(steps)

Ago_demd=np.zeros(steps)

Sep_demd=np.zeros(steps)

Oct_demd=np.zeros(steps)

Nov_demd=np.zeros(steps)

Dic_demd=np.zeros(steps)

Ene_frac=np.zeros(steps)

Feb_frac=np.zeros(steps)

Mar_frac=np.zeros(steps)

Abr_frac=np.zeros(steps)

May_frac=np.zeros(steps)

Jun_frac=np.zeros(steps)

Jul_frac=np.zeros(steps)

Ago_frac=np.zeros(steps)

Sep_frac=np.zeros(steps)

Oct_frac=np.zeros(steps)

Nov_frac=np.zeros(steps)

```
Dic_frac=np.zeros(steps)


for i in range(0,steps):
    if (i<=744*factor-1):
        Ene_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Ene_demd[i]=Fuel_price*Demand[i]
        if Ene_demd[i] == 0:
            Ene_frac[i] = 0
        else:
            Ene_frac[i]=Ene_sav_lim[i]/Ene_demd[i]
    if (i>744*factor-1) and (i<=1416*factor-1):
        Feb_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Feb_demd[i]=Fuel_price*Demand[i]
        if Feb_demd[i] == 0:
            Feb_frac[i] = 0
        else:
            Feb_frac[i]=Feb_sav_lim[i]/Feb_demd[i]
    if (i>1416*factor-1) and (i<=2160*factor-1):
        Mar_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Mar_demd[i]=Fuel_price*Demand[i]
        if Mar_demd[i] == 0:
            Mar_frac[i] = 0
        else:
            Mar_frac[i]=Mar_sav_lim[i]/Mar_demd[i]
    if (i>2160*factor-1) and (i<=2880*factor-1):
        Abr_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Abr_demd[i]=Fuel_price*Demand[i]
        if Abr_demd[i] == 0:
            Abr_frac[i] = 0
```

```
else:
            Abr_frac[i]=Abr_sav_lim[i]/Abr_demd[i]
        if (i>2880*factor-1) and (i<=3624*factor-1):
            May_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
            May_demd[i]=Fuel_price*Demand[i]
            if May_demd[i] == 0:
                May_frac[i] = 0
            else:
                May_frac[i]=May_sav_lim[i]/May_demd[i]
        if (i>3624*factor-1) and (i<=4344*factor-1):
            Jun_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
            Jun_demd[i]=Fuel_price*Demand[i]
            if Jun_demd[i] == 0:
                Jun_frac[i] = 0
            else:
                Jun_frac[i]=Jun_sav_lim[i]/Jun_demd[i]
        if (i>4344*factor-1) and (i<=5088*factor-1):
            Jul_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
            Jul_demd[i]=Fuel_price*Demand[i]
            if Jul_demd[i] == 0:
                Jul_frac[i] = 0
            else:
                Jul_frac[i]=Jul_sav_lim[i]/Jul_demd[i]
        if (i>5088*factor-1) and (i<=5832*factor-1):
            Ago_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
            Ago_demd[i]=Fuel_price*Demand[i]
            if Ago_demd[i] == 0:
                Ago_frac[i] = 0
            else:
```

```
        Ago_frac[i]=Ago_sav_lim[i]/Ago_demd[i]


    if (i>5832*factor-1) and (i<=6552*factor-1):
        Sep_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Sep_demd[i]=Fuel_price*Demand[i]
        if Sep_demd[i] == 0:
            Sep_frac[i] = 0
        else:
            Sep_frac[i]=Sep_sav_lim[i]/Sep_demd[i]
    if (i>6552*factor-1) and (i<=7296*factor-1):
        Oct_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Oct_demd[i]=Fuel_price*Demand[i]
        if Oct_demd[i] == 0:
            Oct_frac[i] = 0
        else:
            Oct_frac[i]=Oct_sav_lim[i]/Oct_demd[i]
    if (i>7296*factor-1) and (i<=8016*factor-1):
        Nov_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Nov_demd[i]=Fuel_price*Demand[i]
        if Nov_demd[i] == 0:
            Nov_frac[i] = 0
        else:
            Nov_frac[i]=Nov_sav_lim[i]/Nov_demd[i]
    if (i>8016*factor-1):
        Dic_sav_lim[i]=Fuel_price*Q_prod_lim[i]/Boiler_eff
        Dic_demd[i]=Fuel_price*Demand[i]
        if Dic_demd[i] == 0:
            Dic_frac[i] = 0
        else:
```

```
        Dic_frac[i]=Dic_sav_lim[i]/Dic_demd[i]


array_de_meses_lim=[np.sum(Ene_sav_lim),np.sum(Feb_sav_lim),np.sum(Mar_sav_lim),np.sum(Abr_sav_lim),np.sum(May_sav_lim),np.sum(Jun_sav_lim),np.sum(Jul_sav_lim),np.sum(Ago_sav_lim),np.sum(Sep_sav_lim),np.sum(Oct_sav_lim),np.sum(Nov_sav_lim),np.sum(Dic_sav_lim)]


array_de_demd=[np.sum(Ene_demd),np.sum(Feb_demd),np.sum(Mar_demd),np.sum(Abr_demd),np.sum(May_demd),np.sum(Jun_demd),np.sum(Jul_demd),np.sum(Ago_demd),np.sum(Sep_demd),np.sum(Oct_demd),np.sum(Nov_demd),np.sum(Dic_demd)]


array_de_fraction=[np.sum(Ene_frac),np.sum(Feb_frac),np.sum(Mar_frac),np.sum(Abr_frac),np.sum(May_frac),np.sum(Jun_frac),np.sum(Jul_frac),np.sum(Ago_frac),np.sum(Sep_frac),np.sum(Oct_frac),np.sum(Nov_frac),np.sum(Dic_frac)]


    return array_de_meses_lim,array_de_demd,array_de_fraction
```

REFERENCIAS

[1] «Task 49- integration guideline IEA». [En línea]. Disponible en: https://task49.iea-shc.org/Data/Sites/7/150218_iea-task-49_d_b2_integration_guideline-final.pdf.

[2] «'Software' libre para predecir la producción energética de energía solar fotovoltaica». [En línea]. Disponible en: https://smart-lighting.es/software-energia-solar-fotovoltaica/. [Accedido: 01-oct-2020].

[3] «Sistema solar de concentración de media temperatura para la producción de calor industrial y frío». [En línea]. Disponible en: https://www.agenciaandaluzadelaenergia.es/sites/default/files/EVENTOS/DOCUMENTOS/generacion_termica_con_renovables_solar_concentra.pdf. [Accedido: 01-oct-2020].

[4] «Transparencias de la asignatura Energía Solar de GITI de la US.»

[5] «IDAE-EVALUACIÓN DEL POTENCIAL DE ENERGÍA SOLAR TERMOELÉCTRICA.» [En línea]. Disponible en: https://www.idae.es/uploads/documentos/documentos_11227_e12_termoelectrica_A_fd47d41f.pdf. [Accedido: 05-nov-2020].

[6] «Centrales Solares Termoeléctricas (CSP) - Enerstar». [En línea]. Disponible en: http://www.enerstar.es/ver/179/Centrales-Solares-Termoeléctricas-CSP.html. [Accedido: 05-nov-2020].

[7] M. Frasquet, «Tesis Miguel Frasquet.»

[8] «Capitulo 2». [En línea]. Disponible en: http://bibing.us.es/proyectos/abreproy/70237/fichero/4.%2BCAPITULO%2B2.%2BINTRODUCCI%25C3%2593N.pdf. [Accedido: 08-nov-2020].

[9] «Ángulo horario - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Ángulo_horario#cite_note-2. [Accedido: 08-ago-2020].

[10] «Ángulo acimut | PVEducation». [En línea]. Disponible en: https://www.pveducation.org/es/fotovoltaica/2-propiedades-de-la-luz-del-sol/ángulo-acimut. [Accedido: 26-oct-2020].

[11] «Coordenadas y carta solar - HelioEsfera». [En línea]. Disponible en: https://www.helioesfera.com/coordenadas-y-carta-solar/. [Accedido: 26-oct-2020].

[12] «Anexo A: Aspectos Básicos de la raidación solar». [En línea]. Disponible en: https://www.tesisenred.net/bitstream/handle/10803/6839/10Nvm10de17.pdf?sequence=11&isAllowed=y. [Accedido: 08-ago-2020].

[13] PVeducation.org, «Solar Time». [En línea]. Disponible en: https://www.pveducation.org/pvcdrom/properties-of-sunlight/solar-time#:~:text=Hour Angle (HRA),the sky of 15°.

[14] «Elevación solar | Astropedia | Fandom». [En línea]. Disponible en: https://astronomia.fandom.com/wiki/Elevación_solar. [Accedido: 08-ago-2020].

[15] «Solar zenith angle - Wikipedia». [En línea]. Disponible en: https://en.wikipedia.org/wiki/Solar_zenith_angle. [Accedido: 26-oct-2020].

[16] «HTML - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/HTML. [Accedido: 02-nov-2020].