



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Graduado/a en Ingeniería Electrónica Industrial y Automática

TRABAJO FIN DE GRADO

Propuesta y desarrollo de contribuciones a la librería de
modelado fotovoltaico de código abierto pvlib-python

Autor: Echedey Luis Álvarez

Co-Tutor:
Rubén Núñez Judez
DEPARTAMENTO DE INGENIERÍA
ELÉCTRICA, ELECTRÓNICA, AUTOMÁTICA
Y FÍSICA APLICADA (D180)

Tutor:
César Domínguez Domínguez
DEPARTAMENTO DE INGENIERÍA
ELÉCTRICA, ELECTRÓNICA, AUTOMÁTICA
Y FÍSICA APLICADA (D180)

Madrid, Septiembre, 2024

Este Trabajo Fin de Grado se ha depositado en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

Título: Propuesta y desarrollo de contribuciones a la librería de modelado fotovoltaico de código abierto pvlib-python

Septiembre, 2024

Resumen

La finalidad de este Trabajo Fin de Grado es la contribución de modelos científicos aplicados a la fotovoltaica dentro de la iniciativa de código abierto *pvlib-python*. Adicionalmente, dentro del marco de contribuciones, se han añadido nuevos modelos y datasets, mejoras a la documentación ya existente, varios arreglos al flujo de integración y desarrollo continuo (CI/CD) y corregido múltiples bugs, así como contribuido transversalmente a la mejora de la calidad del proyecto.

Palabras Clave: fotovoltaica, código libre, pvlib-python, simulación, modelado

Abstract

The purpose of this final-year thesis is the contribution of scientific models used in photovoltaic simulation and research. Contributions have been proposed to the free and open source software *pvlib-python*. Merged pull requests range from new models and dataset inclusion, improvements to the existing documentation, various fixes to the continuous integration and continuous development (CI/CD) workflow and multiple bugfixes. Additionally, the quality of the project has been improved from top to bottom.

Keywords: photovoltaic, open source, pvlib-python, simulation, modelling

La historia que aquí se cuenta empieza en Junio de 2022, con una propuesta cuanto menos inesperada y exótica: aportar a una librería de código abierto para simular sistemas fotovoltaicos.

Esta historia no termina hasta semanas después de la entrega de este trabajo, Septiembre, 2024 y la finalización de la subvención otorgada por el programa *Google Summer of Code*.

Agradecimientos

Me gustaría agradecer a mis tutores César Domínguez Domínguez y Rubén Núñez Judez por darme la posibilidad de invertir mi tiempo y capacidad en un proyecto que se alinea con mis objetivos de autorrealización, así como por ofrecerme medios suficientes para formarme en el campo de lo fotovoltaico.

A la UPM por ofrecer máquinas virtuales de Linux al alumnado, tanto para el desarrollo de este proyecto como para hacer pruebas en terceros proyectos.

A Nuria Martín Chivelet por explicarme detalladamente el funcionamiento de su modelo científico y ofrecerme continuar en esa misma línea de trabajo.

A Tomás García Aguado, responsable del departamento de pruebas de alta tensión del *Laboratorio Central Oficial de Electrotecnia, LCOE*, por facilitar información sobre los datos que incluyen sus informes.

A todos los mantenedores de la librería *pulib-python* por sus revisiones en profundidad. En especial a Kevin Anderson y a Adam Jensen por ofrecerme y guiarme en obtener una beca bajo el programa *Google Summer of Code*¹.

A todas las personas que públicamente han contribuido directa o indirectamente en crear ecosistemas de desarrollo de software libre y de código abierto, en especial a aquellos involucrados en la comunidad de Python y Visual Studio Code entre innumerables otros.

Finalmente, a Aurelio Acevedo Rodríguez por inculcarme la importancia de la sección de agradecimientos. En paz descanse.

¹<https://summerofcode.withgoogle.com/programs/2024/projects/fxPFQqZc>.

Índice general

1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Contexto del proyecto	1
1.3. Objetivos	2
1.4. Estructura del Documento	3
2. Trabajo relacionado y Estado del Arte	4
2.1. La energía solar fotovoltaica	4
2.2. Simulación de sistemas fotovoltaicos	7
2.3. La librería pvlib-python	7
2.3.1. Objetivos de la librería	8
2.3.2. Funcionalidades	9
2.3.3. Repositorio del proyecto	11
2.3.4. Frameworks de desarrollo del proyecto	11
3. Desarrollo	13
3.1. Entorno de desarrollo y herramientas utilizadas	13
3.1.1. La documentación	14
3.2. Contribuciones científicas	19
3.2.1. Modelado de ajuste espectral	19
3.2.1.1. Fundamento teórico	19
3.2.1.2. Resultado	20
3.2.2. Proyección del cenit solar sobre las coordenadas de un colector	20
3.2.2.1. Fundamento teórico	21
3.2.2.2. Resultado	21
3.2.3. Cálculo de fracción de sombra unidimensional	21
3.2.3.1. Fundamento teórico	22
3.2.3.2. Resultado	23
3.2.4. Pérdidas por sombreado en módulos con diodos de bypass	23
3.2.4.1. Fundamento teórico	23
3.2.4.2. Resultado	25
3.2.5. Fracción diaria de radiación difusa fotosintetizable en función de la fracción difusa global	25
3.2.5.1. Fundamento teórico	25
3.2.5.2. Resultado	25
3.2.6. Modelo de pérdidas por heterogeneidad de irradiancia por célula	26
3.2.6.1. Fundamento teórico	26
3.2.6.2. Resultado	27

3.2.7. Transformación de respuesta espectral a eficiencia cuántica externa y viceversa	27
3.2.7.1. Fundamento teórico	27
3.2.7.2. Resultado	27
3.2.8. Adición de base de datos de respuesta espectral de algunas tecnologías	28
3.2.8.1. Fundamento teórico	28
3.2.8.2. Resultado	28
3.2.9. Adición de espectro estándar completo ASTM G173-03	28
3.2.9.1. Fundamento teórico	28
3.2.9.2. Resultado	29
3.2.10. Cálculo geométrico de sombras en 3D	31
3.2.10.1. Fundamento teórico	31
3.2.10.2. Resultado	32
3.3. Contribuciones técnicas	35
3.3.1. Arreglo a los tests de integración continua en Windows con Conda	35
3.3.1.1. Resultado	35
3.3.2. Arreglo a un parámetro ignorado en una función de transposición inversa	36
3.3.2.1. Resultado	36
3.3.3. Dar soporte a otra función para el cálculo del IAM en el flujo orientado a objetos	36
3.3.3.1. Resultado	36
3.3.4. Suprimir una advertencia al publicar la distribución en PyPI	36
3.3.4.1. Resultado	37
3.3.5. Exponer parámetros de tolerancia para resolver el modelo de un diodo	37
3.3.5.1. Resultado	37
3.3.6. Modificar tolerancias erróneas en varios tests unitarios	37
3.3.6.1. Resultado	37
3.3.7. Arreglo de un bug que ignoraba parámetros de una función de lectura de bases de datos	37
3.3.7.1. Resultado	38
3.3.8. Actualizar versiones de las dependencias de la documentación	38
3.3.8.1. Resultado	38
3.4. Contribuciones menores	38
3.4.1. Corrección de erratas en la documentación	38
3.4.2. Corrección de erratas en ejemplos y en código	38
3.4.3. Modificación de escritura de los parámetros opcionales	38
3.4.4. Limpieza de advertencias al construir la documentación	39
3.5. Interacciones con otros usuarios	40
3.5.1. Revisión de propuestas de otros usuarios	40
3.5.2. Soporte técnico a usuarios	40
4. Impacto del trabajo	42
4.1. Impacto general	42
4.2. Objetivos de Desarrollo Sostenible	42
4.3. Impacto académico en el autor	43
5. Resultados y conclusiones	44

ÍNDICE GENERAL

5.1. Resultados	44
5.2. Conclusiones	45
5.3. Trabajo futuro	46
Bibliografía	47

Índice de Figuras

2.1. Componentes de la radiación solar y los instrumentos de medida. Recuperado de http://web.archive.org/web/20230624215046/https://yellowhaze.in/wp-content/uploads/2017/01/dni-dhi-ghi-1-768x437.png	6
3.1. Un ejemplo de renderizado de la documentación de una función en <i>pplib-python</i>	16
3.2. Un ejemplo de página de ejemplo en la documentación de <i>pplib-python</i>	18
3.3. Esquema dos colectores parametrizados, donde uno sombrea al otro. La nomenclatura corresponde la ecuación 3.4. Fuente: Figura 3 en [21].	22
3.4. Esquema de un módulo con 3 diodos de bypass.	24
3.5. Ejemplo en la documentación con el espectro estándar ASTM G173-03 completo.	30
3.6. Diagrama UML de la propuesta de cálculo de sombras en 3D.	33
3.7. Ejemplo de sombreado para coordenadas solares instantáneas en 3D.	35

Índice de Tablas

3.1. Algunas de las expresiones regulares empleadas para modificar la escritura de los parámetros opcionales.	39
---	----

Índice de Listings

3.1. Ejemplo de documentación de una función en <i>pplib-python</i>	15
3.2. Plantilla para elaborar un ejemplo en <i>pplib-python</i>	16
3.3. Comandos para construir la documentación de <i>pplib-python</i>	19
3.4. Caso de uso de ejemplo para la propuesta de cálculo de sombras en 3D.	33

Capítulo 1

Introducción

El cambio climático es un tema de actualidad que plantea un reto social, económico y tecnológico. Dentro de este marco, las energías renovables se presentan como una solución tecnológica a la dependencia de los combustibles fósiles, que son los principales responsables de la emisión de gases de efecto invernadero. Una de las tantas fuentes de energía renovables más prometedoras es la solar fotovoltaica, ya que es renovable y no contamina en su explotación directa.

Este Trabajo de Fin de Grado pretende potenciar la adquisición, investigación e implementación de la energía solar, mejorando herramientas de simulación y diseño de instalaciones fotovoltaicas. Para ello, se han realizado múltiples contribuciones a un proyecto de código abierto llamado *pulib-python*, que es una biblioteca de herramientas escrita en Python para el análisis de sistemas fotovoltaicos.

1.1. Motivación del proyecto

El alumno declara que la motivación para la realización de este Trabajo de Fin de Grado se fundamenta en su interés por las energías renovables, el código libre y su ya experiencia previa en desarrollo de software en Python, también de acceso abierto.

Asimismo una de las principales inquietudes del alumno es aplicar sus conocimientos en generar o facilitar nueva ciencia de forma accesible y contrastable, y que pueda ser utilizada por la comunidad científica y técnica de manera completamente transparente.

1.2. Contexto del proyecto

Con el incremento del interés por las energías renovables y las facilidades del desarrollo software como caldo de cultivo, se ha propuesto al alumno la realización de este Trabajo de Fin de Grado, que consiste en apoyar en el desarrollo de la biblioteca *pulib-python*.

La propuesta parte de los tutores del alumno, que son miembros del grupo de investigación *Instruments and Systems Integration* del *Instituto de Energía Solar* de la propia Universidad Politécnica de Madrid. Asimismo, el proyecto *pulib-python* es un

proyecto de código abierto que se encuentra en desarrollo por otros investigadores de centros de investigación y universidades públicas de múltiples países, y algunos miembros de empresas privadas del mismo campo.

El perfil de usuario de esta biblioteca se puede clasificar entre 3 grupos principales:

- Investigadores que desean realizar simulaciones y estudios de sistemas fotovoltaicos.
- Ingenieros y técnicos que desean optimizar el diseño de instalaciones fotovoltaicas.
- Ingenieros y técnicos que quieren identificar faltas en este tipo de instalaciones, mediante la comparación de datos reales con simulaciones.

Por supuesto, tratándose de un proyecto de código abierto, cualquier persona puede utilizar la biblioteca, por lo que no se descarta la posibilidad de que otros perfiles de usuario puedan beneficiarse de las mejoras realizadas en este Trabajo de Fin de Grado. En este aspecto, sucede que esta iniciativa democratiza el acceso por parte de usuarios más noveles en el campo de la energía solar fotovoltaica, que no tienen acceso a herramientas comerciales.

1.3. Objetivos

La línea principal de este trabajo es la adición de nuevas funcionalidades a la biblioteca *pulib-python*, que permitan mejorar la simulación, investigación y diseño de plantas fotovoltaicas. Para ello, se han establecido los siguientes objetivos:

- Contribuir modelos científicos variados en propósito y utilidad.
- Añadir otras funcionalidades, que no siendo modelos, sean útiles para el usuario.
- Validar el funcionamiento mediante tests unitarios.
- Hacer que cada contribución sea usable aportando una documentación completa y concisa, didáctica si fuere necesario.
- Seguir los rigurosos estándares de calidad de un proyecto científico de código libre.
- Ayudar a la comunidad de usuarios de la biblioteca a través de la resolución de dudas y problemas.
- Participar en la revisión del código de otros contribuyentes.
- Crear asuntos y cuestiones que promuevan la mejora de la biblioteca, en especial para animar a otros contribuyentes a participar.
- Ayudar a mantener la biblioteca actualizada para mejorar su ciclo de vida y arreglar fallos de forma preventiva.

Por otra parte, entre los objetivos secundarios destacan:

- Dar visibilidad a autores nacionales y de la Universidad Politécnica de Madrid.

- Potenciar proyectos de beneficio común desde la Universidad pública.

Si bien no es el propósito específico de este trabajo tratar estos últimos objetivos, se considera que son una consecuencia natural y deseable del presente proyecto.

1.4. Estructura del Documento

La estructura de este Trabajo de Fin de Grado pretende ser intuitiva y distribuida por bloques sobre temas similares. El lector podrá leer a continuación:

- En Capítulo 2 *Trabajo relacionado y Estado del Arte*, se da a conocer el estado actual de la energía solar fotovoltaica y algunas de las herramientas de simulación utilizadas.
 - En Sección 2.3 *La librería pulib-python*, se presentan las características de la biblioteca *pulib-python*.
- En Capítulo 3 *Desarrollo*, se detalla el desarrollo de las contribuciones propuestas a la biblioteca, con factores tanto técnicos como humanos sobre el resultado.
- En Capítulo 4 *Impacto del trabajo*, se presenta el impacto de las contribuciones realizadas tanto en la biblioteca como en otras, y en la comunidad del proyecto.
- En Capítulo 5 *Resultados y conclusiones*, se resume y se hace una valoración global de los resultados obtenidos.

Capítulo 2

Trabajo relacionado y Estado del Arte

En este capítulo se cubre el estado del arte de la energía solar fotovoltaica y, particularmente, de la librería *pplib-python*. El lector podrá encontrar en las siguientes secciones:

- En Sección 2.1 *La energía solar fotovoltaica*, se explica el estado actual de la energía solar fotovoltaica y su fundamento teórico base.
- En Sección 2.2 *Simulación de sistemas fotovoltaicos*, se detallan las herramientas de simulación utilizadas en el sector fotovoltaico y el marco general de comparación de la librería *pplib-python*.
- En Sección 2.3 *La librería pplib-python*, se presentan las características de la biblioteca *pplib-python*.

2.1. La energía solar fotovoltaica

En esta sección se presentará el estado del arte de las diferentes tecnologías, estudios y sistemas relacionados con la energía solar fotovoltaica.

La energía solar fotovoltaica es una tecnología que convierte la radiación solar en electricidad utilizando células solares mediante el efecto fotoeléctrico [1, pp. 701-706]. Este fenómeno consiste en la generación de una corriente eléctrica cuando la luz incide sobre un material semiconductor y excita los electrones de la banda de valencia a la banda de conducción. Esta excitación genera un par electrón-hueco que se separa por la acción de un campo eléctrico externo (en cuyo caso no se produce energía neta positiva) o mediante la distribución de cargas en un semiconductor p-n, que permite la extracción de energía. Este último caso es el de aplicación en células solares fotovoltaicas, pues la intención es obtener energía eléctrica.

Existen varias tecnologías de células solares, como las de silicio, las de película delgada y las más experimentales de concentración y de otros materiales orgánicos y multiunión, que se agrupan en generaciones [2]. Cada generación responde a una serie de características e implantación en el mercado, donde destacan:

- Primera generación: células de silicio monocristalino y policristalino. Se encuentran bien implantadas en el mercado y son las más utilizadas en aplicaciones fotovoltaicas. Según el límite teórico que calcularon Shockley y Queisser en 1961, el silicio es el material más apropiado para la fabricación de células solares, ya que su banda prohibida de 1.1 eV es la que mejor se ajusta al máximo de la radiación solar [1, p. 1126]. Sin embargo, presentan un coste de producción moderado y un alto uso de material. El límite de eficiencia teórico obtenido por los anteriores autores es del 33.7 % [3], pero asumen que no tratan con células solares de concentración ni con células solares de múltiples uniones o tandem.
- Segunda generación: células de película delgada, como las de telururo de cadmio (*CdTe*), las de di-seleniuro de cobre, indio y galio (*Copper indium gallium selenide*, *CuInGaSe₂*), las de arseniuro de galio (*Gallium arsenide*) (*GaAs*) y las de silicio amorfo (a-Si:H). Destacan por ser de capa delgada (*thin-film*) y, consecuentemente, más baratas de producir por el bajo uso de material, si bien pueden llegar a ser composiciones de materiales más caros. Nótese que el silicio amorfo es ampliamente utilizado, en especial en aplicaciones de baja potencia, como calculadoras, pero su eficiencia es inferior a las células de silicio cristalino.
- Tercera generación: células de concentración, células de múltiples uniones y células orgánicas. Se encuentran en fase de investigación y desarrollo, y se caracterizan por su alta eficiencia y coste elevado. Por un lado destacan la tecnología de concentración, que emplea lentes para concentrar la luz solar en células solares de alta eficiencia, normalmente de múltiples uniones, que pueden alcanzar eficiencias superiores al 40 % [4, Tabla 5]. Se emplean sistemas ópticos para disminuir el uso de material semiconductor, el principal factor de coste en estas células.

Cada una tiene sus propias características y aplicaciones específicas. Las células de primera y segunda generación son las más comunes en aplicaciones fotovoltaicas, mientras que las de tercera generación se encuentran en fase de investigación y desarrollo. Se remite el lector a [2] para una revisión más detallada de cada grupo y las peculiaridades de cada material.

En cuanto a los sistemas fotovoltaicos, se han desarrollado diferentes configuraciones, como sistemas conectados a la red, sistemas autónomos y sistemas de bombeo [5]. Cada configuración tiene sus propias ventajas y desafíos, y se han realizado investigaciones para optimizar su diseño y operación. En todos estos casos, es fundamental contar con herramientas de simulación y análisis para evaluar el rendimiento de los sistemas, optimizar su diseño y diagnosticar fallos a partir de datos meteorológicos.

Toda simulación para un sistema fotovoltaico tradicional -en referencia a los colectores planos y no aquellos de concentración, que presentan otras características- se basa en la radiación solar incidente en la superficie de los módulos. Los datos de partida de una simulación siempre son determinados valores meteorológicos como la radiación solar, la temperatura ambiente y la velocidad del viento. Adicionalmente, se emplean modelos empíricos para obtener valores como la radiación solar extraterrestre en la superficie de la atmósfera y deducir otros parámetros de entrada para el resto del flujo de cálculos, en función de los instantes de tiempo que se vayan a analizar.

2.1. La energía solar fotovoltaica

A partir de unos valores de *GHI*, *DNI* y *DHI* -que son las componentes de la radiación solar global en una superficie horizontal, normal al vector solar y difusa en la superficie horizontal, respectivamente- se pueden obtener los valores de radiación solar en una superficie inclinada y orientada en función de los ángulos de inclinación y orientación de los módulos. Estas se relacionan mediante:

$$GHI = DNI \cdot \cos(\theta) + DHI \quad (2.1)$$

La existencia de los parámetros de entrada mencionados se explica mediante la necesidad de independizar las medidas de irradiancia solar de la superficie de interés, y lo que permiten cuantificar los instrumentos de medida en campo. Estos se tratan de piranómetros para medir *GHI*; pirheliómetros, para *DNI*; y piranómetros con disco para *DHI*. En la figura 2.1 se puede observar la relación entre las componentes de la radiación solar y los instrumentos de medida:

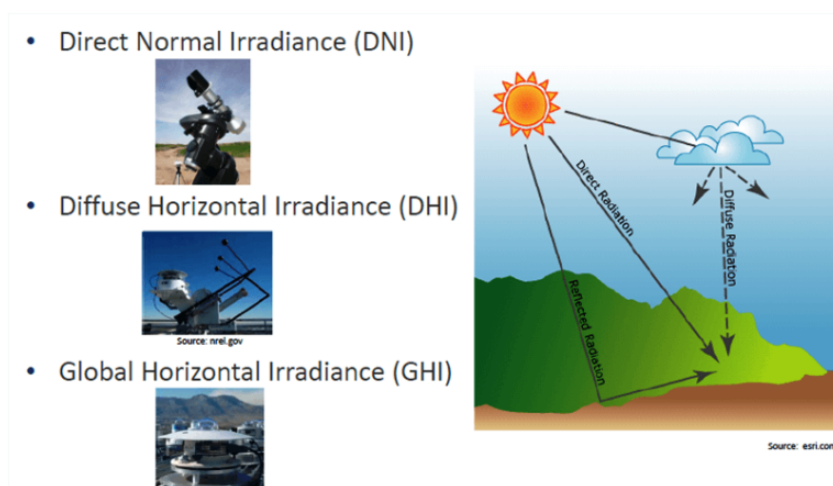


Figura 2.1: Componentes de la radiación solar y los instrumentos de medida. Recuperado de <http://web.archive.org/web/20230624215046/https://yellowhaze.in/wp-content/uploads/2017/01/dni-dhi-ghi-1-768x437.png>

Esta radiación se descompone en tres componentes que dependen de la superficie incidente: la radiación directa, la radiación difusa y la radiación reflejada. La radiación directa es la que proviene del sol y llega directamente a la superficie de los módulos, la radiación difusa es la que proviene del cielo y se dispersa en todas direcciones, y la radiación reflejada o albedo es la que proviene de la reflexión y emisión de radiación sobre superficies cercanas. Estas componentes se suman para obtener la radiación total incidente en la superficie de los módulos, que es la que se emplea para calcular la producción de energía.

En resumen, el estado del arte de la energía solar fotovoltaica abarca una amplia gama de tecnologías, estudios y sistemas. En este Trabajo de Fin de Grado se tratarán algunas de las propuestas que se han realizado en el ámbito de la simulación y el análisis de sistemas fotovoltaicos, con el enfoque particular en la librería *pvl-lib-python*.

2.2. Simulación de sistemas fotovoltaicos

La importancia de simulaciones y análisis previo a la implantación de sistemas fotovoltaicos tanto para inversores, operaciones de financiación y diseñadores ha dado lugar a múltiples herramientas software y modelos como PVWatts, PVGIS, PV-Online, PV*SOL, PVsyst, System Advisor Model (SAM) y muchas más [6, 7]. Normalmente estas herramientas propietarias tienen un foco muy específico en que un usuario pueda calcular el rédito energético y económico de una instalación fotovoltaica, pero no en la investigación y validación de modelos científicos. En la Sección 2.3 se comprobará que la librería *pvlib-python* se centra en la investigación y validación de modelos científicos para la simulación de sistemas fotovoltaicos, y que estos cálculos de rédito económico se dejan a manos del usuario.

Dentro de este grupo de herramientas, surge la iniciativa de código abierto PVLIB-MatLab con origen en los laboratorios de *Sandia National Laboratories, EEUU* hacia el año 2009 [8]. Posteriormente, en 2013¹ inicia el desarrollo de la versión en Python [9, 10, 11, 12, 13], que es de la que trata este trabajo.

La librería *pvlib-python* aporta una serie de mejoras a su contrapartida en MATLAB, entre ellas la infraestructura de tests, la documentación y otros procedimientos de Integración Continua y Desarrollo Continuo (CI/CD) que facilitan el desarrollo, opuesto a la ausencia de tests, control de versiones en un documento de Microsoft Word y, en general, la falta de mantenimiento y actividad de la versión en MATLAB².

2.3. La librería *pvlib-python*

La librería *pvlib-python* es una biblioteca de código abierto que proporciona herramientas para la simulación, análisis e investigación de sistemas fotovoltaicos. Se encuentra desarrollada para el lenguaje de programación interpretado Python [14], que es ampliamente utilizado en la comunidad científica y de desarrollo de software por su sintaxis sencilla, facilidad de desarrollo y diseño orientado a objetos.

Como todo proyecto de código abierto que se encuentra en constante desarrollo, esta librería cuenta con un grupo de desarrolladores y colaboradores que contribuyen a su mantenimiento y mejora continua. Estas personas partícipes del proyecto son mayoritariamente científicos e investigadores de diferentes instituciones y universidades, normalmente públicas, e ingenieros u otros investigadores del ámbito privado. El código y la colaboración se realiza a través del repositorio de código abierto en GitHub³.

El código se encuentra disponible al público bajo la licencia *BSD 3-Clause*⁴, que permite su uso, modificación, redistribución y uso en otros proyectos siempre que haga la atribución de autoría pertinente y no emplee la imagen de *pvlib-python* con fines publicitarios o promotores. La librería se distribuye a través del índice de paquetes

¹<https://web.archive.org/web/20240411190506/https://pvlib-python.readthedocs.io/en/stable/#history-and-acknowledgement>

²http://web.archive.org/web/20211207215130/https://pvlib-python.readthedocs.io/en/v0.9.0/comparison_pvlib_matlab.html.

³Véase <https://github.com/pvlib/pvlib-python>.

⁴Texto de la licencia disponible en <https://opensource.org/license/bsd-3-clause>

de Python, PyPI⁵ y en el canal de *conda-forge*⁶ para la distribución de Python *Conda*, orientada a facilitar conjuntos de librerías para ciencia de datos sin que existan conflictos. *Conda* se distribuye en dos sabores -o conjuntos de librerías- conocidas como *Anaconda* y *Miniconda*, cada una con sus propias características y ventajas en tiempo de instalación, espacio en disco y paquetes preinstalados.

La documentación de la librería⁷ se encuentra disponible en la plataforma *ReadTheDocs*, donde se detallan las funcionalidades, los ejemplos de uso, la estructura del código y la API⁸, las herramientas de desarrollo, las directrices para contribuir al proyecto y la historia de cambios y versiones. La documentación se encuentra disponible exclusivamente en inglés.

2.3.1. Objetivos de la librería

Como se ha comentado anteriormente, la librería *pvlib-python* tiene como objetivo principal proporcionar herramientas para la simulación, análisis e investigación de sistemas fotovoltaicos. Para ello, cuenta con una serie de funciones y clases que permiten realizar cálculos de radiación solar, generación de energía, sombreado, eficiencia de módulos, pérdidas de energía, entre otros. Estas herramientas se basan en modelos científicos y métodos de cálculo validados por la comunidad científica y se han implementado en Python para facilitar su uso y extensión. Es importante denotar que los autores pretenden establecer la implementación de la librería como una referencia fiable y certera de los modelos científicos que se emplean en la simulación de sistemas fotovoltaicos, con un enfoque en la rigurosidad de las publicaciones y artículos científicos que se emplean como base.

Enumerando los objetivos de la librería, se pueden destacar los siguientes:

1. Proporcionar herramientas para la simulación y análisis de sistemas fotovoltaicos.
2. Implementar modelos científicos y métodos de cálculo validados por la comunidad científica.
3. Establecer un referente fiable de las implementaciones de las características citadas anteriormente.
4. Auditar las contribuciones para garantizar la funcionalidad de las aportaciones.
5. Facilitar la integración con servicios de datos meteorológicos y bases de datos de radiación solar.
6. Fomentar la colaboración y la contribución de la comunidad científica y de desarrollo de software.

Cabe destacar que también existen los no-objetivos de la librería, que se refieren a las características que no se pretenden implementar o documentar. Entre estos destacan:

⁵Véase <https://pypi.org/project/pvlib/>

⁶Véase <https://anaconda.org/conda-forge/pvlib>

⁷Accesible en <https://pvlib-python.readthedocs.io/en/stable/>

⁸API es la contracción de *Application Programming Interface* y se trata del conjunto de utilidades del que disponen los usuarios de la librería.

1. Proporcionar contenido didáctico sobre la energía solar fotovoltaica en términos generales, más allá del necesario para entender y aplicar las herramientas de la librería por usuarios con una base de los fundamentos.
2. Hacer inferencias, sugerencias o recomendaciones que no estén publicadas formalmente sobre el uso de los modelos implementados.
3. Facilitar un sistema *ready-to-go* para usuarios finales.

Puede parecer que estos no-objetivos son limitantes, pero en realidad se trata de una forma de garantizar la calidad y la fiabilidad de la librería, ya que se centra en la implementación rigurosa de publicaciones de la comunidad científica.

Una nota importante, en especial sobre el último punto, es que la definición de la misma librería es que se trata de una *toolbox*⁹. Esta nota es en contraposición a las herramientas de simulación y análisis de sistemas fotovoltaicos comerciales, que limitan y simplifican la experiencia del usuario, varias veces dejando de lado la documentación y la transparencia de los cálculos¹⁰.

2.3.2. Funcionalidades

El proyecto de código abierto *pvlib-python* cuenta múltiples características que abarcan un amplio rango de temas relacionados con la producción de energía solar fotovoltaica. Al momento de la redacción de este documento, la versión de la librería es la 0.11.0, publicada el 21 de junio de 2024. A continuación, se detallan algunas de las funcionalidades más destacadas:

- Cálculo de la posición del sol: para determinar la posición del sol en el cielo en función de la localización y la hora del día, en el submódulo `pvlib.solarposition`.
- Cálculo de valores estándares de la radiación solar: la librería proporciona herramientas para calcular la radiación solar en una superficie plana y horizontal en función de la localización, la posición del sol y parámetros atmosféricos. Véase `pvlib.solarposition` y `pvlib.clearsky`.
- Procedimientos de descomposición, transposición y transposición inversa de la radiación solar: se emplean para, a partir de la radiación solar en una superficie plana y horizontal, obtener la radiación solar en una superficie inclinada y orientada determinados ángulos. Y una vez obtenidas las componentes directa, difusa y de albedo de la irradiación, aplicar correcciones convenientes para obtener la radiación efectiva colectada en la superficie. También se puede realizar el proceso inverso, de las componentes en el plano inclinado a la radiación en una superficie plana y horizontal. Estas utilidades se encuentran en el submódulo `pvlib.irradiance`.
- Obtención de parámetros atmosféricos como la columna o masa de aire, un número que representa cuánta atmósfera hay, relativa a la columna completamente vertical ($A_{AM}=1$), en el submódulo `pvlib.atmosphere`.

⁹En español, “caja de herramientas”. <https://github.com/pvlib/pvlib-python/blob/99619e8fc5aea5c5dc4dacabb75b617786cc4a1a/README.md?plain=1#L61>

¹⁰Como hecho anecdótico, se puede observar alguna crítica sobre esto entre los participantes del proyecto, por ejemplo en <https://github.com/pvlib/pvlib-python/issues/2057#issuecomment-2197279047>.

- Valores de albedo predefinidos, bien de materiales sólidos o de superficies naturales de agua, en `pvlib.albedo`.
- Corrección de la radiación incidente en función del ángulo de incidencia: debido a la reflexión que sufre la luz solar al incidir en una superficie de forma oblicua. Las utilidades se encuentran en el submódulo `pvlib.iam`.
- Producción fotovoltaica: mediante puntos característicos como el de máxima potencia (*MPP*) o calculando curvas I-V completas. Se emplean modelos de eficiencia de módulos y pérdidas de energía para estimar la producción de energía en un sistema fotovoltaico, mediante el modelo de un único diodo. Se encuentra en el submódulo `pvlib.singlediode`.
- Cálculo de sombreado: métodos analíticos para calcular la fracción sombreada de hileras de filas. Véase el submódulo `pvlib.shading`.
- Cálculo de ángulos de seguimiento: para seguidores de uno y dos ejes, que permitan coleccionar la máxima radiación solar directa, que normalmente se corresponderá con la máxima producción de energía. Véase el submódulo `pvlib.tracking`.
- Cálculo de temperatura de los módulos: normalmente empíricos, facilitan estimar la producción de energía ya que, habitualmente, la eficiencia de las células es bastante susceptible a la temperatura. Estas utilidades se encuentran en `pvlib.temperature`.
- Estimación de ganancias o pérdidas por efectos del espectro solar: pues en función de la composición del espectro solar incidente en los módulos, la eficiencia de los mismos puede variar. En `pvlib.spectrum`.
- Modelado de eficiencia de inversores: para estimar la eficiencia de los inversores DC-AC en función de la potencia de entrada y sus demás características. En `pvlib.inverter`.
- Modelado de pérdidas en transformadores: para estimar las pérdidas en los transformadores de conexión a red. En `pvlib.transformer`.
- Modelado de pérdidas en cables: para estimar las pérdidas resistivas en los cables. En `pvlib.pvsystem`.
- Integración de APIs externas para la obtención de datos meteorológicos relacionados con la fotovoltaica, en `pvlib.iotools`.
- Abstracciones de los componentes que conforman una instalación fotovoltaica, mediante las clases `Location`, `Array`, `PVSystem` y la clase que agrupa el flujo computacional `Modelchain`, en los submódulos `pvlib.location`, `pvlib.pvsystem` y `pvlib.modelchain`.
- Cálculos adicionales para sistemas bifaciales, que son aquellos colectores planos que permiten la captación de radiación solar por ambas caras del módulo. En `pvlib.bifacial`.

2.3.3. Repositorio del proyecto

Como se ha mencionado anteriormente, el código de la librería *pvlb-python* se encuentra alojado en un repositorio de código abierto. Un repositorio es una carpeta que contiene una serie de archivos, de los cuales los más importantes se gestionan mediante un sistema de control de versiones, VCS por sus siglas en inglés. Este sistema permite llevar un registro de los cambios realizados en los archivos, así como la posibilidad de volver a versiones anteriores en caso de que se produzca un error o un fallo en el código. El uso de esta herramienta es indispensable en proyectos de software, ya que facilita la colaboración entre los desarrolladores y la gestión de las versiones del código.

En el caso de *pvlb-python*, el repositorio se encuentra alojado en la plataforma GitHub, que es una de las más populares para el alojamiento de proyectos de código abierto.

El repositorio cuenta con múltiples archivos y sub-carpetas que configuran el proyecto. Algunos de los más destacables son:

- `README.md`: es el primer archivo en verse en la plataforma online y contiene la descripción del proyecto, su propósito y cómo instalarlo, entre otros.
- `LICENSE`: es el archivo que contiene la licencia del proyecto, en este caso la licencia *BSD 3-Clause*.
- `pyproject.toml`, `setup.cfg`, `setup.py`, `MANIFEST.in`: son los archivos que definen la configuración del proyecto, como el nombre, la versión, la descripción, las dependencias, los archivos que componen una distribución precompilada o de código fuente, etc. que los usuarios verán en la plataformas de distribución de paquetes.
- `docs/`: es la carpeta que contiene archivos de configuración y explicativos que permiten generar la documentación del proyecto. No obstante, el código que se expone públicamente se documenta automáticamente a partir de unos comentarios en el código.
- `pvlb/`: es la carpeta que contiene el código fuente de la librería, organizado en sub-carpetas y archivos según su funcionalidad. Los tests unitarios y de integración se encuentran en esta carpeta.
- `AUTHORS.md`, `CODE_OF_CONDUCT.md`, `codecov.yml`, `readthedocs.yml`, `paper/`, `ci/`, `benchmarks/` y `.github/`: son archivos y carpetas que contienen información adicional sobre el proyecto y la configuración de todos los flujos de integración continua: destacan los tests, los benchmarks y la documentación.

2.3.4. Frameworks de desarrollo del proyecto

Como tal, el desarrollo del proyecto requiere el conocimiento de algunas herramientas que permiten la colaboración con el repositorio principal y cumplir con las directrices para contribuir al proyecto.

En el lado de las utilidades esenciales, se encuentran:

- *Git*: como sistema de control de versiones que se emplea para gestionar los cambios en el código.

- *GitHub*: es la plataforma de alojamiento de proyectos de código abierto que se emplea para colaborar en el desarrollo del proyecto. Permite crear *issues* para reportar errores o sugerir mejoras, proponer cambios y revisarlos públicamente en *pull requests*. Además proporciona máquinas virtuales para ejecutar los procedimientos de integración y desarrollo continuos.
- *ReadTheDocs* junto con *sphinx*: es la plataforma que se emplea para ejecutar *sphinx*, el framework que produce la documentación en formato HTML, y alojar el producto. La documentación se escribe en formato *reStructuredText* y se genera automáticamente a partir de los comentarios en el código, scripts con los ejemplos y archivos fuente con el cuerpo más general.
- *pytest*: es la librería de Python que se emplea para escribir y ejecutar tests unitarios y de integración en el código. Facilita la creación de *mocks* u observadores del estado de partes del mismo proyecto, la reutilización de datos de tests y la ejecución opcional de los tests en función de las dependencias disponibles y visualizar los resultados en un informe. Posiblemente la mejor característica es poder ejecutar los tests uno a uno, en grupos o todos a la vez.
- *Codecov*: es la plataforma que se emplea para medir la cobertura de los tests en el código. Permite visualizar la cobertura de los tests en un informe y comprobar si se están realizando pruebas suficientes en el código.
- *flake8*: es una utilidad que detecta errores de estilo en el código, como la longitud de las líneas, la indentación, la presencia de espacios en blanco, entre otros. Facilita la escritura de código limpio y legible, por ende, mantenible.

Capítulo 3

Desarrollo

La primera parte de este capítulo describe el *modus operandi* del autor de este Trabajo Fin de Grado para contribuir a la librería de modelado fotovoltaico *pulib-python*, en el que se detallan las herramientas y el entorno de desarrollo empleados.

En el resto de secciones se tratarán las contribuciones de todo índole realizadas, abiertas o planificadas, agrupadas por la temática y la importancia. En aquellas más complejas, se describirán las bases teóricas y se contará el resultado de la propuesta, si es aceptado o no.

3.1. Entorno de desarrollo y herramientas utilizadas

En el caso del autor de este Trabajo Fin de Grado, se emplea el siguiente software para el desarrollo del proyecto consistentemente:

- *Visual Studio Code*: es el editor de código que se emplea para escribir y editar el código fuente del proyecto. Es un editor de código semi-abierto, ligero y rápido que cuenta con una amplia gama de extensiones para facilitar el desarrollo de software. Es una alternativa multipropósito a otros editores de código como *PyCharm* o *Spyder* en el caso de Python. Es importante denotar que la gran inmensa utilidad que proporciona *VSCode* es gracias a las extensiones creadas por la comunidad de desarrolladores, que permiten desde la edición de archivos de texto plano hasta la depuración de código, pasando por la integración con servicios de control de versiones y la ejecución de tests de forma visual. He aquí extensiones que se emplean en el desarrollo del proyecto:
 - *Python*: es la extensión que se emplea para el desarrollo de código en Python. Proporciona funcionalidades como el autocompletado de código, la visualización de la documentación de las funciones, la ejecución de scripts y la depuración de código.
 - *Ruff*: permite formatear el código según las directrices de estilo de *flake8*.
 - *GitHub Copilot*: un asistente de generación de código en línea integrado en el editor, que se emplea para sugerir fragmentos de código y documentación en función del contexto. Agiliza el desarrollo.

3.1. Entorno de desarrollo y herramientas utilizadas

- *Code Spell Checker*: es un corrector ortográfico que se emplea para detectar errores de ortografía en el código y en la documentación.
- *Jupyter Notebooks*: es la extensión que se emplea para editar y ejecutar *notebooks* de Jupyter en el editor, un formato que permite visualizar las variables del contexto y facilita el debugging interactivo. Los ejemplos de *pvl-lib-python* se realizan con un formato similar a las celdas de texto o código de una *notebook*.
- Resaltado de sintaxis de varios formatos de archivos, como *reStructuredText*, *Markdown*, *YAML* y *TOML*: para facilitar la edición de la documentación y los archivos de configuración.
- *GitHub Desktop*: es la interfaz gráfica que se emplea para colaborar en el desarrollo del proyecto. Permite visualizar los cambios, crear *ramas*, hacer *commits*, *pull requests* y *merges*, entre otros. En resumen, administra nuestra copia del repositorio para que los desarrolladores del proyecto puedan revisar y aceptar las propuestas.
- *pip* y *venv*: como gestor de paquetes y entornos aislados de desarrollo de Python nativos. Se emplean para instalar las dependencias del proyecto y usar entornos con las versiones específicas requeridas aislado del resto del sistema.

De forma discreta, se ha hecho uso de otras herramientas como:

- *Git*: para clonar las ramas del autor de este TFG y correr partes de la integración continua, normalmente la documentación, en las máquinas virtuales de linux provistas por la UPM¹.
- *Miniconda*: una distribución de Python que se emplea para instalar y gestionar las dependencias del proyecto. Facilita la creación de entornos virtuales y la instalación de paquetes de Python. Se utilizó para diagnosticar un error de precisión por la compilación de algunas librerías y que hacía fallar un test.
- *LibreOffice Calc*: para generar datos de prueba y comprobar las implementaciones de las ecuaciones de los modelos.

El desarrollo de la librería se ha hecho habitualmente en Windows 10, en un portátil HP *15-dw2003ns* así que se ha aprovechado a usar el subsistema de Windows para Linux (WSL) para ejecutar tests y construir la documentación en un entorno similar al de integración continua. No obstante, por los recursos limitados del portátil, y por poder cambiar de ramas mientras se construye la documentación, se ha hecho uso de las máquinas virtuales de la UPM para ejecutar los tests y construir la documentación.

3.1.1. La documentación

En esta sección pondremos en valor lo que realmente es lo más importante de un proyecto de programación, en especial de código abierto: la documentación. Es más, se puede argumentar el valor de todas las aportaciones es documentar lo más rigurosamente posible los modelos y métodos que se implementan, tanto para dar a conocerlos como para que el uso sea correcto.

¹Accesible a través de <https://escritorio.upm.es/>.

Como se comentaba anteriormente, la documentación de la librería *pulib-python* se encuentra alojada en la plataforma *ReadTheDocs*, que expone los archivos HTML para que los usuarios puedan consultarla en línea. La documentación se genera automáticamente a partir de los comentarios en el código fuente, que se escriben en formato *reStructuredText* y se construye con el framework *sphinx*.

Existen archivos específicos para indicar qué funciones o métodos son públicos, hacer páginas de inicio, de referencia, de ejemplos, de instalación, de contribución, etc. Además, se pueden incluir imágenes, tablas, gráficos, enlaces, referencias, entre otros elementos que facilitan la comprensión de los conceptos.

sphinx emplea el estilo de documentación de *pydata-sphinx-theme*, que organiza las secciones de la documentación y da un estilo homogéneo a la web. Además, para la creación de los ejemplos se emplea la extensión *sphinx-gallery*, que ejecuta unos scripts de Python por secciones similares a una *notebook* de Jupyter y captura la salida de texto estándar y los gráficos para mostrarlos en la documentación.

La *docstring* de cada función, que es el comentario que se escribe en la primera línea de la definición de la función y se emplea para autogenerar la documentación, utiliza el estilo de *numpydoc*. Este estilo permite incluir información sobre los parámetros de entrada, los valores de retorno, las excepciones que se pueden lanzar, entre otros. Además, se pueden incluir ejemplos de uso de la función e informar avisos de precaución sobre aspectos más específicos.

A continuación, una plantilla de ejemplo de documentación y de código de una función cualquiera:

Listing 3.1: Ejemplo de documentación de una función en *pulib-python*.

```
1 def example_model(param1, param2):
2     """
3     Brief model description.
4
5     Long model description, also found at [1]_.
6
7     .. versionadded:: 0.1.0
8
9     .. warning::
10         This docstring is an example.
11
12     Parameters
13     -----
14     param1 : numeric
15         Description of the parameter.
16     param2 : numeric
17         Description of the parameter.
18
19     Returns
20     -----
21     float
22         Return type of the function.
23
24     Notes
25     ----
26     Additional notes about the function, detailed explanations if needed. Even an equation:
27
28     .. math::
29
30         f(x, y) = x + y
31
32     Examples
33     -----
```

3.1. Entorno de desarrollo y herramientas utilizadas

```
34 >>> example_model(1, 5)
35 6.0
36
37 References
38 -----
39 .. [1] Author, A. (2024). Title of the paper. Journal, 1(1), 1-10. :doi:`10.0001/populate`
40 " " "
41 return float(param1 + param2)
```

Esta función, una vez listada en el archivo correspondiente del índice que nos interese -aquí usamos el submódulo `pvlb.solarposition` como ejemplo-, creará una página como la que se muestra en la figura 3.1:

The screenshot shows the documentation page for `pvlb.example_model`. The page layout includes a sidebar on the left with a search bar and a list of classes under the 'Solar Position' section. The main content area displays the function signature `pvlb.example_model(param1, param2)`, a brief description, a long description, a 'New in version 0.1.0' notice, a 'Warning' box stating 'This docstring is an example.', parameters, returns, notes, examples, and references. The examples section shows a code snippet: `>>> example_model(1, 5)` followed by `6.0`. The references section lists a single reference: [1] Author, A. (2024). Title of the paper. Journal, 1(1), 1-10. DOI: 10.0001/populate. Below the references, there is a section titled 'Examples using pvlb.example_model' which contains a small plot and the text 'Example title'. The footer of the page contains copyright information: © Copyright 2013-2021, Sandia National Laboratories and pvlb python Development Team. Created using Sphinx 4.5.0.

Figura 3.1: Un ejemplo de renderizado de la documentación de una función en *pvlb-python*.

Por otro lado, un ejemplo cuenta con la siguiente estructura:

Listing 3.2: Plantilla para elaborar un ejemplo en *pvlb-python*.


```
1 " " "
2 Example title
```

Desarrollo





```
3 =====
4
5 Brief model description (shown in preview card).
6 """
7
8 # %%
9 # Text paragraph, in reStructuredText format. Can use sections, subsections, etc., and math as
   in LaTeX.
10 # More text.
11
12 # This is a comment (there is a newline above)
13 import matplotlib.pyplot as plt
14 from pvlib import example_model
15 print("Hello, world!")
16 plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
17 plt.show()
18
19 # %%
20 # Return to paragraph text.
21
22 sum_val = example_model(1, 5)
23 print(f"Sum of 1 and 5 is {sum_val}.")
```

Este archivo, ubicado en la carpeta correcta (aquí el archivo es `docs\examples\solar-position\example_example.py`), hará que se cree automáticamente una página como la que se muestra en la figura 3.2.

3.1. Entorno de desarrollo y herramientas utilizadas



User Guide **Example Gallery** API reference What's New Contributing



Q Search the docs ...

ADR Model for PV Module Efficiency
Agrivoltaic Systems Modelling
Bifacial Modeling
Irradiance Decomposition
Irradiance Transposition
I-V Modeling
Reflections
Shading
Soiling
Solar Position
 Example title
 Sun path diagram

Note

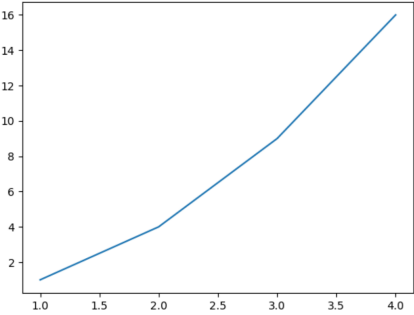
Go to the end to download the full example code.

Example title

Brief model description (shown in preview card).

Text paragraph, in reStructuredText format. Can use sections, subsections, etc., and math as in LaTeX. More text.

```
# This is a comment (there is a newline above)
import matplotlib.pyplot as plt
from pvlib import example_model
print("Hello, world!")
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



Out: Hello, world!

Return to paragraph text.

```
sum_val = example_model(1, 5)
print(f"Sum of 1 and 5 is {sum_val}.")
```

Out: Sum of 1 and 5 is 6.0.

Total running time of the script: (0 minutes 0.060 seconds)

Download Jupyter notebook: [example_example.ipynb](#)

Download Python source code: [example_example.py](#)

Gallery generated by Sphinx-Gallery

Previous

[Solar Position](#)

Next

[Sun path diagram](#)

© Copyright 2013-2021. Sandia National Laboratories and pvlib python Development Team.
Created using Sphinx 4.5.0.

Figura 3.2: Un ejemplo de página de ejemplo en la documentación de *pvlib-python*.

A continuación se muestran los comandos que deben ejecutarse para construir la documentación:

- En un sistema basado en *Linux* o *WSL*.
- Clonando el repositorio original de la librería con *Git*.
- Empleando las mismas versiones que a día de la redacción de este documento se emplea en la integración continua (*pvlib-python=0.11.0*) - en especial hay que hacer la instalación de Python3.8.
- Con el entorno aislado de desarrollo (*venv*) activado.

18

- Y realizando una instalación local mediante *pip*.

Listing 3.3: Comandos para construir la documentación de *pvlb-python*.

```
1 # instalar Python 3.8 desde el repositorio de deadsnakes,
2 # en las librerías por defecto de Ubuntu no se encuentra disponible por antigüedad
3 sudo add-apt-repository ppa:deadsnakes/ppa -y
4 sudo apt-get update
5 sudo apt-get install python3.8 python3.8-venv -y
6
7 # clonar el repositorio de pvlb-python
8 git clone https://github.com/pvlb/pvlb-python
9 cd pvlb-python
10
11 # crear el entorno virtual, instalar pvlb y las dependencias de la documentación
12 python3.8 -m venv .venv
13 source .venv/bin/activate
14 python3.8 -m pip install .[doc]
15
16 # construir la documentación
17 cd docs/sphinx
18 make html
19
20 # abrir la documentación en el navegador
21 xdg-open build/html/index.html
```

3.2. Contribuciones científicas

3.2.1. Modelado de ajuste espectral

- *Pull Request*: #1658

El desarrollo inició en Septiembre de 2022, en Febrero de 2023 se planteó la *pull request*, que finalmente se cerró en Mayo de 2023 sin incluirse los cambios en la librería.

3.2.1.1. Fundamento teórico

El modelo [15] plantea una relación entre la efectividad bajo un espectro estándar y la efectividad bajo un espectro arbitrario caracterizado por la masa de aire y el índice de claridad. Al principio de la implementación, tanto el autor de este TFG como el mentor del proyecto, César Domínguez, pensaron que se trataba de un modelo de ajuste similar a otros en la literatura que corrigen la irradiancia incidente, para dar lugar a la efectiva. Ejemplos de estos modelos ya se encontraban en la librería, como el modelo desarrollado por la empresa *First Solar* descrito en [16] o el de Caballero et al. en [17]. Además, para la versión 0.11.0 de la librería un compañero del programa *Google Summer of Code* implementó un par de modelos más que funcionan de la misma forma. Los modelos en cuestión crean un factor de ajuste M que se define, según el estándar IEC 60904-7 [17, Eq. (2)]:

$$M = \frac{\int_{\lambda_1}^{\lambda_2} E(\lambda) SR(\lambda) d\lambda \int_{\lambda_3}^{\lambda_4} E^*(\lambda) d\lambda}{\int_{\lambda_1}^{\lambda_2} E^*(\lambda) SR(\lambda) d\lambda \int_{\lambda_3}^{\lambda_4} E(\lambda) d\lambda} \quad (3.1)$$

Resultó que, tras un elevado esfuerzo que llevó meses, el modelo en [15] no se ajustaba a esta descripción, sino que planteaba la siguiente ecuación:

$$M = \frac{S_{efE(\lambda)}}{S_{ef\bar{G}(\lambda)}} \quad (3.2)$$

Sin embargo, era la fracción de las tres relaciones que se planteaban en la tesis doctoral de Nuria Martín Chivelet [18], que es:

$$PS = 1 - \frac{S_{efE(\lambda)}}{S_{ef\bar{G}(\lambda)}} \frac{E_{\lambda < \lambda_0}}{\bar{G}_{\lambda < \lambda_0}} \frac{\bar{G}}{E} \quad (3.3)$$

Se puede observar que la ecuación 3.2 solo contempla una parte de la definición que aplica del ajuste espectral, por ende, invalida la aplicación que se esperaba del modelo de dicho artículo.

Es gracias a los tutores que se identifica este error tras su solicitud de compararlo con modelos ya existentes.

3.2.1.2. Resultado

Se descarta aplicar los procedimientos descritos en su tesis [18] por:

- La ausencia de acceder al documento de forma online.
- La ausencia de una versión en inglés, que pueda servir como referencia.
- La particularidad de los datos sobre los que se hace el modelo.

Lamentablemente, incluso tras contactar presencialmente con la autora y habiendo disfrutado tanto de una explicación detallada de su modelo científico como de la posibilidad de continuar en esa misma línea de trabajo, y contando con una copia física de su tesis, se desestima continuar en esa línea de trabajo debido a las razones anteriores.

Se cierra la propuesta tres meses después de plantearla.

3.2.2. Proyección del cenit solar sobre las coordenadas de un colector

- *Issue*: #1734
- *Pull Request*: #1904

Esta es la primera de una trilogía de contribuciones que se plantean para aplicar un modelo de pérdidas por sombreado en módulos con diodos bypass, cuyo autores pertenecen a la Escuela Técnica Superior de Ingeniería y Diseño Industrial, *ETSIDI*. El objetivo de esta primera contribución es calcular la proyección del cenit solar sobre las coordenadas de un colector, que se emplea para calcular la fracción de sombra unidimensional en geometrías de paneles que comparten eje de rotación en común.

Las otras dos contribuciones se detallan en 3.2.3 y en 3.2.4.

Esta funcionalidad ya existía como parte de alguna función de la librería, así que el aporte consistió en rehacerla de nuevo empleando una referencia bibliográfica y contrastando las implementaciones. Por supuesto, la documentación se cobró la parte más importante del tiempo de desarrollo.

3.2.2.1. Fundamento teórico

Dos cálculos de bastante interés en geometría solar es obtener los ángulos óptimos de seguimiento para un colector y calcular la fracción de sombra unidimensional incidente. Ambos cálculos tienen en común un paso muy importante, que es saber con qué ángulo inciden los rayos directos del Sol sobre la superficie del colector, pero referenciado al plano de rotación del mismo. Para facilitar el entendimiento de este concepto, es preferible imaginar un sistema de rotación de un solo eje y considerar que un colector fijo es un caso particular de un seguidor uniaxial.

Nos encontramos estos dos casos de uso:

- Para el cálculo de los ángulos óptimos de seguimiento en seguidores de un solo eje, asumiendo que interesa seguir al Sol en su trayectoria diaria, se debe conocer el ángulo de incidencia de los rayos solares sobre la superficie del colector en el plano de rotación del mismo. Es decir, proyectar el cenit solar en el plano perpendicular al eje de rotación del colector, que es aquel que contiene todos los vectores normales al plano del colector.
- En el caso de la fracción unidimensional de sombra, interesa saber en donde impactan los límites del colector frontal sobre el trasero. Para ello, se proyecta el cenit solar en el plano perpendicular al eje de rotación del colector, que es aquel que contiene los dos vectores normales a los planos de los colectores.

Se puede encontrar en más detalle en el artículo de Lorenzo, Narvarte y Muñoz en [19], personal de la *ETSIDI*, a partir del cual continúa el trabajo de [20], sección sobre *True-Tracking Angle*. Esta última referencia es ampliamente utilizada a lo largo del repositorio.

3.2.2.2. Resultado

Después de cincuenta y tres comentarios en la propuesta, finalmente se incluyen los cambios como parte de la librería en la versión 0.10.4.

Accesible en `pvlib.shading.projected_solar_zenith_angle`.

3.2.3. Cálculo de fracción de sombra unidimensional

- *Issue*: #1689
- *Pull Request*: #1962

Esta propuesta es la segunda de la trilogía de propuestas para poder aplicar un modelo de pérdidas por sombreado. Continúa con la propuesta anterior, 3.2.2, y se encarga de calcular la fracción de sombra unidimensional en paneles con determinadas geometrías. La última propuesta de la trilogía es 3.2.4.

Aquí se plantea la aplicación de un modelo para conocer la fracción de sombra unidimensional en paneles que comparten eje de rotación en común. La implementación se basa en [21]. No obstante, la implementación inició con un póster de una conferencia anterior, a partir de una propuesta de cambios de un trabajador de *First Solar* en la librería *pvlib-python*, que se puede consultar en <https://github.com/pvlib/pvlib-python/pull/1725>.

3.2.3.1. Fundamento teórico

El modelo propuesto en [21] parte de un diseño de dos colectores que comparten la misma dirección del eje, y uno se encuentra más cercano al Sol que el otro. Lo interesante de este diseño es que tiene en cuenta múltiples variables de diseño, como la pendiente, la separación entre el eje y el plano colector, e inclinaciones distintas del colector sombreado y el que sombrea.

Se requiere conocer el ángulo proyectado del cenit. A partir de este ángulo, mediante intersección de rectas, se puede conocer la fracción de sombra unidimensional. Realmente se trata de una función muy compleja por el número de entradas que tiene, pues adicionalmente el cálculo de esta proyección se hace internamente en la función para simplificar la API - es decir, la interfaz programática que usarán los usuarios puede resultar densa.

Se llama fracción de sombra **unidimensional** porque se mide la fracción de sombra a lo largo de la línea del avance para una misma azimuth pero distinta elevación solar. Normalmente este valor es el que dota de mayor información, pues los cambios de la sombra debido a la azimuth no cobran tanta importancia en la mayoría de ubicaciones terrestres. Son aquellas más cercanas a los polos los que más se ven afectados por la cambiante azimuth. A continuación, la imagen 3.3 muestra un esquema del sistema y su parte sombreada:

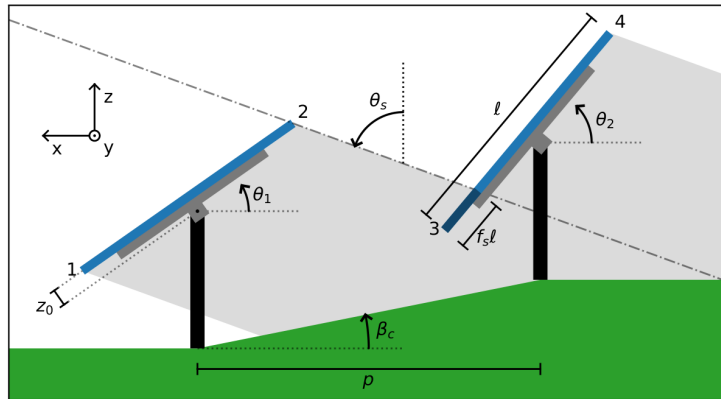


Figura 3.3: Esquema dos colectores parametrizados, donde uno sombrea al otro. La nomenclatura corresponde la ecuación 3.4.

Fuente: Figura 3 en [21].

Las ecuaciones relevantes implementadas son (12) y (13) de [21], 3.4 y 3.5 respectivamente en este documento:

$$\begin{aligned}
 t^* = & \frac{1}{2} \left(1 + \left| \frac{\cos(\theta_1 - \theta_s)}{\cos(\theta_2 - \theta_s)} \right| \right) \\
 & + \operatorname{sgn}(\theta_s) \frac{z_0}{\ell} \left(\frac{\sin(\theta_2 - \theta_s) - \sin(\theta_1 - \theta_s)}{|\cos(\theta_2 - \theta_s)|} \right) \\
 & - \frac{p}{\ell} \left(\frac{\cos(\theta_s - \beta_c)}{|\cos(\theta_2 - \theta_s)| \cos(\beta_c)} \right)
 \end{aligned} \tag{3.4}$$

$$f_s = \begin{cases} 0 & \text{si } t^* < 0 \\ t^* & 0 \leq t^* \leq 1 \\ 1 & \text{si } t^* > 1 \end{cases} \quad (3.5)$$

3.2.3.2. Resultado

Después de 102 comentarios en la propuesta, finalmente se incluyen los cambios como parte de la librería en la versión 0.11.0, junto con un ejemplo ilustrativo que tiene en cuenta posibles dificultades que pueda encontrar un usuario².

Accesible en `pvlib.shading.shaded_fraction1d`.

Además, como anécdota se ha de mencionar que durante el desarrollo de esta propuesta se identifica la ausencia de ejemplos que planteasen módulos orientados al Norte (propio de las instalaciones en el hemisferio Sur) en todo el repositorio, por lo que se incluye un ejemplo de este tipo en la documentación. Se puede atribuir el mérito a Rubén Nuñez por hablar al estudiante del sesgo a favor del hemisferio norte en el mundo de la fotovoltaica.

3.2.4. Pérdidas por sombreado en módulos con diodos de bypass

- *Issue*: #2063
- *Pull Request*: #2070

Con esta propuesta finaliza la trilogía de contribuciones del modelo de pérdidas por sombreado en módulos con diodos de bypass. Las dos propuestas anteriores son descritas en 3.2.2 y en 3.2.3.

La propuesta es de un modelo realizado por personal de la *ETSIDI* y se encarga de calcular las pérdidas por sombreado en módulos con diodos de bypass. La implementación se basa en el trabajo de Martínez-Moreno, F., Muñoz, J. y Lorenzo, E., en [22].

3.2.4.1. Fundamento teórico

Los módulos de paneles solares de silicio se conforman de múltiples células fotovoltaicas. Una célula, cuando es irradiada por la luz solar, genera una corriente eléctrica. Si conectamos todas las células en serie, la corriente generada por todas las células es la misma, pero la tensión generada por cada célula se suma. Si una célula se sombrea, la corriente generada por ella disminuye, y por tanto la corriente generada por el conjunto disminuye. Esta célula sombreada se comporta como una carga para el resto, pues deben forzar la corriente que pasa por este elemento (que se suele modelar como un diodo). Supone un riesgo de seguridad y de degradación, pues una célula sombreada puede calentarse excesivamente y dañar las capas materiales de su módulo.

La solución que se emplea industrialmente consiste en añadir *diodos de bypass*. Estos permiten que, cuando una serie de células tiene sombra, la corriente mayori-

²Véase en https://pvlib-python.readthedocs.io/en/latest/gallery/shading/plot_shaded_fraction1d_ns_hsat_example.html.

taria generada por el resto de células bien iluminadas fluya a través de este diodo, y no de la célula sombreada. De esta forma, se protege frente al sobrecalentamiento y la degradación temprana.

Ha de hacerse énfasis en que un diodo protege varias células, ya que no es rentable económicamente poner un diodo por cada célula. El planteamiento que nos encontramos en [22] consiste en, a partir del número de grupos de células protegidos por un diodo que se encuentran sombreados, asumir que la tensión de este grupo es nula por tener un diodo en conducción y cancelar la potencia que generaría.

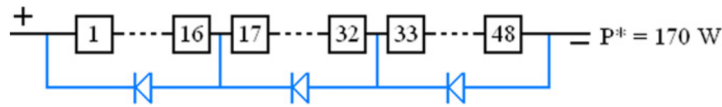


Figura 3.4: Esquema de un módulo con 3 diodos de bypass.

En la imagen 3.4, si suponemos que la célula número 1 está sombreada, la corriente mayoritaria generada por los grupos 17 a 32 y 33 a 48 fluye a través del diodo que está en paralelo con las células 1 a 16. No importa en este caso la corriente del grupo 1 a 16, pues la tensión de este grupo es nula por tener un diodo en conducción.

Nótese que estos grupos se definen en [22] como *bloques*, y un bloque está sombreado en cuanto una de sus células recibe una fracción infinitesimal de sombra.

Lo más complicado de esta contribución es explicar en detalle cómo identificar el número de bloques y su disposición en el módulo, pues existen varias posibilidades. Hay módulos lo suficientemente pequeños para que solo tengan un diodo, los hay con dos y con tres diodos, y los hay *half-cut* que también tienen 3 diodos, pero con una disposición que crea 6 bloques en vez de 3. Además, la progresión de los bloques que se va sombreando depende del sistema y la geometría de las sombras.

El resultado de este modelo establece la cantidad de potencia que se perdería respecto de las mismas condiciones sin sombra, $SL = 1 - \frac{P_{\text{sombreado}}}{P_{\text{no sombreado}}}$. Además, anular la potencia de un bloque sombreado se hace sobre la componente directa de la irradiancia, que es la que normalmente genera las sombras, pues la componente difusa sigue impactando en las células sombreadas y creando un mínima parte de aporte energético.

La expresión de pérdidas de potencia es la ecuación 3.6, Eqs. [6] y [8] en [22].

$$(1 - F_{ES}) = (1 - F_{GS}) \left(1 - \frac{N_{SB}}{N_{TB} + 1} \right) \quad (6)$$

$$\left(1 - \frac{P_S}{P_{NS}} \right) = \left(1 - \frac{[(B + D^{CIR})(1 - F_{ES}) + D^{ISO} + R]}{G} \right) \quad (8) \quad (3.6)$$

3.2.4.2. Resultado

Tras unos 85 comentarios, finalmente se incluyen los cambios como parte de la librería en la versión 0.11.0, junto con un ejemplo de caso de uso³.

Accesible en `pvlib.shading.direct_martinez`.

3.2.5. Fracción diaria de radiación difusa fotosintetizable en función de la fracción difusa global

- *Issue*: #2047
- *Pull Request*: #2048

Esta contribución se trata de un pequeño modelo que abre un nuevo tema en la librería: la agrivoltaica. La agrivoltaica es una técnica en la que coexisten la producción de energía solar y la producción agrícola en un mismo terreno. La ventaja es que un exceso de irradiancia puede ser perjudicial para la plantación, y el retorno económico del terreno puede ser mayor que si solo se dedicase a la producción de energía solar. Además, contribuye positivamente en la polémica de utilizar suelo exclusivamente para la producción de energía.

El modelo en cuestión es la continuación de un trabajo realizado por Spitters C. J. T. et al. que cubre la separación de irradiación en directa y difusa en general [23], y que posteriormente él desglosa en dos expresiones en [24], que se puede sustituir una en la otra. El objetivo de este modelado que hace es calcular la fracción de irradiación difusa que es útil para las plantas -es decir, fotosintetizable- a partir de la fracción de irradiación difusa global.

3.2.5.1. Fundamento teórico

La irradiación difusa fotosintetizable (*PAR*, por sus siglas en inglés), es la radiación difusa que se encuentra en el rango de longitudes de onda que las plantas pueden absorber y utilizar para la fotosíntesis. Es interesante de cara a simular el crecimiento y producción de las plantas, y se emplea en modelos de cultivo. Esto último queda fuera del alcance de la librería, pero se plantea como un paso en motivar y facilitar el diseño de sistemas agrivoltaicos.

Se utiliza irradiación diaria [$J/m^2/dia$], en vez del valor instantáneo irradiancia propio de la simulación de sistemas fotovoltaicos [W/m^2].

Un análisis de distintos modelos y la validación de los mismos que contiene la expresión única se puede encontrar en [25]. Este artículo es del cual origina la propuesta de contribución.

3.2.5.2. Resultado

Con la mayor complicación de discernir las unidades de entrada, finalmente se incluyen los cambios como parte de la librería en la versión 0.11.0, junto con un

³Véase en https://pvlib-python.readthedocs.io/en/latest/gallery/shading/plot_martinez_shade_loss.html.

ejemplo de uso⁴.

Accesible en `pvlib.irradiance.diffuse_par_spitters`.

3.2.6. Modelo de pérdidas por heterogeneidad de irradiancia por célula

- *Issue*: #1541
- *Pull Request*: #2046

Esta contribución implementa un modelo de pérdidas sobre la potencia de salida para módulos bifaciales, es decir, aquellos que pueden recibir irradiación solar tanto por una cara delantera como por la trasera. El modelo se aplica para tener en cuenta irradiancia que no es homogénea en la superficie del módulo. Se trata del trabajo descrito en [26]. Presenta interés en sistemas bifaciales, donde la cara trasera normalmente se expone a la luz reflejada por el suelo y otras obstrucciones, y por tanto la irradiancia de esa cara no es homogénea.

3.2.6.1. Fundamento teórico

Anteriormente se explicaba el mecanismo de interconexión de células solares fotovoltaicas, y cómo una célula sombreada puede actuar como una carga para el resto de células. De forma similar ocurre a pequeña escala, cuando una o varias células reciben valores de irradiancia ligeramente distintos al resto. La célula que recibe menos irradiancia limita la corriente, y la potencia de salida del módulo se ve reducida.

Desde un punto de vista computacional, resolver un sistema de múltiples células, cada una con su propia irradiancia, es realmente costoso en tiempo y en recursos. Cada célula tendría su propia curva I-V, que representa cuanta corriente y tensión genera dependiendo del punto de tensión de trabajo. El planteamiento que se hace en [26] es realizar un trabajo previo de caracterización de la heterogeneidad, cuantificarla y establecer un modelo de menor orden de complejidad.

Para caracterizar distribuciones de irradiancia, en el artículo de Deline et al. [26] se plantea utilizar la desviación estándar, muy común para distribuciones normales, o la *Diferencia Absoluta Media Relativa (RMAD*, por sus siglas en inglés), que es una medida de dispersión que se argumenta ser más adecuada para distribuciones no normales [27].

La pérdida de potencia de salida M se calcula con un polinomio evaluado en *RMAD*:

$$M = 1 - \frac{P_{\text{array}}}{\sum P_{\text{cells}}} \quad (3.7)$$

donde P_{array} es la potencia de salida del módulo, y P_{cells} es la potencia máxima de salida de cada célula.

Se proponen dos modelos para el polinomio que define M , para dos perfiles de irradiancias globales distintos: uno para sistemas sujetos fijos y otro para seguidores

⁴Véase en https://pvlib-python.readthedocs.io/en/latest/gallery/agrivoltaics/plot_diffuse_PAR_Spitters_relationship.html.

de un eje. No obstante, solo se implementa el de sistemas fijos ya que la referencia indica que para valores anuales parece ser más preciso.

3.2.6.2. Resultado

Tras una ardua discusión de 86 comentarios en los que se plantean distintas formas de implementar el modelo, aclaraciones sobre las unidades de entrada y salida, y modificaciones al planteamiento original, finalmente se incluyen los cambios como parte de la librería en la versión 0.11.1 (sin publicar oficialmente a día de la redacción de este documento), junto con un ejemplo de uso⁵.

Accesible en `pvlib.bifacial.power_mismatch_deline`.

Los autores originales del modelo científico piden contrastar la implementación con la suya, y se valora hacerlo en un futuro próximo antes de la siguiente versión de la librería. Se plantea como tarea futura.

3.2.7. Transformación de respuesta espectral a eficiencia cuántica externa y viceversa

- *Issue*: #2040
- *Pull Request*: #2041

Esta contribución podría calificarse de menor debido a la ausencia de dificultades en su implementación. No obstante, por dotar de una nueva funcionalidad científica a la librería, se incluye en este apartado.

La propuesta consiste en dos funciones, una inversa de la otra, que convierten la respuesta espectral y la eficiencia cuántica externa.

3.2.7.1. Fundamento teórico

La eficiencia cuántica externa es una medida de la eficiencia de una célula solar, y se define como la razón de fotones incidentes que generan una corriente eléctrica para determinado color de la luz. La respuesta espectral es la corriente generada por una célula solar en función de la longitud de onda de la luz incidente.

La relación entre ambas es la siguiente [28, pp. 15-16, Eq. (7)]:

$$SR(\lambda) = \frac{q \cdot \lambda}{h \cdot c} \cdot EQE(\lambda) \quad (3.8)$$

3.2.7.2. Resultado

Además de la relación, se añade la posibilidad de normalizar los valores de salida, es decir, hacer que el máximo retornado sea 1. Se incluyen los cambios sin mayores dificultades en la versión 0.11.0 de la librería.

Accesibles en:

- `pvlib.spectrum.qe_to_sr`.

⁵Véase en https://pvlib-python.readthedocs.io/en/latest/gallery/bifacial/plot_irradiance_nonuniformity_loss.html.

- `pvlib.spectrum.sr_to_ge`.

3.2.8. Adición de base de datos de respuesta espectral de algunas tecnologías

- *Issue*: #2037
- *Pull Request*: #2038

Con esta propuesta se pretendía añadir una serie de respuestas espectrales o de eficiencia cuántica externa de células solares de distintas tecnologías comunes, para facilitar la investigación.

3.2.8.1. Fundamento teórico

Una curva de respuesta espectral indica la capacidad que tiene un semiconductor fotovoltaico en convertir la luz incidente en corriente eléctrica en función de la longitud de onda. La eficiencia cuántica externa similarmente es una medida de la eficiencia de una célula solar para convertir fotones en pares electrón-hueco.

Dependiendo de la tecnología de cada material, la respuesta varía. Además, se puede argumentar que otros aspectos constructivos también afectan, como los espesores de las capas y la presencia de impurezas.

3.2.8.2. Resultado

La propuesta no se llegó a añadir a la librería, ya que los datos provistos de un repositorio público (en Duramat⁶) no estaban respaldados por un procedimiento que garantizase que los datos fueran representativos. Se cierra la propuesta.

3.2.9. Adición de espectro estándar completo ASTM G173-03

- *Issue*: #2039
- *Pull Request*: #1963

Esta adición plantea añadir las componentes de irradiancia directa y extraterrestre del estándar ASTM G173-03, que es un espectro de referencia para la radiación solar en la superficie terrestre. La componente global ya se encontraba en la librería. Se aprovecha para añadir flexibilidad y prevenir la adición de nuevos estándares, como el similar ASTM G173-23 en un futuro.

3.2.9.1. Fundamento teórico

Un espectro estándar es de gran utilidad para comparar módulos y establecer métodos idénticos de tomar las medidas en la industria. En el caso del espectro estándar ASTM G173-03, se establecen una serie de puntos entre los 280 nm y los 4000 nm que simulan una distribución espectral bastante plausible.

La irradiancia extraterrestre es la irradiancia que se recibe en el espacio, y la irradiancia directa es la irradiancia que se recibe en la superficie terrestre sin ser

⁶Véase <https://www.osti.gov/biblio/2204677>.

dispersada por la atmósfera, y la global es toda la que se recibe en la superficie terrestre.

El estándar ASTM G173-03 se encuentra en [29], pero los valores del espectro están disponibles abiertamente en <https://www.nrel.gov/grid/solar-resource/spectra-am1.5.html>. La nueva revisión ASTM G173-23 se encuentra en [30], pero no cuenta con datos abiertos en la red a día de la redacción de este documento.

3.2.9.2. Resultado

Se implementa esta función más genérica, y se hace obsoleta la antigua función (`pvlib.spectrum.get_am15g`) que solo devolvía la componente global. Se incluyen los cambios en la versión 0.11.0 de la librería, junto con un ejemplo en la documentación⁷. El ejemplo se puede encontrar en la figura 3.5.

⁷Véase en https://pvlib-python.readthedocs.io/en/stable/gallery/spectrum/plot_standard_ASTM_G173-03.html.

ASTM G173-03 Standard Spectrum

This example demonstrates how to read the data from the ASTM G173-03 standard spectrum bundled with pvlib and plot each of the components.

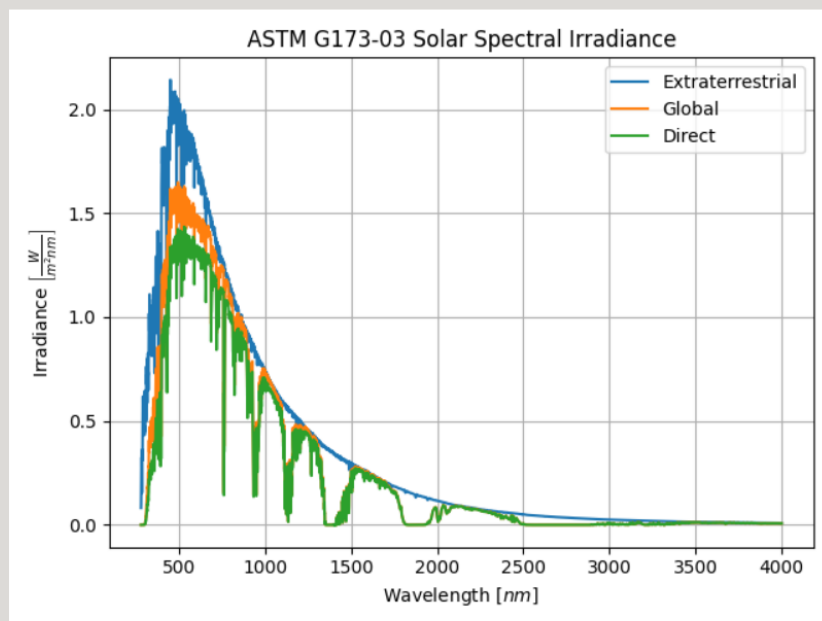
The ASTM G173-03 standard provides reference solar spectral irradiance data.

```
import matplotlib.pyplot as plt
import pvlib
```

Use `pvlib.spectrum.get_reference_spectra()` to read a spectra dataset bundled with pvlib.

```
am15 = pvlib.spectrum.get_reference_spectra(standard="ASTM G173-03")

# Plot
plt.plot(am15.index, am15["extraterrestrial"], label="Extraterrestrial")
plt.plot(am15.index, am15["global"], label="Global")
plt.plot(am15.index, am15["direct"], label="Direct")
plt.xlabel(r"Wavelength [nm]")
plt.ylabel(r"Irradiance $\left[\frac{W}{m^2 nm}\right]$")
plt.title("ASTM G173-03 Solar Spectral Irradiance")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.325 seconds)

Download Jupyter notebook: [plot_standard_ASTM_G173-03.ipynb](#)

Download Python source code: [plot_standard_ASTM_G173-03.py](#)

Gallery generated by Sphinx-Gallery

Figura 3.5: Ejemplo en la documentación con el espectro estándar ASTM G173-03 completo.

Accesible en `pvlib.spectrum.get_reference_spectra`.

3.2.10. Cálculo geométrico de sombras en 3D

- *Issue*: #2069
- *Pull Request*: #2106

Se ha podido comprobar que el cálculo de sombras en sistemas fotovoltaicos dentro de *pvl-lib-python* es un tema que se ha mejorado a lo largo de este TFG. Adicionalmente, se ha podido comprobar que la librería no cuenta con una función que permita calcular sombras a partir de escenas en 3D, pero realmente parece ser un tema sobre el que hay literatura. Además, *PVsyst*, un software de simulación de sistemas fotovoltaicos, cuenta con esta funcionalidad.

Todo esto se puede explorar en un artículo sobre el sombreado de campos de cultivo en sistemas agrivoltaicos [31].

La propuesta que aquí se hace realiza una generalización del cálculo de sombras a superficies limitadas y libres en el espacio 3D.

Es importante denotar que esta propuesta está pendiente de revisión y aprobación, en especial de saber si merecerá o no la pena incluirla en la librería por la complejidad que añade y algunas modificaciones al procedimiento del artículo original.

3.2.10.1. Fundamento teórico

Para esta contribución es imprescindible tratar algunos temas de geometría y cálculo vectorial:

- Definición de una recta a partir de un punto y un vector.
- Intersección de una recta con un plano.
- Traslación de puntos.
- Rotación de puntos respecto del origen, mediante matrices de rotación o representación de ángulos de Euler.
- Limitar superficies a determinadas coordenadas de otro plano.

No se ahonda en estos detalles ya que para realizar una contribución de esta índole se requiere conocer otras utilidades que nos brinden las librerías de cálculo científico en Python, como *NumPy*, *SciPy* y *Shapely*. La primera es muy conocida por permitir el cálculo con vectores, la segunda por facilitar múltiples tipos de cálculos matemáticos y la tercera por permitir cálculos geométricos y geoespaciales. No obstante, la intersección plano-recta y la traslación de puntos son operaciones que se pueden realizar muy fácilmente con los operadores de Python.

Lo primero que necesita un flujo de trabajo de este tipo es definir coordenadas para los objetos de la escena y sus límites. Para ello, habrá de establecerse un sistema de referencia. En el caso de la propuesta realizada, se opta por cambiar el sistema de coordenadas propuesto en [31] por el de [20], que es más conocido y utilizado en la librería.

Posteriormente, una vez creadas las superficies, tanto las sombreadas como las que generan sombras, se debe calcular el vector de posición solar. Este vector se

calcula a partir de la posición del Sol en el cielo, que se puede obtener con la función `pvlib.solarposition.get_solarposition` y con relaciones trigonométricas.

A continuación, se proyectan los vértices de las superficies que sombrean sobre el plano sombreado, y estos puntos definirán una sombra en la escena 3D. Nótese por tanto que esta sombra puede y debe tener tres coordenadas.

Para obtener la sombra 3D final, debe limitarse los límites de esta a la superficie de interés. Para esto se emplea *Shapely*, que permite realizar operaciones geométricas con polígonos.

Si se deseara obtener la sombra en un plano 2D, para realizar otros cálculos y facilitar la visualización, se debe realizar una traslación que ubique la figura en un plano que pase por el origen y unas rotaciones contrarias a las que definen el plano sobre la que se proyectó.

3.2.10.2. Resultado

Ahondando en los detalles, el autor de este mismo TFG decide crear un paradigma orientado a objetos para facilitar el uso de esta funcionalidad. Se plantean dos objetos, uno base para cualquier superficie poligonal (`FlatSurface`) y una especialización para superficies rectangulares, que son las más comunes en sistemas fotovoltaicos (`RectangularSurface`).

El diagrama UML resultante sería:

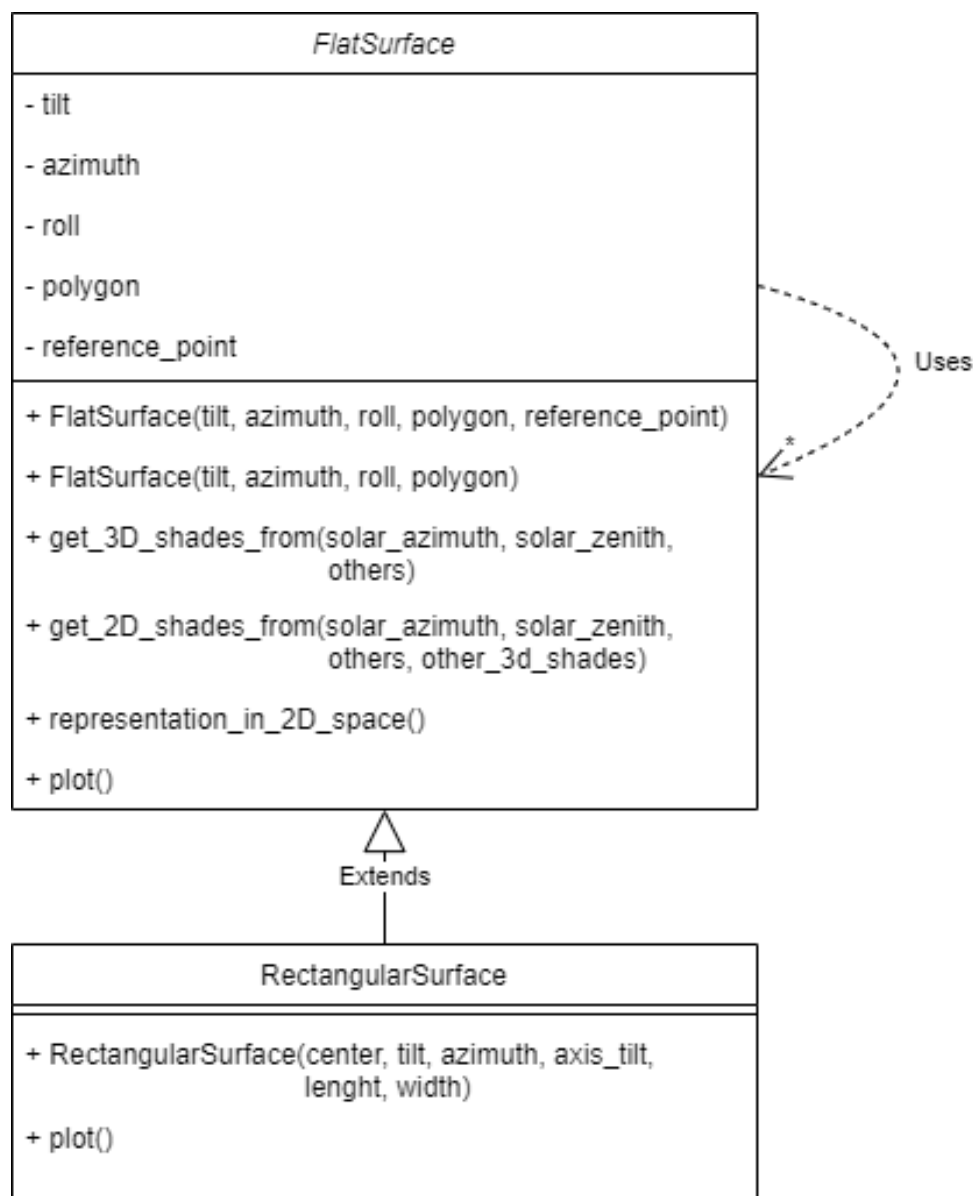


Figura 3.6: Diagrama UML de la propuesta de cálculo de sombras en 3D.

La elección de un diagrama tan sencillo no es arbitraria: aquí prima la simplicidad y la facilidad de revisar el código para determinar si tiene valor o no dentro de la librería *pvl-lib-python*. Y realmente se logra muy bien facilitar la API. Véase el ejemplo desarrollado a continuación:

Listing 3.4: Caso de uso de ejemplo para la propuesta de cálculo de sombras en 3D.

```

1 from pvl.lib.spatial import RectangularSurface
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
4 import shapely
5
6 solar_azimuth = 165 # degrees
7 solar_zenith = 75 # degrees
8
9 # Define two rows of panels
10 row1 = RectangularSurface( # south-most row
  
```

```

11     center=[0, 0, 3], azimuth=165, tilt=20, axis_tilt=10, width=2, length=20
12 )
13
14 row2 = RectangularSurface( # north-most row
15     center=[0, 3, 3], azimuth=165, tilt=30, axis_tilt=10, width=2, length=20
16 )
17
18 # Calculate shadows
19 shades_3d = row2.get_3D_shades_from(solar_zenith, solar_azimuth, row1)
20 shades_2d = row2.get_2D_shades_from(
21     solar_zenith, solar_azimuth, shades_3d=shades_3d
22 )
23
24 # Plot
25 row_style = {"color": "darkblue", "alpha": 0.5}
26 shade_style = {"color": "dimgrey", "alpha": 0.8}
27 row_style_2d = {**row_style, "add_points": False}
28 shade_style_2d = {**shade_style, "add_points": False}
29
30 fig = plt.figure(figsize=(10, 10))
31
32 # Split the figure in two axes
33 gs = fig.add_gridspec(10, 1)
34 ax1 = fig.add_subplot(gs[0:7, 0], projection="3d")
35 ax2 = fig.add_subplot(gs[8:, 0])
36
37 # 3D plot
38 ax1.view_init(
39     elev=60,
40     azim=-30, # matplotlib's azimuth is right-handed to Z+, measured from X+
41 )
42 row1.plot(ax=ax1, **row_style)
43 row2.plot(ax=ax1, **row_style)
44 for shade in shades_3d.geoms:
45     if shade.is_empty:
46         continue # skip empty shades; else an exception will be raised
47     # use Matplotlib's Poly3DCollection natively since experimental
48     # shapely.plotting.plot_polygon does not support 3D
49     vertexes = shade.exterior.coords[:-1]
50     ax1.add_collection3d(Poly3DCollection([vertexes], **shade_style))
51
52 ax1.axis("equal")
53 ax1.set_zlim(0)
54 ax1.set_xlabel("West(-) <X> East(+) [m]")
55 ax1.set_ylabel("South(-) <Y> North(+) [m]")
56
57 # 2D plot
58 row2_2d = row2.representation_in_2D_space()
59 shapely.plotting.plot_polygon(row2_2d, ax=ax2, **row_style_2d)
60 for shade in shades_2d.geoms:
61     shapely.plotting.plot_polygon(shade, ax=ax2, **shade_style_2d)
62
63 # Calculate the shaded fraction
64 shaded_fraction = sum(shade.area for shade in shades_2d.geoms) / row2_2d.area
65 print(f"The shaded fraction is {shaded_fraction:.2f}")

```

Debe denotarse que la mayor parte del código supone imprimir la escena y las sombras por pantalla. La parte más interesante, que es el cálculo de las sombras y la fracción sombreada, se reduce a unas pocas 14 líneas de puro código, mientras que se necesitan 27 para mostrar el resultado en 3D y 2D. La escena con las sombras es la que se muestra en la figura 3.7:

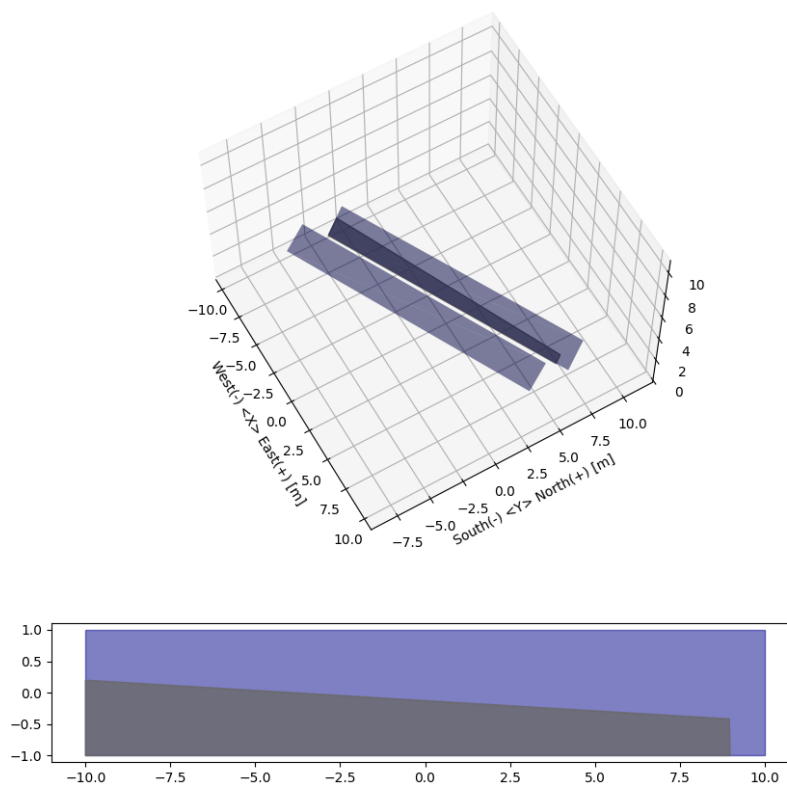


Figura 3.7: Ejemplo de sombreado para coordenadas solares instantáneas en 3D.

3.3. Contribuciones técnicas

3.3.1. Arreglo a los tests de integración continua en Windows con Conda

- *Issue*: #2000
- *Pull Request*: #2007

Por causas desconocidas, pero que se pueden intuir, fallaba un test que esperaba con bastante precisión que una entrada numérica a una función que resuelve la curva I-V del diodo devolviese un cero casi exacto. Se determinó que la causa era un exceso de precisión que se le pedía al test, y esto se cumplía en determinadas arquitecturas, y se modificó para que aceptase un margen de error mayor. Se puede deducir que se debe al entorno de pre-compilación de la librería.

3.3.1.1. Resultado

El cambio se incluyó con éxito.

3.3.2. Arreglo a un parámetro ignorado en una función de transposición inversa

- *Issue*: #1970
- *Pull Request*: #1971

Hasta avanzadas versiones en la librería no se añadieron los procedimientos de integración continua que existen a día de hoy y permiten identificar fallos de análisis estático en el código. Estos fallos son aquellos que se pueden identificar sin necesidad de ejecutar el código, y en este caso se trataba de un parámetro que no se estaba utilizando en una función.

Este parámetro sin usarse fue encontrado por el autor de este mismo TFG gracias al resaltado de sintaxis y procedió a modificar la función para que lo utilizase. El parámetro era `xtol` de la función `pvlib.irradiance.ghi_from_poa_driesse_2023`. Esta función hace una transposición inversa, que es deducir la irradiancia global horizontal a partir de la irradiancia en un plano inclinado, mediante un método numérico. El parámetro `xtol` es la tolerancia absoluta en la convergencia del método numérico.

3.3.2.1. Resultado

El cambio se incluyó con éxito, aportando tests de integridad para la función.

3.3.3. Dar soporte a otra función para el cálculo del IAM en el flujo orientado a objetos

- *Issue*: #1742
- *Pull Request*: #1832

El modificador del ángulo de incidencia tiene en cuenta la reflexión de la luz incidente cuando impacta oblicuamente en una superficie. La librería *pvlib-python* cuenta con muchos modelos que calculan este modificador, pero no todos están disponibles en la *Modelchain*, un flujo de trabajo orientado a objetos que facilita la simulación de sistemas fotovoltaicos.

El modelo que faltaba dar soporte era un interpolador de datos ángulo-modificador, `pvlib.iam.interp`.

3.3.3.1. Resultado

El cambio se incluyó con éxito y ahora se encuentra disponible su interfaz en la *Modelchain*, en `pvlib.modelchain.ModelChain.interp_aoi_loss`.

3.3.4. Suprimir una advertencia al publicar la distribución en PyPI

- *Pull Request*: #1778

Es buena práctica revisar los procedimientos que se realizan automáticamente, ya que con cierta frecuencia dan lugar a advertencias que se ignoran por no ser una cuestión crítica. Dentro de la labor que decide aplicar el autor de este TFG, se revisa manualmente el proceso de tests unitarios, construcción y publicación de la librería

en PyPI, y se detecta una advertencia que se puede suprimir aportando un parámetro en el fichero del proyecto.

No se trata de nada crítico: tan sólo es la plataforma PyPI que solicita explícitamente el formato del texto de la descripción del proyecto.

3.3.4.1. Resultado

El cambio se incluyó con éxito, gracias también a la profunda revisión de un mantenedor que detectó un fallo en el formato que el autor de este TFG había especificado.

3.3.5. Exponer parámetros de tolerancia para resolver el modelo de un diodo

- *Issue*: #1249
- *Pull Request*: #1764

Una de las solicitudes que algún usuario había planteado como mejora era la posibilidad de modificar los parámetros de tolerancia de los métodos numéricos que se utilizan para resolver la curva I-V de un diodo. Estos métodos son iterativos y requieren de una precisión para converger.

Por defecto la tolerancia es de 10^{-6} , lo que a este usuario le parecía excesivo. Nótese que dependen de los parámetros de la curva del diodo que haya que resolver.

3.3.5.1. Resultado

El cambio se incluyó con éxito, y se añadieron tests profundos para asegurar que los parámetros de tolerancia se recibían adecuadamente.

3.3.6. Modificar tolerancias erróneas en varios tests unitarios

- *Pull Request*: #2082

Originalmente la librería empleó un framework de testing que especificaba las tolerancias como número de decimales después de la coma. Al cambiar de framework al nuevo y más moderno *pytest*, no se modificaron estos números para reflejar el nuevo significado de las tolerancias, como un margen absoluto en torno al que deben estar los números.

3.3.6.1. Resultado

El cambio se incluyó con éxito, sin diagnosticar ningún fallo derivado de esta errata.

3.3.7. Arreglo de un bug que ignoraba parámetros de una función de lectura de bases de datos

- *Issue*: #2018
- *Pull Request*: #2020

En *pvl-lib-python* existe la función `pvl-lib.pvsystem.retrieve_sam` que permite leer de bases de datos ya incluidas en la librería o remotas mediante una URL, mediante dos parámetros distintos. Resulta que si se especificaban ambos, solo se leía la base de datos de la distribución, y no la remota, sin emitir ningún aviso.

3.3.7.1. Resultado

Se aplicó un patrón de diseño que excluía la posibilidad de que ambos parámetros se especificasen a la vez, y se incluyó sin dificultades.

3.3.8. Actualizar versiones de las dependencias de la documentación

- *Pull Request*: #2112

Esta contribución se trata de otro de los trabajos en curso que lleva el autor de este TFG a día de redacción de este documento. Se trata de actualizar las versiones de las dependencias de la documentación, que se construye automáticamente con *Sphinx* y algunas extensiones extras, para dotar de un nuevo *look-and-feel* (o, estilo) a la documentación.

3.3.8.1. Resultado

Se sigue trabajando en esta contribución, procurando que pueda pasar el mayor tiempo posible sin necesidad de ser actualizada.

3.4. Contribuciones menores

3.4.1. Corrección de erratas en la documentación

- *Pull Request*: #1599
- *Pull Request*: #1860
- *Pull Request*: #1996

Sin mayor elaboración al respecto, se han solventado erratas en la documentación.

3.4.2. Corrección de erratas en ejemplos y en código

- *Pull Request*: #1776
- *Pull Request*: #1833

Son pequeñas modificaciones que no requieren mayor explicación: actualizar la salida de un ejemplo y comprobar que el único script de ejemplo que no se ejecuta, lo hiciera.

3.4.3. Modificación de escritura de los parámetros opcionales

- *Issue*: #1574
- *Pull Request*: #1828

■ Pull Request: #2084

En Python existe un patrón de diseño que es darle el valor `None` a los parámetros opcionales, y luego comprobar si son `None` para asignarles un valor por defecto o ignorarlos si fuese el caso. Normalmente, a la hora de documentarlo lo normal es indicar que el parámetro es opcional, pero en la librería se indicaba que el valor por defecto era `None`, lo cual es redundante y obfusca las intenciones del código.

Se cambiaron todas las ocurrencias de `default None` a `optional`, aunque hubo que iterar múltiples veces para dar con todos los falsos negativos y no obtener falsos positivos. Para esta tarea se empleó el buscador en archivos integrado en *Visual Studio Code*, que está basado en *ripgrep* y hace uso de expresiones regulares, una forma de búsqueda y sustitución muy potente. Las expresiones regulares son un lenguaje formal que permite buscar patrones en texto, por ejemplo sustituyendo un carácter por grupos de caracteres, o buscando un patrón que se repite un número determinado de veces.

Se emplearon múltiples expresiones regulares, escritas en el sabor PCRE2, que es el que soporta *ripgrep*. Según la utilidad, podemos encontrar pequeñas variaciones en la forma de escribir estas expresiones. En los *commits* de la propuesta se indican todas las expresiones empleadas, pero aquí se adjuntan algunas de las usadas:

Expresión regular	Patrón de sustitución
(? {8} {4})(?w*) ? : (?.*), default:? None\.	\$1\$2 : \$3, optional
(? {8} {4})(?w*) ? : (?.*), default:? None?	
(? {8} {4})(?w*) ? : (?.*), or [nN]one, optional	
(? {8} {4})(?w*) ? : [Nn]one, (?.*), default:? (?.*).\.	\$1\$2 : \$3, default \$4
(? {8} {4})(?w*) ? : (?.*), or [Nn]one(?.*)	\$1\$2 : \$3\$4
If None	If not specified,

Tabla 3.1: Algunas de las expresiones regulares empleadas para modificar la escritura de los parámetros opcionales.

Adicionalmente se arreglaron algunos links a los DOI, que son identificadores únicos de documentos científicos, gracias también a la búsqueda y sustitución con expresiones regulares:

- Expresión de búsqueda: `(?:doi|DOI) : (?!\`)\s?(.*?) (\.\n|\n)`
- Expresión de sustitución: `:doi:~$1~$2`

3.4.4. Limpieza de advertencias al construir la documentación

■ Pull Request: #2030

De nuevo, en la línea de erradicar advertencias en los flujos de integración continua, se eliminaron advertencias que se emitían al construir la documentación. Estas advertencias no eran críticas, pero al formar parte de unos registros que se consultan con cierta periodicidad cada vez que alguien hace contribuciones, se prefiere evitar ruido visual. De esta forma, el proceso de revisión es más eficiente y no confunde ni a los revisores ni a los contribuyentes.

Además, se mejoró así el renderizado de la documentación en bastantes casos.

3.5. Interacciones con otros usuarios

Ha sido constante la comunicación con los revisores y los contribuyentes que más activos son en la librería. Se ha procurado perseguir una comunicación clara, concisa y razonada, y se ha agradecido la colaboración y la transmisión de nuevas ideas, si bien no se han aplicado todas. Este caso es el más esperable, pues se tratan de las personas con más responsabilidad sobre la librería.

Por otro lado, se ha ayudado a guiar a los compañeros alumnos del *Google Summer of Code* en sus propuestas y reuniones.

Y finalmente, con el sector de usuarios que discretamente han participado, bien por dudas o por descubrir errores, se ha transmitido y aportado con claridad y amabilidad.

A continuación se puede encontrar en más detalle estos dos últimos grupos, que son lo de mayor interés.

3.5.1. Revisión de propuestas de otros usuarios

En especial con aquellos compañeros partícipes bajo la subvención de *Google Summer of Code*, pero también con otros usuarios que han planteado propuestas en la librería, se ha colaborado en la revisión de sus propuestas.

Se podrían citar muchos ejemplos, ya que esta parte es bastante abundante por requerir un esfuerzo relativamente bajo: incluso revisando un propuesta basada en algún artículo, pero no pudiendo consultar este, se puede aportar mucho. Normalmente se ha revisado el renderizado y claridad de la documentación, y la integridad de los tests. Se presta máxima atención a la parte más social, dar el crédito a todos los nuevos implicados en la librería.

En otras situaciones, se ha aportado información técnica sobre normativa, como en una propuesta de un modelo de pérdidas en transformadores. Revisores del código pidieron información sobre los tests que se realizan físicamente en transformadores de alta tensión. El mismo autor de este TFG aportó un ejemplo y la normativa que se sigue en España para realizar estos tests. Se agradece la colaboración de la empresa *Laboratorio Central Oficial de Electrotecnia*, departamento de Alta Tensión, por facilitar la información.

3.5.2. Soporte técnico a usuarios

Otro aspecto, de no gran importancia ya que no se ha dispuesto de mucho tiempo para ello, es el soporte técnico a usuarios. Se ha respondido a preguntas en *GitHub*, y en la lista de correo por *Google Groups*. Implicarse en esto ha permitido conocer mejor la librería, detectar errores en el código y adelantar futuros errores con las dependencias.

Es más, respecto de las dependencias, se inicia parcialmente un trabajo en eliminar una sub-dependencia de una de las dependencias, valga la redundancia, de *pulib-python*. Se trata de *solarfactors*, un clon mantenido por la comunidad de *pulib* del paquete *pufactors*, que es un software de simulación de factores de vista en sistemas fotovoltaicos bifaciales.

Desarrollo

Sucede que esta dependencia utiliza una versión antigua e incompatible de *Shapely*, que no se puede instalar en Python 3.12, así que se plantea eliminarla. Este trabajo, si bien iniciado, resulta especialmente complejo por trabajar con un código base novedoso para el alumno, por requerir una elevada inversión de tiempo y no estar ligado estrechamente a la librería objeto de este TFG.

Capítulo 4

Impacto del trabajo

En este capítulo se analizará el valor que aporta el trabajo realizado en este TFG.

La sección 4.1 se centrará en el impacto general del trabajo, mientras que la sección 4.2 analizará el impacto del trabajo en relación con los Objetivos de Desarrollo Sostenible (ODS). Por último, se hace una valoración de competencias adquiridas por el alumno en la sección 4.3.

4.1. Impacto general

Este Trabajo Fin de Grado permite visibilizar y extender el uso de algunos modelos científicos relacionados con la fotovoltaica. Además, se ha facilitado el mantenimiento de la librería *pulib-python* y se ha mejorado la documentación de la misma, contribuyendo así a la persistencia de este proyecto. Incluso sin haberse añadido algunas de las contribuciones propuestas, el trabajo realizado ha permitido reforzar líneas de trabajo que antes ni se consideraban.

Se espera que algunos de los modelos sean bastante útiles para la comunidad científica y técnica, en especial aquellos que consisten en modelos de pérdidas. Estos modelos permiten optimizar el diseño de plantas y mejorar el rendimiento económico de la instalación. Además, la mejora de la documentación de la librería *pulib-python* facilita su uso y extensión, lo que puede llevar a un aumento de la comunidad de usuarios y contribuidores.

Dentro del programa subvencionado *Google Summer of Code*, se ha trabajado estrechamente con otros compañeros desarrolladores y se les ha ayudado a desenvolverse en el proyecto. Esto ha permitido que se hayan añadido nuevas funcionalidades a la librería y se haya mejorado la calidad del código.

Asimismo, este trabajo se alinea con los objetivos de la Universidad pública de promover la investigación, la transferencia y democratización libre del conocimiento.

4.2. Objetivos de Desarrollo Sostenible

Con este trabajo se ha contribuido a la consecución de los Objetivos de Desarrollo Sostenible (ODS):

- **ODS 3: Salud y bienestar.** Al potenciar el uso de energías renovables, se contribuye a la reducción de la contaminación y, por tanto, a la mejora de la salubridad del medio ambiente.
- **ODS 7: Energía asequible y no contaminante.** Debido a que se facilitan herramientas de diseño y análisis de instalaciones fotovoltaicas, se contribuye a la mejora de la eficiencia de las mismas y, por tanto, a la reducción del impacto de instalar paneles solares.
- **ODS 9: Industria, innovación e infraestructura.** Se ha trabajado en la mejora de la infraestructura de la librería *pulib-python*, lo que facilita la innovación y desarrollo en el sector de la energía fotovoltaica.
- **ODS 11: Ciudades y comunidades sostenibles.** Las mejoras facilitadas permiten implementar instalaciones fotovoltaicas de forma más fiable, lo que contribuye a su implantación.
- **ODS 13: Acción por el clima.** Este ODS sigue en la misma línea que los demás: como fuente de energía renovable, y en especial por ser la fotovoltaica, se reduce la emisión de gases de efecto invernadero y se contribuye a la lucha contra el cambio climático.

4.3. Impacto académico en el autor

Este trabajo ha permitido al estudiante adquirir una serie de habilidades y conocimientos que le han sido y le continuarán siendo útiles en su carrera profesional. Además, le ha permitido conocer de primera mano cómo se trabaja en un proyecto de código abierto para poder aplicarlo en actuales proyectos de código abierto que mantiene.

Si bien puede destacar que la adquisición de conocimiento del lenguaje de Python no ha sido muy destacable, toda la parte de herramientas y la plataforma de GitHub ha sido muy enriquecedora:

- Se ha aprendido a usar procedimientos de integración y desarrollo continuo.
- Se ha aprendido a desarrollar tests comprensivos y útiles.
- Se ha aprendido a escribir documentación de calidad.
- Se ha reforzado la capacidad de trabajo colaborativo en código.
- Se han mejorado las habilidades sociales y de comunicación a través de plataformas como GitHub.
- Se ha mejorado la destreza de comunicación en inglés, tanto oral como escrita.

Capítulo 5

Resultados y conclusiones

En este capítulo se resumen los resultados obtenidos en el desarrollo del Trabajo Fin de Grado, aquellos tanto satisfactorios como los que no, y se extraen conclusiones personales del estudiante. Además, se proponen posibles líneas de trabajo futuro, algunas que seguirán desarrollándose y otras que se podrían seguir a partir de este TFG.

5.1. Resultados

A pesar de que los ciclos de revisiones y correcciones toman un elevado tiempo de desarrollo, se han conseguido implementar varios modelos y funcionalidades nuevos en *pvlb-python*. Además, se ha mejorado la documentación de la librería y se han corregido errores en la misma. Cabe destacar:

1. Exponer el cálculo de la proyección del cenit solar sobre las coordenadas de un colector.
2. Implementar un cálculo de la fracción de sombra unidimensional.
3. La implementación de un modelo de pérdidas de potencia por sombreado, según número de diodos de bypass.
4. Un modelo de fracción de irradiancia difusa fotosintetizable, en función de la irradiancia difusa global.
5. Un modelo de pérdidas de potencia por heterogeneidad en la irradiancia incidente, de interés para módulos bifaciales.
6. Dos funciones de conversión recíprocas de responsividad espectral y eficiencia cuántica externa, con posibilidad de normalizar la salida.
7. Adición de una función general para obtener espectros estándares, que cuenta con el estándar ASTM G173-03, pero que será muy fácilmente extendible.
8. Solventar 5 bugs y 2 características nuevas de las que han informado usuarios varios.
9. Mejorar la documentación ya existente de forma amplia.

Resultados y conclusiones

10. Participar en detectar posibles contribuciones para primeros contribuyentes en el futuro.
11. Aportar ideas y planteamientos que faciliten contribuir a la librería.

Algunas cifras significativas, para la librería *pvl-lib-python* son:

- Se han aceptado 22 *pull requests*.
- Se han desestimado 4 *pull requests*.
- Se mantienen abiertas 6 *pull requests*.
- Se han revisado 14 *pull requests* de otros usuarios.
- Se han abierto 18 *issues*.
 - De las cuales se han cerrado 11 *issues*.
- Se ha participado en 24 *issues* de otros usuarios.
 - De las cuales se han cerrado 13 *issues*.

Adicionalmente se aprecia impacto en otros repositorios como:

- *solarfactors*: un repositorio para el cálculo de factores de vista de sistemas bifaciales, en una escena 2D.
 - Se ha informado de problemas de compatibilidad con la versión de Python 3.12 por el uso de la dependencia `shapely<2`¹. Esta problemática es la anteriormente descrita en 3.5.2.
- *openpvtools/pv-foss-engagement*: una página con información sobre distintas librerías de Python empleadas en fotovoltaica.
 - Se ha informado de la ausencia de *solarfactors* en la lista de librerías² y se deja abierta la futura colaboración.

5.2. Conclusiones

Con la realización de este Trabajo Fin de Grado, se promueven y mejoran proyectos de código abierto ya establecidos, lo que garantiza la usabilidad de las aportaciones realizadas. Además, el estudiante ha adquirido una serie de habilidades y conocimientos que le serán útiles en su carrera profesional.

Se han mejorado múltiples aspectos de la librería *pvl-lib-python*, desde errores menores en el código a la implementación de nuevos modelos y funcionalidades. Se ha mejorado la documentación bastante, dotando al proyecto de una mayor usabilidad. El aporte de este TFG es completamente transversal a todos los efectos de este repositorio y se espera que sea de utilidad para la comunidad científica y técnica. Se espera que además se facilite el acceso de nuevos contribuyentes a la librería.

Desde un punto de vista más personal, el alumno declara la alineación de sus motivaciones y expectativas con los resultados obtenidos. Asimismo cree viable una

¹Véase <https://github.com/pvlib/solarfactors/issues/16>.

²Véase <https://github.com/openpvtools/pv-foss-engagement/issues/8>.

mayor oferta de trabajos fin de grado en la misma línea, si bien es cierto que hay dos aspectos que podrían haberse mejorado:

- Los artículos que se ofrecían candidatos a aportar a veces no contaban con requisitos funcionales razonables para hacer de ellos una propuesta formal.
- La planificación del trabajo, que responde a ciclos de trabajo extensos por el aprendizaje paralelo al desarrollo, la demora de las revisiones de .

5.3. Trabajo futuro

Inicialmente se trabajará en contrastar la implementación del modelo descrito en 3.2.6, cuyos autores solicitan y facilitan información para contrastar la implementación con la suya.

Se continuarán las propuestas ya planteadas, tanto dentro como fuera del programa subvencionado *Google Summer of Code*. Y se priorizará dotar a la web de una mejor estética y usabilidad empleando las últimas versiones posibles.

Además, posiblemente se intente continuar en la línea de solventar el problema de la dependencia `shapely<2` en *solarfactors*.

Por último, se ayudará a mejorar la documentación sobre cómo contribuir, y plantear una hoja de ruta para futuros contribuyentes que no hayan visto cómo programar en esta librería antes, pues se identifica que no todos los contribuyentes han visto cómo usar control de versiones en sus titulaciones, en contraste al Grado en Ingeniería Electrónica Automática e Industrial que cursa el estudiante.

Bibliografía

- [1] K.W. Böer y D. Bimberg. *Survey of Semiconductor Physics, Survey of Semiconductor Physics*. Wiley, 2002. ISBN: 9780471355724. URL: https://books.google.es/books?id=_pdvAQAAACAAJ.
- [2] Mahmood H. Shubbak. «Advances in solar photovoltaics: Technology review and patent trends». En: *Renewable and Sustainable Energy Reviews* 115 (nov. de 2019), pág. 109383. ISSN: 1364-0321. DOI: 10.1016/j.rser.2019.109383.
- [3] William Shockley y Hans J. Queisser. «Detailed Balance Limit of Efficiency of p-n Junction Solar Cells». En: *Journal of Applied Physics* 32.3 (mar. de 1961), págs. 510-519. ISSN: 0021-8979. DOI: 10.1063/1.1736034.
- [4] Martin A. Green et al. «Solar cell efficiency tables (Version 63)». en. En: *Progress in Photovoltaics: Research and Applications* 32.1 (2024), págs. 3-13. ISSN: 1099-159X. DOI: 10.1002/pip.3750.
- [5] O. Perpiñán. *Energía Solar Fotovoltaica*. 2020. URL: <http://oscarperpinan.github.io/esf/>.
- [6] Joshua S Stein y Geoffrey T Klise. «Models used to assess the performance of photovoltaic systems.» En: (dic. de 2009). DOI: 10.2172/974415. URL: <https://www.osti.gov/biblio/974415>.
- [7] Nallapaneni Manoj Kumar. «Simulation Tools for Technical Sizing and Analysis of Solar PV Systems». En: *Proceedings of the 6th World Conference on Applied Sciences, Engineering and Technology (WCSET-2017), 26-27 August 2017, UM-PO, Indonesia, ISBN 13: 978-81-930222-3-8, pp 218-222, At Universitas Muhammadiyah Ponorogo, Indonesia*. (ene. de 2017). URL: https://www.academia.edu/35141273/Simulation_Tools_for_Technical_Sizing_and_Analysis_of_Solar_PV_Systems.
- [8] Joshua S. Stein et al. «PVLIB: Open source photovoltaic performance modeling functions for Matlab and Python». En: *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*. Jun. de 2016, págs. 3425-3430. DOI: 10.1109/PVSC.2016.7750303. URL: <https://ieeexplore.ieee.org/abstract/document/7750303>.
- [9] Kevin S. Anderson et al. «pvlb-python: 2023 project update». En: *Journal of Open Source Software* 8.92 (2023), pág. 5994. DOI: 10.21105/joss.05994. URL: <https://doi.org/10.21105/joss.05994>.
- [10] Joshua S. Stein. «The photovoltaic Performance Modeling Collaborative (PVPMLC)». En: *2012 38th IEEE Photovoltaic Specialists Conference*. Jun. de 2012, págs. 003048-003052. DOI: 10.1109/PVSC.2012.6318225. URL: <https://ieeexplore.ieee.org/document/6318225>.
- [11] Robert W. Andrews et al. «Introduction to the open source PV LIB for python Photovoltaic system modelling package». En: *2014 IEEE 40th Photovoltaic Spe-*

- cialist Conference (PVSC)*. Jun. de 2014, págs. 0170-0174. DOI: 10.1109/PVSC.2014.6925501. URL: <https://ieeexplore.ieee.org/document/6925501>.
- [12] William F. Holmgren et al. «pvlib-python 2015». En: *2015 IEEE 42nd Photovoltaic Specialist Conference (PVSC)*. Jun. de 2015, págs. 1-5. DOI: 10.1109/PVSC.2015.7356005. URL: <https://ieeexplore.ieee.org/document/7356005>.
- [13] William F. Holmgren y Derek G. Groenendyk. «An open source solar power forecasting tool using PVLIB-Python». En: *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*. Jun. de 2016, págs. 0972-0975. DOI: 10.1109/PVSC.2016.7749755. URL: <https://ieeexplore.ieee.org/document/7749755>.
- [14] G. van Rossum. *Python tutorial*. Inf. téc. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), mayo de 1995.
- [15] N. Martín y J. M. Ruiz. «A new method for the spectral characterisation of PV modules». en. En: *Progress in Photovoltaics: Research and Applications* 7.4 (1999), págs. 299-310. ISSN: 1099-159X. DOI: 10.1002/(SICI)1099-159X(199907/08)7:4<299::AID-PIP260>3.0.CO;2-0.
- [16] Mitchell Lee y Alex Panchula. «Spectral correction for photovoltaic module performance based on air mass and precipitable water». En: *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*. Jun. de 2016, págs. 1351-1356. DOI: 10.1109/PVSC.2016.7749836. URL: <https://ieeexplore.ieee.org/abstract/document/7749836>.
- [17] J. A. Caballero et al. «Spectral Corrections Based on Air Mass, Aerosol Optical Depth, and Precipitable Water for PV Performance Modeling». En: *IEEE Journal of Photovoltaics* 8.2 (mar. de 2018), págs. 552-558. ISSN: 2156-3403. DOI: 10.1109/JPHOTOV.2017.2787019.
- [18] Nuria Martín Chivelet. «Estudio de la influencia de la reflexión, el ángulo de incidencia y la distribución espectral de la radiación solar en los generadores fotovoltaicos». PhD Thesis. 1999.
- [19] E. Lorenzo, L. Narvarte y J. Muñoz. «Tracking and back-tracking». en. En: *Progress in Photovoltaics: Research and Applications* 19.6 (2011), págs. 747-753. ISSN: 1099-159X. DOI: 10.1002/pip.1085.
- [20] Kevin Anderson y Mark Mikofski. *Slope-Aware Backtracking for Single-Axis Trackers*. English. NREL/TP-5K00-76626. Jul. de 2020. DOI: 10.2172/1660126. URL: <https://www.osti.gov/biblio/1660126>.
- [21] Kevin S. Anderson y Adam R. Jensen. «Shaded fraction and backtracking in single-axis trackers on rolling terrain». En: *Journal of Renewable and Sustainable Energy* 16.2 (abr. de 2024), pág. 023504. ISSN: 1941-7012. DOI: 10.1063/5.0202220.
- [22] F. Martínez-Moreno, J. Muñoz y E. Lorenzo. «Experimental model to estimate shading losses on PV arrays». En: *Solar Energy Materials and Solar Cells* 94.12 (dic. de 2010), págs. 2298-2303. ISSN: 0927-0248. DOI: 10.1016/j.solmat.2010.07.029.
- [23] C. J. T. Spitters, H. A. J. M. Toussaint y J. Goudriaan. «Separating the diffuse and direct component of global radiation and its implications for modeling canopy photosynthesis Part I. Components of incoming radiation». En: *Agricultural and Forest Meteorology* 38.1 (oct. de 1986), págs. 217-229. ISSN: 0168-1923. DOI: 10.1016/0168-1923(86)90060-2.
- [24] C. J. T. Spitters. «Separating the diffuse and direct component of global radiation and its implications for modeling canopy photosynthesis Part II. Calculation of canopy photosynthesis». En: *Agricultural and Forest Meteorology* 38.1

- (oct. de 1986), págs. 231-242. ISSN: 0168-1923. DOI: 10.1016/0168-1923(86)90061-4.
- [25] S. Ma Lu et al. «Photosynthetically active radiation decomposition models for agrivoltaic systems applications». En: *Solar Energy* 244 (sep. de 2022), págs. 536-549. ISSN: 0038-092X. DOI: 10.1016/j.solener.2022.05.046.
- [26] Chris Deline et al. «Estimating and parameterizing mismatch power loss in bifacial photovoltaic systems». en. En: *Progress in Photovoltaics: Research and Applications* 28.7 (2020), págs. 691-703. ISSN: 1099-159X. DOI: 10.1002/pip.3259.
- [27] URL: <https://www.semanticscholar.org/paper/Gini%E2%80%99s-Mean-difference%3A-a-superior-measure-of-for-Yitzhaki/e4d00851cbbadf386ed051397c>
- [28] Tom Markvart y Luis Castañer. «Principles of solar cell operation». En: *Practical Handbook of Photovoltaics*. Elsevier, 2012, págs. 7-31.
- [29] *Standard Tables for Reference Solar Spectral Irradiances: Direct Normal and Hemispherical on 37° Tilted Surface* — *astm.org*. <https://www.astm.org/g0173-03.html>. 2003.
- [30] *Standard Tables for Reference Solar Spectral Irradiances: Direct Normal and Hemispherical on 37° Tilted Surface* — *astm.org*. <https://www.astm.org/standards/g173>. 2023.
- [31] Sebastian Zainali et al. «Direct and diffuse shading factors modelling for the most representative agrivoltaic system layouts». En: *Applied Energy* 339 (jun. de 2023), pág. 120981. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2023.120981.