

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

TRABAJO FINAL DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa del Trabajo Final de Graduación

Requisito para optar por el título de Ingeniero en Electrónica

Grado Académico de Licenciatura

Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del Trabajo Final de Graduación denominado **Desarrollo de una herramienta de evaluación para modelos de simulación de paneles solares bifaciales mediante Python**, realizado por el señor Juan Pablo Cubero Murillo y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

TEC | Tecnológico
de Costa Rica

Firmado digitalmente
por ANIBAL COTO
CORTES (FIRMA)
Fecha: 2022.06.13 15:
25:51-06'00'

Ing. Aníbal Coto Cortés

Profesor lector

JUAN CARLOS
JIMENEZ
ROBLES (FIRMA)

Firmado digitalmente
por JUAN CARLOS
JIMENEZ ROBLES (FIRMA)
Fecha: 2022.06.15
23:54:56 -06'00'

Ing. Juan Carlos Jiménez Robles

Profesor lector

Firmado por SERGIO ARTURO MORALES HERNANDEZ (FIRMA)
PERSONA FISICA, CPF-02-0455-0934.
Fecha declarada: 13/06/2022 03:21 PM
Esta representación visual no es fuente
de confianza. Valide siempre la firma.

Ing. Sergio Morales Hernández

Profesor asesor

Cartago, 13 de junio de 2022

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica



**Desarrollo de una herramienta de
evaluación para modelos de simulación de
paneles solares bifaciales mediante Python**

Informe de Trabajo Final de Graduación para optar por el título
de Ingeniería en Electrónica con el grado académico de
Licenciatura

Juan Pablo Cubero Murillo
Carné: 2014049805

Profesor asesor:
Sergio Morales Hernández

Cartago, Costa Rica
I Semestre 2022

Declaración de autenticidad

Declaro que el presente Trabajo de Graduación ha sido realizado en su totalidad por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado material bibliográfico, he procedido a indicar las fuentes mediante citas. En consecuencia, asumo responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe.



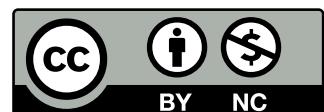
Juan Pablo Cubero Murillo

Cédula: 304910412

Cartago, junio 2022

Desarrollo de una herramienta de evaluación para modelos de simulación de paneles solares bifaciales mediante Python © 2022 by Juan Pablo Cubero-Murillo is licensed under Attribution-NonCommercial 4.0 International. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/>

Esta obra está bajo una licencia Creative Commons “Atribución-NoComercial 4.0 Internacional”.



Resumen

Los sistemas fotovoltaicos son una parte importante de las tecnologías para generación de energía limpia, en medio de la creciente “economía verde” que busca reducir los efectos que provocan el cambio climático sobre nuestro planeta. Entre las distintas tecnologías fotovoltaicas disponibles actualmente se encuentran los sistemas bifaciales, los cuales se han estado utilizando cada vez más en los últimos años, principalmente en plantas de generación eléctrica, debido a su capacidad de aprovechar tanto la cara frontal como la posterior de un módulo para generar más energía que un panel solar convencional (monofacial), prácticamente al mismo costo. Se proyecta que su utilización siga en aumento en la próxima década, sin embargo, aún se cuenta con desafíos para generar confianza en las partes interesadas en invertir en instalaciones de este tipo, principalmente alrededor de la estandarización y las herramientas y modelos de simulación y estimación energética.

El presente proyecto pretende contribuir con el desafío de los modelos de estimación energética para módulos fotovoltaicos bifaciales al desarrollar una herramienta de software mediante el uso de *Python* con una interfaz gráfica de usuario versátil e intuitiva, que permite ingresar datos generados previamente por simuladores para evaluar la precisión entre ellos e incorporar mediciones reales para ayudar con la determinación de su exactitud, mediante la generación de gráficas y tablas representativas.

Se utilizan simulaciones para tres modelos de estimación energética: modelo empírico (*Solar World*), factor de forma (*PVSyst*) y trazadores de rayos (*NREL Bifacial Radiance*), para generar comparaciones sobre la ganancia bifacial, energía bifacial y eficiencia energética del módulo fotovoltaico bifacial. Adicionalmente, se incorporan mediciones reales de energía y ganancia bifacial tomadas de un experimento realizado en la planta de evaluaciones fotovoltaicas en exteriores de la Universidad de Ciencias Aplicadas de Anhalt, llamada *Anhalt Photovoltaic Performance and Lifetime Laboratory* (APOLLO), con la finalidad de generar errores de *Root Mean Square Error* y *Mean Bias Error* para validar la exactitud de los simuladores respecto a los datos medidos en campo, comparándolos también con la metodología de evaluación usada en un estudio alrededor de dichas mediciones en APOLLO.

El propósito general de este proyecto es contribuir a la colaboración científica internacional y la generación de conocimiento que permita crear un ambiente de mayor confianza y seguridad para las partes interesadas en invertir en instalaciones fotovoltaicas bifaciales, específicamente alrededor de las herramientas de simulación de estimación energética, y así acelerar los procesos de aceptación y aplicación de estas tecnologías.

Palabras clave: Energía solar, estimación energética, panel solar bifacial, Python, simulador

Abstract

Photovoltaic systems are an important part of clean energy generation technologies, amidst the growing “green economy” that seeks to reduce the effects of climate change on our planet. Among the different photovoltaic technologies currently available are bifacial systems, which have become more commonly used in recent years, mainly in power generation plants, due to their ability to take advantage of both the front and back faces of a module to generate more energy than a conventional (monofacial) solar panel, at practically the same cost. Their usage is projected to continue increasing in the next decade, however, there are still challenges to build confidence in the parties interested in investing in this type of facilities, mainly around standardization and energy simulation and estimation tools.

The present project aims to assist with the challenge of energy estimation models for bifacial photovoltaic modules, by developing a software tool through the use of *Python* with a versatile and intuitive graphical user interface, which allows the input of data previously generated by simulators to assess the precision of each other and incorporate real measurements to assist in determining their accuracy, by generating representative graphs and tables.

Simulations for three energy estimation models are used: empirical model (*Solar World*), form factor (*PVSyst*) and ray tracing (*NREL Bifacial Radiance*), to generate comparisons on the bifacial gain, bifacial energy and energy efficiency of the bifacial module. Additionally, real bifacial energy and gain measurements taken from an experiment conducted at the Anhalt University of Applied Sciences outdoor photovoltaic evaluation plant, called *Anhalt Photovoltaic Performance and Lifetime Laboratory* (APOLLO), are incorporated with the purpose of generating *Root Mean Square Errors* and *Mean Bias Errors* to validate the accuracy of the simulators with respect to the data measured in the field, also comparing them with the evaluation methodology used in a study around said measurements at APOLLO.

The general purpose of this project is to contribute to international scientific collaboration and the generation of knowledge that allows creating an environment of greater confidence and security for parties interested in investing in bifacial photovoltaic installations, specifically around energy estimation and simulation tools, and thus accelerate the processes of acceptance and application of these technologies.

Keywords: Bifacial solar panel, energy estimation, Python, simulator, solar enery.

Dedicatoria

A mis amados padres, Juan Carlos y Edith.

Agradecimientos

Primeramente quiero agradecerle especialmente a mis papás, Juan Carlos y Edith, ya que sin ellos no estaría hoy presente. Gracias por su apoyo y amor incondicional, por años de esfuerzo, trabajo duro y sacrificios. Siempre han sido mis más grandes maestros para la vida, presentes en las buenas y en las malas. También quiero agradecerle a mis hermanas, Daniela y Elena, por siempre brindarme apoyo y consejos de hermana mayor y menor.

Un agradecimiento a mi novia Rosselyn, quien me ha acompañado a lo largo de mis años de estudio universitario, brindándome siempre su amor, su apoyo y sus consejos.

También gracias a todos mis amigos y amigas dentro y fuera de la universidad, ahora colegas ingenieros e ingenieras. Durante años compartimos alegrías, risas, tristezas y lágrimas, largos días y noches de estudio y trabajo duro. Estoy orgulloso de todo lo que hemos logrado, y siempre estarán presentes en mi corazón.

Gracias al Tecnológico de Costa Rica y a la Escuela de Electrónica por ser mi segundo hogar durante años. También a todos los profesores que fueron parte de mi formación, no solo como buen profesional, sino también como una buena persona. Especiales gracias al profesor Hugo Sánchez Ortíz, quien fundamentó mis primeras bases en el mundo de los sistemas fotovoltaicos, y formó parte indispensable de este proyecto.

En general, mi gratitud infinita a todas las personas cercanas que han sido parte de mi vida hasta este punto, brindándome su apoyo y motivación. Este logro lo conseguimos juntos, y está dedicado a todos ustedes.

Juan Pablo Cubero Murillo
Cartago, Costa Rica, junio 2022

Índice general

1. Introducción	13
1.1. Entorno del proyecto	13
1.2. Definición del problema	18
1.2.1. Generalidades	18
1.2.2. Síntesis del problema	19
1.3. Enfoque de la solución	20
1.4. Objetivos	21
1.4.1. Meta	21
1.4.2. Objetivo general	21
1.4.3. Objetivos específicos	21
1.5. Estructura del documento	22
2. Fundamentos teóricos	23
2.1. Sistemas fotovoltaicos	23
2.1.1. Paneles monofaciales	27
2.1.2. Paneles bifaciales	28
2.1.3. Mercado para módulos bifaciales	29
2.2. Simuladores de estimación energética	32
2.3. Comparación de herramientas	35
2.4. Python	37
3. Arquitectura de herramienta de evaluación	38
3.1. Modelos de estimación seleccionados	38
3.1.1. Solar World	38
3.1.2. PVsyst	39
3.1.3. NREL Bifacial Radiance	41
3.2. Diseño de la herramienta	42
3.3. Entorno de desarrollo	50
3.3.1. Características del sistema	50
3.3.2. Bibliotecas utilizadas	50
3.3.3. Instalación de bibliotecas y herramientas	52
3.4. Casos simulados	54
3.4.1. Simulación en PVsyst	57
3.4.2. Simulación en NREL Bifacial Radiance	60
3.4.3. Simulación en Solar World	61
3.5. Métodos desarrollados	62
3.5.1. Interfaces gráficas en gui.py	62
3.5.2. “Variables globales” en globals.py	70
3.5.3. Abrir archivos y extraer información en extraction.py	71
3.5.4. Organizar datos en sorting.py	73

3.5.5. Cálculo de variables energéticas en empirical.py	75
3.5.6. Estimación de errores en statistics.py	76
3.5.7. Creación de gráficas en plotter.py	78
3.5.8. Generación de tablas en tables.py	81
4. Pruebas y verificación de la herramienta	85
4.1. Ejecución del programa	85
4.2. Entradas del programa	86
4.2.1. Carga de archivos de PVsyst	86
4.2.2. Carga de archivos de NREL Bifacial Radiance	88
4.2.3. Carga de datos para Solar World	90
4.2.4. Carga de datos reales	92
4.3. Salidas del programa	94
4.3.1. Gráficas y tablas entre simulaciones	94
4.3.2. Gráficas y tablas con datos reales	96
4.3.3. Gráficas y tablas de errores	97
4.4. Análisis y validación de resultados simulados y medidos	101
4.4.1. Ganancia bifacial	102
4.4.2. Energía bifacial	103
4.5. Síntesis	106
5. Conclusiones y recomendaciones	108
5.1. Conclusiones	108
5.2. Recomendaciones	109
A. Abreviaturas	110

Índice de figuras

1.1. Disminuciones en las emisiones de carbono bajo el escenario de 1.5 °C (%) [1]	13
1.2. Tendencias entre tecnologías para generación de energía a partir de fuentes renovables, entre los años 2010 y 2020, a partir de la capacidad instalada (MW) a nivel mundial [2].	14
1.3. Comparación entre tecnología solar fotovoltaica y solar térmica, entre los años 2010 y 2020, a partir de la capacidad instalada (MW) a nivel mundial [2].	15
1.4. Matriz energética de Costa Rica en el año 2021. [3]	15
1.5. Parque Solar Cooperativo, Pocosol de San Carlos. [4]	16
2.1. Organización de un generador fotovoltaico [5]	24
2.2. Tipos de celdas solares de silicio: Capa fina, monocristalina y policristalina [6]	25
2.3. Tipos de panel solar fotovoltaico según fabricación (silicio) [7]	25
2.4. Curva característica I-V de un panel fotovoltaico [7]	25
2.5. Curva característica I-V según la temperatura [7]	26
2.6. Curva característica I-V según la irradiancia [7]	27
2.7. Panel solar monofacial con marco y recubrimiento metálico trasero (a) y sin marco y recubrimiento trasero de vidrio (b) [8]	28
2.8. Comparación entre un modulo bifacial y monofacial [9]	28
2.9. Estructura de una celda monofacial y una celda bifacial [10]	29
2.10. Sectorización del mercado mundial para módulos monofaciales y bifaciales [11]	30
2.11. Flujo general de modelado de estimación energética en paneles bifaciales (adaptado de [9])	32
3.1. Interfaz gráfica de PVsyst, página principal de un proyecto	40
3.2. Interfaz gráfica de PVsyst, configuración de conjunto fotovoltaico	40
3.3. Interfaz gráfica de NREL Bifacial Radiance	41
3.4. Diagrama de bloques de la herramienta diseñada	43
3.5. Diagrama de componentes de la herramienta diseñada	44
3.6. Diagrama de casos de uso de la herramienta diseñada	45
3.7. GUI principal del programa	46
3.8. GUI para el caso de PVsyst	47
3.9. GUI para el caso de NREL	47
3.10. GUI para el caso de Solar World	48
3.11. GUI para el caso de datos medidos	49
3.12. Instalaciones experimentales, Bernburg, Alemania [9]	54
3.13. Mediciones tomadas en Bernburg, Alemania [9]	56
3.14. Orientación e instalación del panel fotovoltaico en PVsyst	58

3.15. Configuración de modulo e inversor en PVsyst	58
3.16. Configuración de albedo cero en PVsyst	59
3.17. Configuración de albedo variable en PVsyst	59
3.18. Código para agregar un módulo al programa de NREL, mediante el archivo <i>module.json</i>	60
3.19. Configuración de parámetros en herramienta NREL Bifacial para el mes de enero del año 2021	61
3.20. Creación de la estructura básica de la ventana principal	62
3.21. Creación del menú desplegable	62
3.22. Función asignada al menú	63
3.23. Creación de botones de <i>check</i>	63
3.24. Obtención de la selección de los botones de <i>check</i>	64
3.25. Creación y asignación de funciones para botones adicionales de la interfaz	65
3.26. Configuración básica de la ventana correspondiente a PVsyst	66
3.27. Configuración del directorio de los archivos para PVsyst	66
3.28. Código para llamar función de carga de archivos y asignación de potencia	67
3.29. Código para configurar ventana auxiliar de NREL	68
3.30. Código para llamar función de carga de archivos en NREL	68
3.31. Función <i>setcommonalb</i> de la ventana de Solar World	68
3.32. Función <i>entries</i> de la ventana de Solar World	69
3.33. Funciones <i>entriesBG</i> y <i>entriesBE</i> , correspondientes a la ventana de datos reales	70
3.34. Variables inicializadas como vacías en el módulo <i>globals.py</i>	71
3.35. Código para leer archivos y extraer información de PVsyst en el módulo <i>extraction.py</i>	72
3.36. Código para leer archivos y extraer información de NREL en el módulo <i>extraction.py</i>	73
3.37. Código de la función <i>sort</i> correspondiente al caso de PVsyst	74
3.38. Código de la función <i>sort</i> correspondiente al caso de NREL Bifacial Radiance	75
3.39. Código de las funciones contenidas en el módulo <i>empirical.py</i>	76
3.40. Código para estimar RMSE y MBE en <i>statistics.py</i>	77
3.41. Código para estimar errores de ganancia y energía bifacial en <i>statistics.py</i>	77
3.42. Código para graficar 1 variable en <i>plotter.py</i>	78
3.43. Código para graficar 2 variables en <i>plotter.py</i>	78
3.44. Código para graficar 3 variables en <i>plotter.py</i>	79
3.45. Código para graficar 4 variables en <i>plotter.py</i>	79
3.46. Código para graficar los tres modelos junto con los datos reales en <i>plotter.py</i>	80
3.47. Código para graficar los errores de ganancia bifacial para los tres modelos de simulación en <i>plotter.py</i>	81
3.48. Código para generar una tabla de tres columnas mediante la función <i>threecolumns</i> en <i>tables.py</i>	82
3.49. Código de la función <i>gen_table_sim</i> para los tres modelos de simulación y datos reales en <i>tables.py</i>	83

3.50. Código de la función <i>gen_table_sim</i> para los tres modelos de simulación en <i>tables.py</i>	83
3.51. Código de la función <i>gen_table_errors</i> para los tres modelos de simulación en <i>tables.py</i>	84
4.1. Comando inicial del programa desde <i>Windows Power Shell</i>	85
4.2. GUI principal con selección de PVsyst	86
4.3. GUI auxiliar con opciones para PVsyst	87
4.4. Ventana para cargar archivos CSV en caso de PVsyst	87
4.5. Notificación de carga exitosa de CSV en PVsyst	88
4.6. Notificación de potencia asignada correctamente en PVsyst	88
4.7. GUI principal con selección de NREL	89
4.8. GUI auxiliar con opciones para NREL	89
4.9. Ventana para cargar archivos CSV en caso de NREL	90
4.10. Notificación de carga exitosa de CSV en NREL	90
4.11. GUI principal con selección de Solar World	91
4.12. GUI auxiliar con opciones para Solar World	91
4.13. Notificación de cálculo exitoso en Solar World	92
4.14. GUI principal con selección de datos medidos	92
4.15. GUI auxiliar con espacios para introducir datos medidos	93
4.16. Notificación de carga exitosa de datos medidos	93
4.17. GUI principal con la selección de los tres modelos de estimación energética	95
4.18. Ganancia bifacial simulada para un año, correspondiente a los tres modelos de simulación	95
4.19. Energía bifacial simulada para un año, correspondiente a los tres modelos de simulación	96
4.20. Eficiencia energética del modulo simulada para un año, correspondiente a los tres modelos de simulación	96
4.21. Ganancia bifacial para un año, correspondiente a los tres modelos de simulación y a los datos reales	97
4.22. Energía bifacial para un año, correspondiente a los tres modelos de simulación y a los datos reales	97
4.23. RMSE y MBE para la ganancia bifacial	99
4.24. Tablas de RMSE y MBE para la ganancia bifacial, correspondientes a los tres modelos de simulación	99
4.25. RMSE y MBE para la energía bifacial	100
4.26. Tablas de RMSE y MBE para la energía bifacial, correspondientes a los tres modelos de simulación	100
4.27. Energía bifacial para modelos simulados de la herramienta diseñada vs simulaciones de referencia tomadas de [9]	104
4.28. RMSE y MBE para la energía bifacial de la herramienta diseñada	105
4.29. RMSE y MBE para la energía bifacial del caso de referencia [9]	105

Índice de cuadros

2.1. Matriz de Pugh (considerando los pesos)	36
3.1. Características del sistema	50
3.2. Parámetros constantes para las simulaciones	55
3.3. Valores de albedo promedio correspondientes a cada mes (adaptación de tabla tomada de [9])	55
3.4. Valores de referencia para BGE y energía bifacial, tomados de [9]	57
4.1. Parámetros utilizados para las simulaciones, según modelo simulado	101
4.2. Valores menores y mayores de RMSE para la ganancia bifacial, en el caso de los tres simuladores	102
4.3. Valores menores y mayores de RMSE para la energía bifacial, en el caso de los tres simuladores.	104

Capítulo 1

Introducción

1.1. Entorno del proyecto

En la lucha para combatir el cambio climático y reducir las emisiones de carbono, bajo la meta de mantener el calentamiento global por debajo de 1.5 °C (respecto a la temperatura promedio global previo a la Revolución Industrial) hasta el año 2050, en un compromiso planteado en el Acuerdo de París [12], las energías renovables tienen un papel fundamental. Según la Agencia Internacional de Energías Renovables (IRENA, por sus siglas en inglés), el uso de fuentes renovables para la producción de energía eléctrica representa un 25 % de los esfuerzos planteados para mantener el escenario de 1.5 °C, que en conjunto con la aplicación de técnicas para la conservación y uso eficiente de la energía, técnicas para la fijación y remoción de carbono, entre otros esfuerzos, podrían permitir una reducción de 36.9 gigatoneladas (3.69×10^{12} kilogramos) de CO₂ al año en el 2050 (esto se puede observar en la figura 1.1) [1].

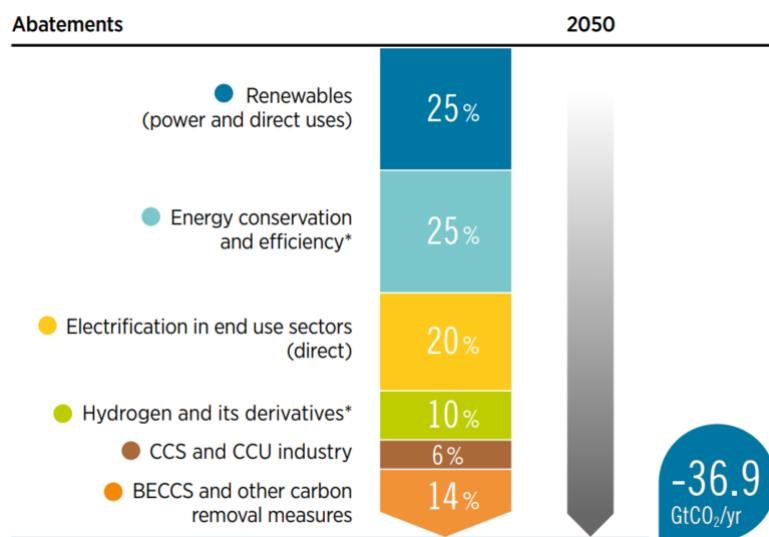


Figura 1.1: Disminuciones en las emisiones de carbono bajo el escenario de 1.5 °C (%) [1]

En los últimos años se ha trabajado en la transición hacia una matriz energética renovable a nivel global. Una de las tecnologías que ha ganado mayor impulso es la solar fotovoltaica, como se puede observar en la figuras 1.2 y 1.3. En la gráfica 1.2 se compara la tendencia a nivel mundial entre tecnologías para la generación de electricidad mediante fuentes renovables, entre los años 2010 y 2020, a partir de la capacidad instalada (MW) durante cada año. Los sistemas solares fotovoltaicos experimentaron el mayor aumento,

pasando de 40.334 MW en el año 2010, a 709.674 MW en el 2020.

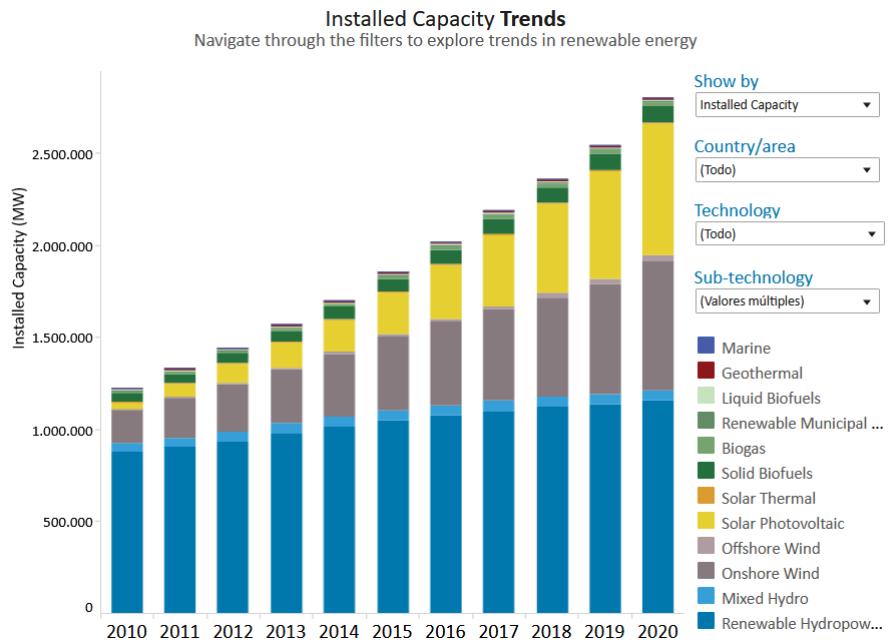


Figura 1.2: Tendencias entre tecnologías para generación de energía a partir de fuentes renovables, entre los años 2010 y 2020, a partir de la capacidad instalada (MW) a nivel mundial [2].

En la figura 1.3 se muestra de forma aislada este crecimiento de la tecnología solar, comparando solar fotovoltaica contra solar térmica [2]. Se proyecta que este crecimiento continúe durante las próximas décadas camino al año 2050, ya que la demanda está en aumento.

A nivel de Costa Rica, el ente principal a cargo de la generación y distribución de energía es el Instituto Costarricense de Electricidad (ICE). El ICE ha logrado construir, desde su fundación en el año 1949, una matriz energética enfocada en fuentes renovables: agua, calor de la tierra, viento, sol y biomasa, utilizando hidrocarburos en menor medida durante casos específicos. La generación solar por parte del ICE proviene de tres vertientes: la carga incorporada a la red nacional en su planta de Miravalles (Guanacaste), la generación distribuida y la electrificación en zonas indígenas o sin cableado por condiciones específicas [13]. De acuerdo con el Centro Nacional de Control de Energía (CENCE), para el año 2021 el 99.92 % de la demanda energética del país fue cubierta por fuentes renovables, de estas, el 0.08 % correspondió a la energía fotovoltaica. La principal fuente de energía en el país es (y lo ha sido históricamente) la hidroeléctrica, que para el 2021 cubrió un 71.7 % de la demanda (ver figura 1.4) [3]. A pesar de la gran cobertura de la demanda energética por parte del sector hidroeléctrico, desde el año 2015 se ha planteado la necesidad de alejarse del enfoque a los proyectos de este tipo, ya que existen sectores sociales y ambientales que han criticado tal desarrollo en virtud de los impactos

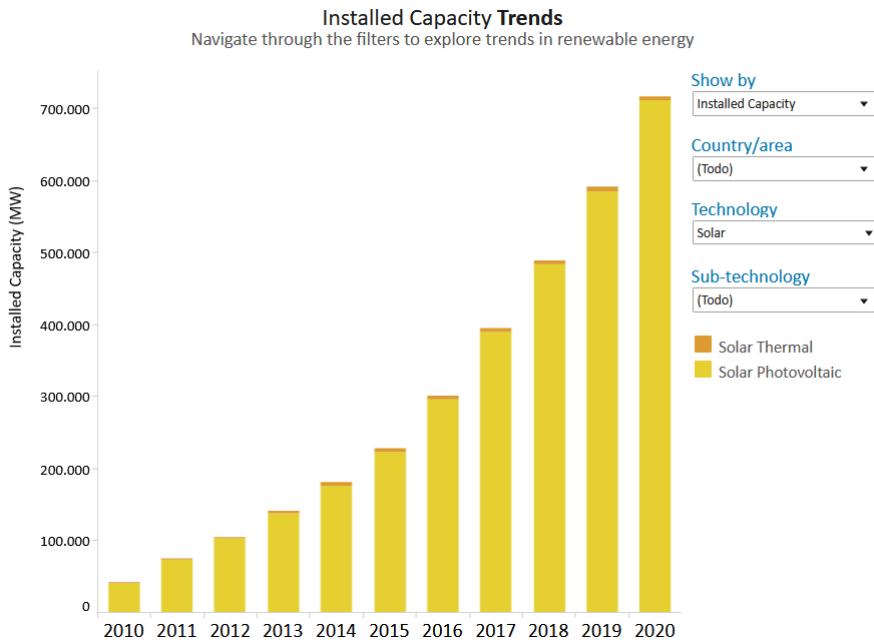


Figura 1.3: Comparación entre tecnología solar fotovoltaica y solar térmica, entre los años 2010 y 2020, a partir de la capacidad instalada (MW) a nivel mundial [2].

que producen [14].

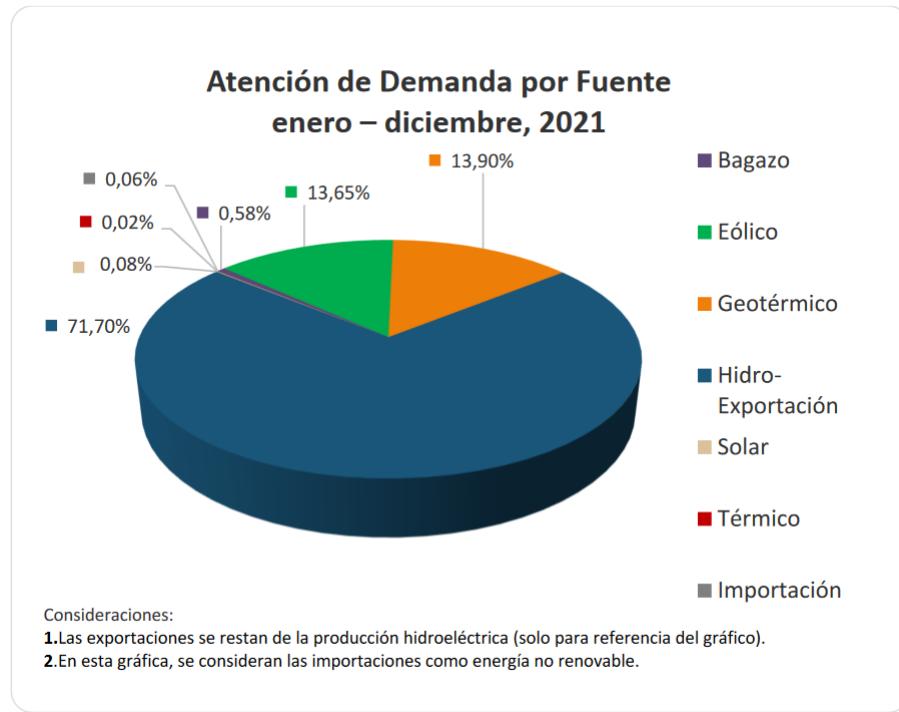


Figura 1.4: Matriz energética de Costa Rica en el año 2021. [3]

El VII Plan Nacional de Energía 2015-2030 establece que actualmente el país se encuentra enfocado en la diversificación de la matriz energética hacia las Energías Renovables No Convencionales (ERNC), como la solar y la eólica. Plantea, por ejemplo, que para el período 2024-2030 todas las viviendas no conectadas a la red dispongan de sistemas fotovoltaicos [13], y también se está apostando a los sistemas de generación distribuida, con ayuda de cooperativas locales generadoras de energía y empresas privadas. Ejemplos de esto son un proyecto generado a partir de cooperativas, y otro de carácter privado, que suministran energía con la colaboración del ICE: el primero está ubicado en San Carlos (Santa Rosa de Pocosol). Este proyecto, llamado Parque Solar Cooperativo, fue inaugurado en 2019 y está compuesto por 19.000 paneles solares en una propiedad de 11.2 hectáreas, con la intención de generar 5 MW y abastecer a unas 5000 familias de la zona [4]. Este proyecto se puede observar en la figura 1.5. El segundo proyecto, de iniciativa privada y a cargo de la empresa BMR Energy, es llamado Instalación Solar de Valle Escondido, se localiza en Bagaces, Guanacaste, y tiene previsto suministrar hasta 5 MW al ICE una vez finalizada su construcción [15].



Figura 1.5: Parque Solar Cooperativo, Pocosol de San Carlos. [4]

Dentro del mercado de los sistemas fotovoltaicos existe gran cantidad de tecnologías y técnicas para la generación de energía eléctrica a partir de la radiación del sol que incide sobre la tierra. Una de estas técnicas es la construcción de paneles solares bifaciales, donde se aprovechan ambas caras del panel (delantera y trasera) para generar electricidad. Estos paneles solares están avanzando rápidamente en el mercado, y debido a esto es necesario garantizar su confiabilidad. Parte de los procesos que utilizan los fa-

bricantes de estos paneles para asegurarle al consumidor cierta producción de energía es por medio de las simulaciones por computadora, sin embargo, estas implican modelados matemáticos complejos, los cuales deben ser verificados, probados y comparados.

Para el mejoramiento de las tecnologías, la colaboración internacional es fundamental. Ejemplo de esto es la colaboración entre el Instituto Tecnológico de Costa Rica y la Universidad de Ciencias Aplicadas de Anhalt (*Anhalt University of Applied Sciences*), la cual cuenta con varias sedes en distintas ciudades de Alemania. Ambas instituciones educativas intercambian entre sí información, datos y resultados, lo que permite el avance del conocimiento.

1.2. Definición del problema

1.2.1. Generalidades

Como se mencionó anteriormente, en la actualidad existen proyectos fotovoltaicos a gran escala (tanto a nivel nacional como internacional), y existe una creciente demanda por estos sistemas que se proyecta hacia las siguientes décadas. A la hora de plantear y definir estos proyectos, con cualquier tecnología fotovoltaica que se utilice, es necesario estimar la producción de energía que se puede realizar en el área de interés, para evaluar la viabilidad de dicho proyecto. Por lo general, entre mayor certeza se tenga sobre las características del proyecto y los réditos que este puede generar, mayor confianza producirá sobre las partes interesadas (inversionistas, usuarios, ingenieros, etc.). Otro aspecto que se debe considerar es la posibilidad de fallas, tanto eléctricas como debidas por sombras o ensuciamiento, las cuales pueden reducir la efectividad de los paneles o incluso dañarlos, causando pérdidas económicas y de tiempo, tanto por el mantenimiento requerido para reparar o reemplazar los paneles dañados, como por el tiempo de desconexión de estos, afectando tanto al proveedor como al consumidor. Para mejorar la predicción del comportamiento de los sistemas fotovoltaicos ante estos casos se pueden hacer simulaciones por computadora, lo que permite analizar estas situaciones sin necesidad de realizar prototipos o pruebas físicas que demoren más tiempo y requieran inversiones de capital, como el entorno de simulación propuesto en [16].

El mercado de los paneles solares fotovoltaicos abarca desde tecnologías con décadas de experiencia e implementación, hasta otras emergentes y aún en períodos de experimentación. Durante años, este mercado se ha enfocado principalmente en la producción y utilización de paneles solares monofaciales, los cuales solo tienen una cara expuesta a la luz solar (mientras la otra cara es opaca, usualmente con una protección bloqueando el paso de la luz), debido a su bajo costo y el mayor conocimiento de su funcionamiento en comparación con otras tecnologías. Sin embargo, en la actualidad hay un área que está resultando llamativa para las posibles partes interesadas en adquirir e invertir, a pesar de tener casi el mismo tiempo de existir que los sistemas monofaciales: paneles solares bifaciales. La tecnología bifacial permite generar electricidad tanto en la cara frontal como posterior de un panel, lo que habilita la obtención de más energía a casi el mismo precio que un sistema monofacial.

Para garantizar la confiabilidad y rentabilidad de un proyecto utilizando esta tecnología es necesario utilizar herramientas de simulación y evaluación, sin embargo, la precisión y validez de los modelos actuales aún necesitan un mayor desarrollo para garantizar resultados confiables, principalmente para la estimación de la energía que se puede generar con la cara trasera del panel, ya que esta cara por lo general no recibe la luz directa del sol, sino que depende de los varios parámetros que afectan la trayectoria e intensidad de la luz que llega a esta sección del panel. A pesar del interés por esta tecnología, uno de los principales desafíos es la falta de claridad respecto a la estimación real de energía que producen, esto debido a la dependencia del panel trasero a diferentes

factores como el albedo (porcentaje de radiación que cualquier superficie refleja respecto a la radiación que incide sobre ella), la altura del módulo respecto al suelo, el ángulo de inclinación, la reflectancia de la superficie sobre la que se encuentra, la elevación del sol y la irradiación difusa [9].

De acuerdo con [9], existen tres técnicas principales para la simulación de potencia en la cara trasera de paneles bifaciales: modelos empíricos, modelos de factor de vista (*view factor models*), y modelos de trazado de rayos (*ray tracing models*), para cada una de estas técnicas existen diferentes pruebas y modelados que se pueden realizar. Los resultados de estos modelados y estimaciones pueden ser variables, por lo que es necesario evaluar y hacer comparaciones entre estos resultados para generar estimaciones más confiables, lo que puede ayudar a asegurar la viabilidad y rentabilidad de un proyecto utilizando paneles bifaciales. Estos entornos simulados son escasos, lo que limita y retrasa la cooperación científica y el avance de estas tecnología.

1.2.2. Síntesis del problema

Falta de una plataforma para la evaluación de distintas técnicas de modelado para la radiación trasera en los módulos bifaciales.

1.3. Enfoque de la solución

Las diferentes herramientas de software existentes, utilizadas para la estimación energética en paneles bifaciales (por ejemplo, las desarrolladas por Prism Solar [17], Solar World [18], PV Syst [19], Purdue University [20], NREL Bifacial VF [21], entre otras) ofrecen la posibilidad de realizar simulaciones y obtener información fundamental para definir la viabilidad de un proyecto fotovoltaico. Sin embargo, usualmente al realizar simulaciones con estas herramientas, y comparar sus resultados contra mediciones físicas realizadas in-situ, se presentan variaciones considerables y errores que podrían significar una perdida de confiabilidad, si se compara con la estandarización de estas simulaciones y estimaciones para el sector fotovoltaico monofacial. Adicionalmente, en la actualidad no se cuenta con algún estándar para generar estimaciones en paneles bifaciales, aunque se trabaja con algunas normativas que incluyen parcialmente dichas instalaciones (como el IEC 60904 [22]), lo que dificulta acelerar la transición a este tipo de sistemas.

Entre los principales desafíos actualmente para los sistemas bifaciales, se encuentra el mejoramiento de la precisión, exactitud y confiabilidad de las estimaciones energéticas a partir de herramientas de software. Por esto, se busca trabajar sobre este desafío mediante la creación de una herramienta de evaluación y validación (mediante software) que permita comparar dichas estimaciones contra mediciones reales, en un ambiente centralizado, flexible, dinámico e intuitivo, a través de gráficas e información fácilmente visualizable, para así generar reportes más robustos y facilitar la comprensión de la información de los simuladores fotovoltaicos bifaciales.

1.4. Objetivos

1.4.1. Meta

Aportar al proceso de modelado de la radiación trasera para sistemas fotovoltaicos bifaciales mediante el desarrollo de una herramienta de simulación que permita comparar y evaluar la precisión y exactitud de modelos de estimación energética previamente simulados.

1.4.2. Objetivo general

- Desarrollar una herramienta de evaluación para modelos de simulación de estimación energética de paneles solares bifaciales (previamente generados) mediante Python, con la finalidad de facilitar la selección del modelo más exacto disponible.

1.4.3. Objetivos específicos

- Diseñar un método para recopilar, ordenar y clasificar las simulaciones de estimación energética para paneles bifaciales.
- Desarrollar el método para comparación de medición de métodos de estimación energética para paneles bifaciales, utilizando algoritmos de Python.
- Validar la herramienta desarrollada con la incorporación de datos reales.

1.5. Estructura del documento

El documento está estructurado de la siguiente manera:

- Capítulo 1: Este capítulo sirve como introducción para el documento. Aquí se da una visión general de las razones para desarrollar el proyecto, así como los objetivos que se quieren conseguir.
- Capítulo 2: Presenta algunas definiciones necesarias para una mayor comprensión de las distintas partes del proyecto.
- Capítulo 3: Aquí se detalla la arquitectura de la herramienta diseñada, así como una explicación sobre las herramientas de simulación complementarias.
- Capítulo 4: Presenta las entradas y salidas del programa, se realiza una demostración de uso y se validan los resultados mediante la incorporación de datos reales de referencia.
- Capítulo 5: Por último, este capítulo presenta las conclusiones y recomendaciones del trabajo realizado.

Capítulo 2

Fundamentos teóricos

El presente capítulo pretende brindar las definiciones y conceptos teóricos necesarios para una mejor comprensión del desarrollo del proyecto. Como parte de esto se habla sobre los sistemas fotovoltaicos en general, los paneles solares monofaciales y bifaciales, descripción de los simuladores de estimación energética, se hace una comparación de las herramientas que se planteó utilizar en el desarrollo del proyecto, y se mencionan conceptos básicos sobre el lenguaje de programación Python.

2.1. Sistemas fotovoltaicos

Los sistemas solares basados en la tecnología fotovoltaica son aquellos con la capacidad de convertir directamente la radiación proveniente del Sol en electricidad. Este proceso se realiza por medio de la celda solar, la cual es la unidad básica de estos sistemas. Esta energía solar fotovoltaica puede ser utilizada en una amplia gama de aplicaciones según las necesidades, principalmente en la generación de energía para las redes eléctricas centralizadas, configuraciones que complementan el uso de energía autogenerada y conexión a los sistemas eléctricos centrales (sistemas fotovoltaicos conectados a la red), o para satisfacer la demanda energética de sistemas desconectados de dicha red, en combinación con baterías recargables (sistemas fotovoltaicos autónomos) [23].

Un generador fotovoltaico es el encargado de transformar la energía del Sol en energía eléctrica mediante el efecto fotoeléctrico (emisión de electrones por un material al incidir sobre él una radiación electromagnética). La unidad básica de este generador es la celda fotovoltaica, que al usar varias unidades se pueden interconectar para formar un módulo o panel fotovoltaico. Si se conectan varios módulos, ya sea en serie o paralelo, se forma una cadena fotovoltaica. Y al unir varias de estas cadenas, se tendrá un grupo o matriz fotovoltaica. Esta jerarquía de elementos de un generador fotovoltaico se puede ver ejemplificada en la figura 2.1.

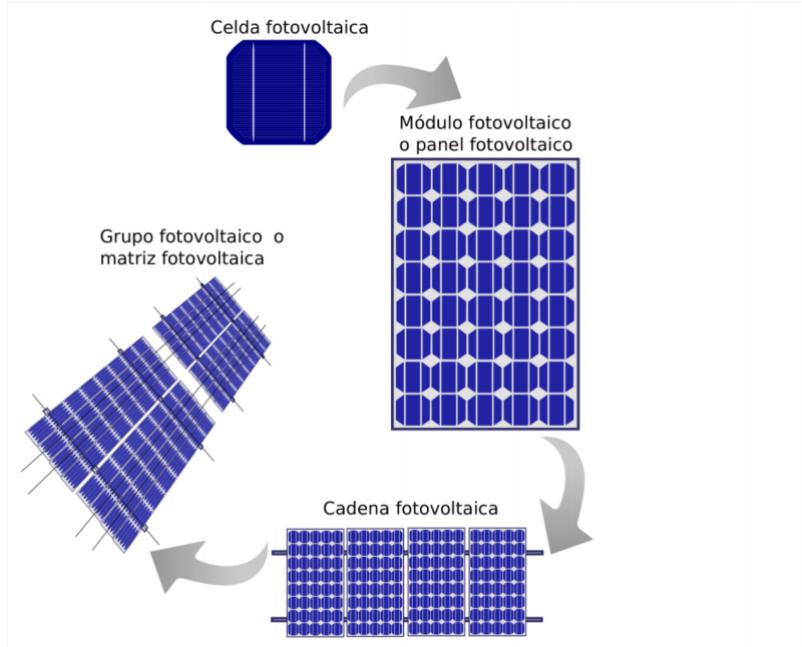


Figura 2.1: Organización de un generador fotovoltaico [5]

La forma de fabricar las celdas fotovoltaicas afecta su eficiencia para aprovechar la radiación solar. El material más utilizado para crear estas celdas es el silicio, el cual representa un 95 % de los módulos producidos hoy en día [24]. En el mercado actual, los principales métodos de fabricación permiten crear celdas de tipo monocristalino (constituidas por un único cristal de silicio con estructura uniforme), policristalino (formadas por muchos cristales de silicio) y celdas amorfas de capa fina (obtenidas al depositar silicio sobre una base de vidrio, plástico u otro material). Generalmente, estas celdas tienen diferencias visuales a simple vista, como se aprecia en las figuras 2.2 y 2.3. Existen otros materiales y técnicas de fabricación de celdas solares, como por ejemplo las hechas con compuestos de diferentes elementos como cadmio y telurio (CdTe), o cobre, indio, galio y selenio (CIGS). También se producen celdas solares a partir de perovskita, compuestos orgánicos, y con estructuras llamadas puntos cuánticos (partículas de diferentes materiales semiconductores con un tamaño de unos cuantos nanómetros), entre otros materiales, tecnologías y técnicas de fabricación.

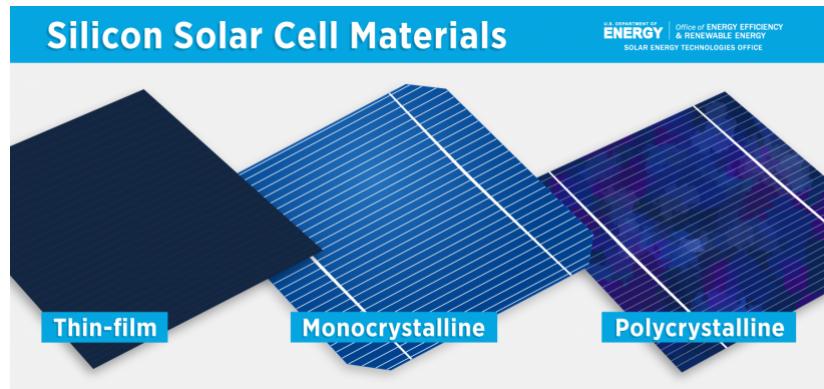


Figura 2.2: Tipos de celdas solares de silicio: Capa fina, monocristalina y policristalina [6]

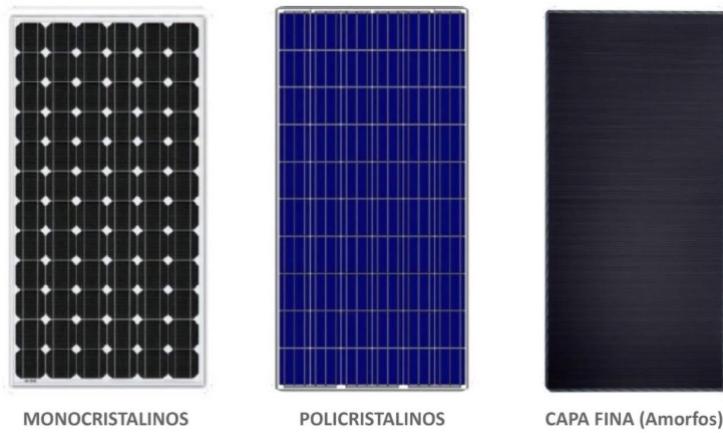


Figura 2.3: Tipos de panel solar fotovoltaico según fabricación (silicio) [7]

El comportamiento eléctrico de un panel o módulo fotovoltaico se puede representar mediante la curva característica de corriente y tensión (I-V). Este corresponde a condiciones ambientales concretas y normalizadas: radiación solar de 1000 W/m^2 y una temperatura ambiente de 25°C , a pesar de que estas condiciones son muy variables según la ubicación geográfica y su clima [7]. La curva I-V se puede observar en la figura 2.4.

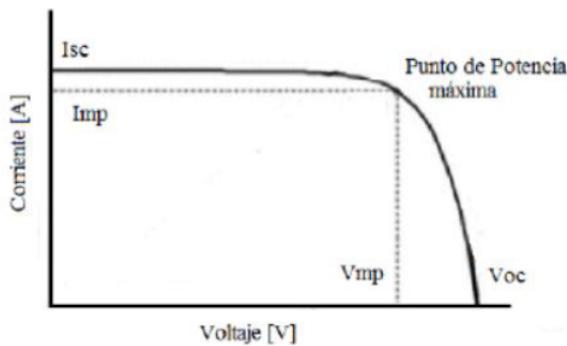


Figura 2.4: Curva característica I-V de un panel fotovoltaico [7]

En esta curva participan los siguientes parámetros:

- Corriente de cortocircuito (I_{sc}): Corriente máxima medida en el panel cuando la tensión de salida es cero.
- Corriente en el punto de máxima potencia (I_{mp}): Valor de la corriente cuando la potencia producida por el panel es la máxima.
- Voltaje de corto circuito (V_{oc}): Tensión máxima medida en el panel cuando la corriente es cero.
- Voltaje en el punto de máxima potencia (V_{mp}): Valor del voltaje cuando la potencia producida por el módulo es la máxima.
- Potencia máxima: Máxima potencia que produce el panel bajo condiciones determinadas de iluminación y temperatura.

Las condiciones ambientales hacen que el funcionamiento de un panel fotovoltaico varíe, modificando la curva característica I-V. Entre mayor sea la temperatura ambiente (y por ende, la temperatura interna de cada celda), menores valores de voltaje se alcanzan, como se puede apreciar en la figura 2.5. De manera similar, la irradiancia condiciona la corriente generada por el panel: a mayor irradiancia, mayor corriente (figura 2.6).

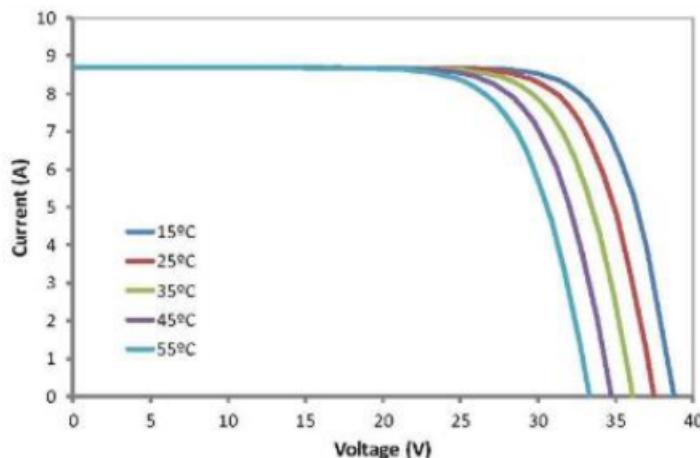


Figura 2.5: Curva característica I-V según la temperatura [7]

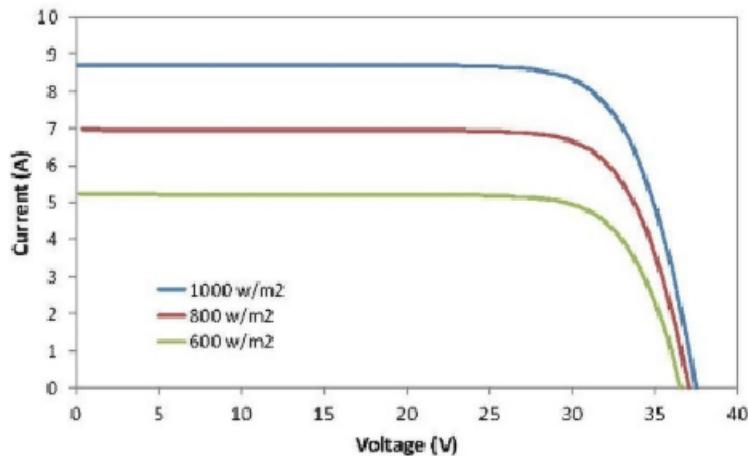


Figura 2.6: Curva característica I-V según la irradiancia [7]

Además de los materiales y técnicas de fabricación, los paneles solares se pueden clasificar en monofaciales y bifaciales. Ambos comparten casi la misma construcción, pero sus diferencias son lo suficientemente significativas como para trabajar de forma diferente con ambos tipos de sistemas.

2.1.1. Paneles monofaciales

Los paneles solares monofaciales han sido los más utilizados en el mercado durante años debido a que su funcionamiento es bien conocido y comprendido. Estos paneles solamente producen energía por una de sus caras, donde intentan aprovechar la mayor cantidad de longitudes de onda mediante las celdas solares, recubrimientos de vidrio y otros materiales reflectantes y refractantes. Por lo general, los paneles monofaciales están constituidos por un marco de aluminio que permite el montaje sobre un soporte, un recubrimiento exterior delantero de cristal templado, un encapsulado de las celdas que les da protección adicional contra los elementos, las celdas solares, una cubierta posterior que brinda protección y puede reflejar parte de la radiación, y una caja de conexiones eléctricas. Un ejemplo de esta construcción se puede observar en la imagen (a) de la figura 2.7. Estos paneles también se pueden construir sin un marco de aluminio (llamados *frameless*), e incluso cambiando la cubierta posterior por vidrio, como la imagen (b) de la figura 2.7.

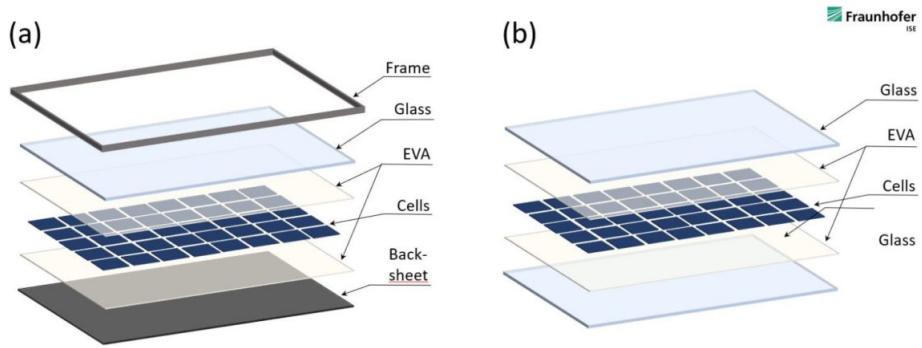


Figura 2.7: Panel solar monofacial con marco y recubrimiento metálico trasero (a) y sin marco y recubrimiento trasero de vidrio (b) [8]

2.1.2. Paneles bifaciales

Los paneles solares bifaciales son aquellos capaces de producir electricidad usando tanto la cara delantera como la trasera. Generalmente pueden producir más energía y tienen una mayor durabilidad debido a la resistencia de ambas caras a la radiación UV, y si se construyen sin un marco metálico, se pueden reducir efectos como la degradación inducida por potencial (PID) [25]. En un panel bifacial la radiación directa y difusa reflejadas por la superficie bajo el panel pueden ser aprovechadas, mientras que en un panel monofacial esta radiación se refleja al ambiente. Esto se puede ver ilustrado en la figura 2.8

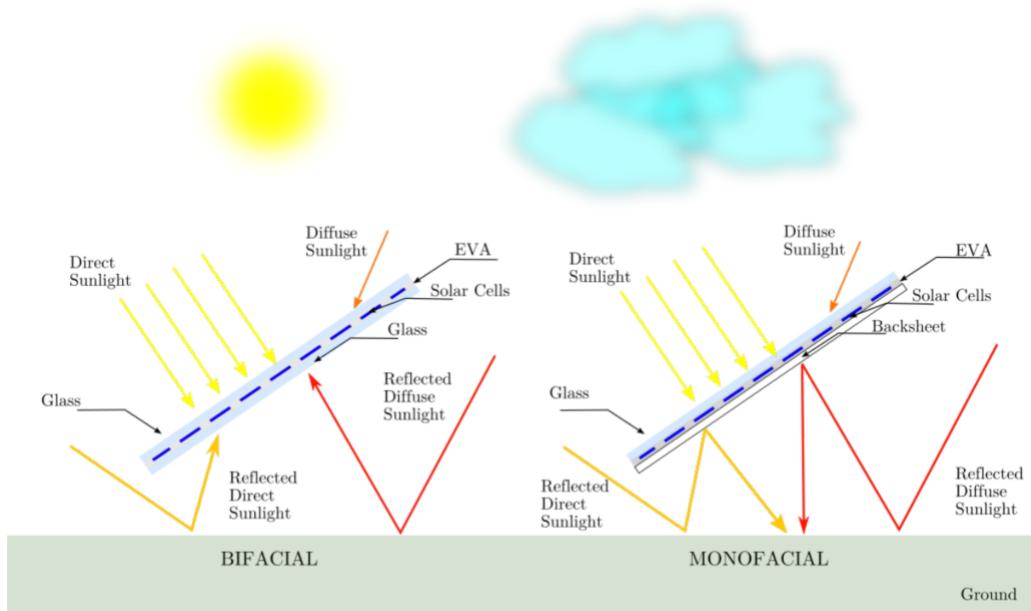


Figura 2.8: Comparación entre un modulo bifacial y monofacial [9]

En su estructura y construcción son muy similares a los monofaciales, algunos tienen marco, otros no, pueden tener panel trasero de vidrio o de algún polímero claro, pero siempre tienen en común la producción de electricidad en ambas caras y un recubrimiento trasero que permita el paso de la luz. Pueden existir paneles sin marco, con ambos recubrimientos externos de vidrio para exponer las celdas a la radiación, pero ser monofaciales. Lo que permite la producción de energía en ambas caras es una pequeña diferencia en la construcción de sus celdas: las celdas bifaciales tienen contactos/conectores para conducir la electricidad en ambas caras de su superficie, mientras que las monofaciales solo los tienen en una, como se puede apreciar en la figura 2.9.

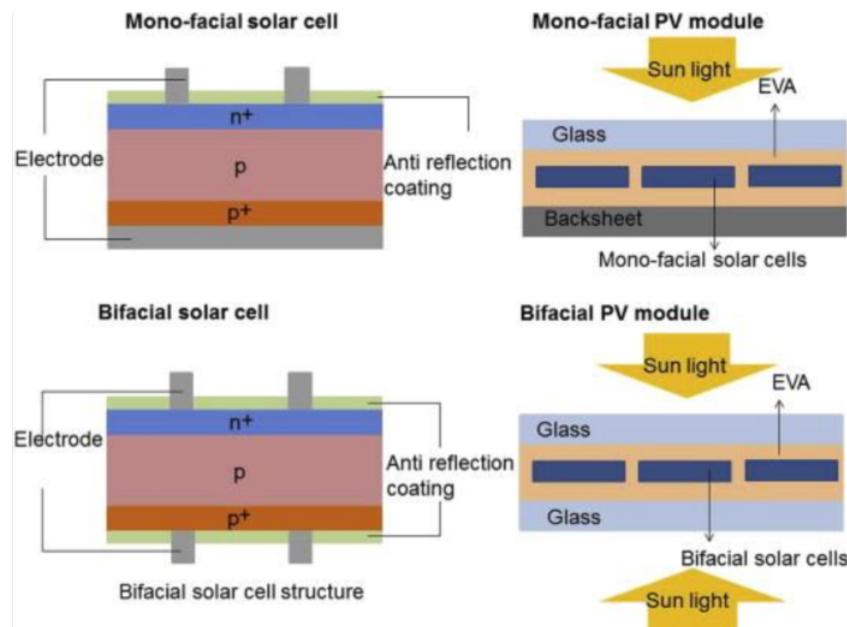


Figura 2.9: Estructura de una celda monofacial y una celda bifacial [10]

2.1.3. Mercado para módulos bifaciales

De acuerdo con el *International Technology Roadmap for Photovoltaic (ITRPV)* [11], se espera que el mercado mundial para las tecnologías bifaciales tenga un crecimiento significativo en la próxima década, tanto a nivel de módulos como de celdas. La figura 2.10 muestra que para el año 2021, del mercado total de sistemas fotovoltaicos el sector correspondiente a módulos bifaciales con celdas bifaciales correspondía aproximadamente al 30 %, mientras que para el año 2032 se proyecta que este aumente a poco más del 60 %, desplazando a las tecnologías monofaciales, las cuales lideran el mercado en la actualidad. Estas proyecciones consideran que estos módulos bifaciales serán utilizados principalmente en instalaciones de plantas generadoras.

Los módulos monofaciales se pueden adaptar para utilizar celdas fotovoltaicas bifaciales, y este mercado también espera un incremento. Se proyecta que el uso de módulos monofaciales con celdas bifaciales aumente entre un 20 % y 30 % en los próximos 10 años.

Esto demuestra que el mercado global está volteando la mirada hacia las tecnologías bifaciales.

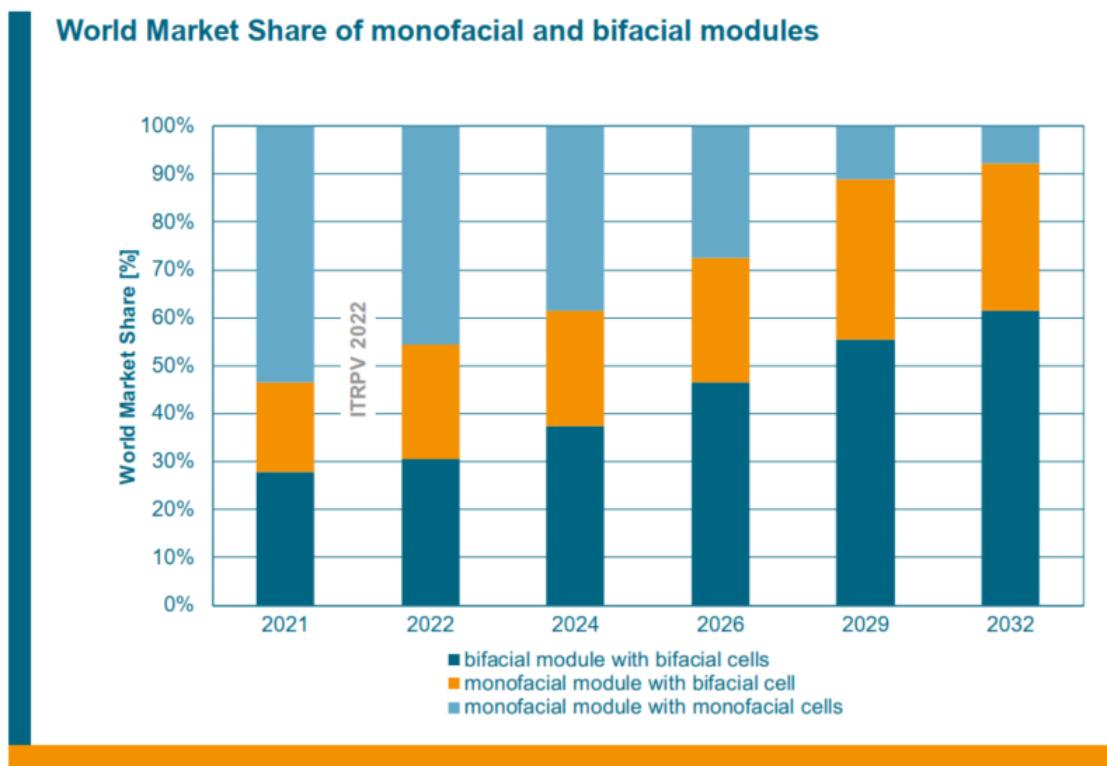


Figura 2.10: Sectorización del mercado mundial para módulos monofaciales y bifaciales [11]

Como se puede apreciar, la importancia de la tecnología bifacial en el mercado energético va en aumento, por lo que es vital generar confianza en las inversiones para este sector. Actualmente siguen existiendo desafíos para mejorar la predicción de las herramientas de simulación en módulos bifaciales, por lo que se sigue trabajando para generar mejoras que permitan identificar las oportunidades de obtener la mayor generación energética al menor costo, y así reducir el riesgo y la incertidumbre para los inversionistas. Según [9], los principales obstáculos para mejorar la simulación de sistemas bifaciales son la falta de estandarización y la falta de *feedback* por parte de sistemas grandes ya existentes. Para resolver ambos es necesaria la colaboración científica.

La generación energética en la cara posterior de un módulo bifacial, como se mencionó anteriormente, depende de muchos factores como la altura del módulo respecto al suelo, albedo, ángulo de inclinación y *azimuth*, y todas estas variables pueden fluctuar según la ubicación geográfica, época del año, tipo de suelo y decisiones de construcción e instalación de los módulos y los *arrays* fotovoltaicos. La mayor parte de las herramientas de simulación existentes en la actualidad fueron diseñadas para sistemas monofaciales, por lo que se encuentran con desafíos para incorporar todas estas variables al estimar

la producción energética en casos bifaciales. Por esta razón es muy importante trabajar sobre estas barreras, y así acelerar la adopción de estas tecnologías.

Respecto a la falta de estandarización adecuada, actualmente la mayoría de estándares internacionales utilizados por la comunidad industrial y científica fueron diseñados para sistemas monofaciales y no consideran la producción energética en la cara trasera, como el IEC 61724 [26] y el IEC 61853 [27], utilizados para evaluación y monitoreo en exteriores. Existe mayor avance en la evaluación en interiores de módulos bifaciales, como el estándar IEC 60904 [28], y se encuentra en desarrollo un nuevo estándar de IEC para celdas y módulos bifaciales [29].

Como menciona [9], la creación y el mejoramiento de estándares para evaluación, simulación y monitoreo de sistemas bifaciales solo es posible si los fenómenos están bien estudiados. Para esto es necesario que las comunidades industrial y científica trabajen en conjunto para mejorar la precisión de herramientas de simulación y evaluación. El presente trabajo tiene esta intención en mente, ya que podría ser utilizado para estudios posteriores de sistemas fotovoltaicos en metodologías de evaluación *round-robin*, en conjunto con otras instituciones o empresas, tomando como ejemplo el estudio realizado en [30], donde se hace una colaboración internacional para comparar mediciones y simulaciones realizadas en diferentes laboratorios en China, Italia, Suiza, Japón y Alemania. La herramienta planteada en este proyecto no pretende competir contra otras herramientas existentes en el ámbito fotovoltaico bifacial, sino más bien complementar sus funciones, llenar algunos vacíos y colaborar con el proceso de implementación y aceptación de tecnologías fotovoltaicas.

2.2. Simuladores de estimación energética

La tarea de crear modelos para la estimación energética de paneles bifaciales ha sido estudiada por diferentes grupos de investigación. En general, el proceso de crear este tipo de simulaciones sigue un flujo de modelado, variando un poco según cada herramienta. Una forma de ver este flujo se puede apreciar en la figura 2.11, pero aún queda mucho trabajo por hacer.

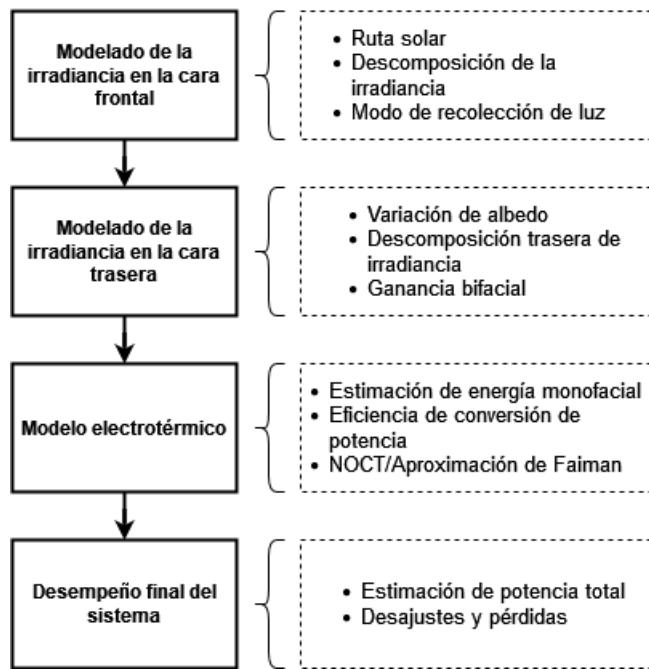


Figura 2.11: Flujo general de modelado de estimación energética en paneles bifaciales (adaptado de [9])

Como primer paso se realiza el modelado de la irradiancia en la cara frontal, que es el método más común utilizado, debido a que está completamente estudiado y analizado para la estimación de energía en módulos monofaciales, y es un proceso estandarizado. A partir de este modelado se obtiene la trayectoria del Sol, las irradiancias directas, difusa e irradiancia de albedo.

Seguidamente, se estiman los efectos que tiene la irradiancia reflejada (directa, difusa y albedo) sobre el modelado en la cara trasera. Aquí se obtiene como resultado una ganancia bifacial relativa.

En la creación del modelo electrotérmico se estima la energía producida por el módulo monofacial, y además se obtiene como resultado una conversión de potencia, la eficiencia energética del módulo, las pérdidas térmicas, y la temperatura de operación nominal de

la celda (NOCT, por sus siglas en inglés).

Por último se hace un análisis del desempeño final del sistema. Aquí se aplican las correcciones finales de acuerdo con las suposiciones hechas en los modelos anteriores.

Cada uno de estos modelos tiene diferentes aproximaciones o algoritmos para cada paso, sin embargo, la mayor diferencia se encuentra en el modelado de la irradiancia en la cara trasera del panel. Este es uno de los aspectos más desafiantes en la estimación energética de paneles bifaciales, ya que en la cara trasera influyen más factores que en la delantera.

Entre los factores de los que depende la salida energética de la cara trasera de los paneles bifaciales hay elementos relacionados con su montaje, como la elevación del panel respecto al suelo, su orientación y el ángulo de inclinación, así como otros relacionados con las condiciones ambientales, como la intensidad de la radiación recibida, el albedo del suelo y la temperatura.

Repasando lo mencionado en la sección 1.2.1, las principales técnicas de modelado que se encuentran en la literatura se pueden clasificar en tres categorías: *modelos empíricos*, de *factor de forma*, y de *trazado de rayos*. Un *modelo empírico* se puede definir como un modelo matemático creado a partir de experimentos y mediciones. A partir de este se puede crear una especie de “ecuación de caja negra”, la cual da una idea de cómo funciona el fenómeno en estudio, pero no tanto así el “por qué”. Entre las principales ventajas se que pueden encontrar al utilizar este tipo de modelado es que los resultados pueden ser simples ecuaciones matemáticas, dar una idea general del comportamiento del objeto de estudio, y la posible facilidad de implementación en un sistema computacional.

Por otra parte, también presentan desventajas, como la dependencia que pueden tener los modelos de sus experimentos (es decir, el modelo no se puede adaptar a todas las diferentes posibilidades), que estos modelos no explican detalladamente cómo trabaja un sistema, y que debido a las limitaciones en los experimentos algunos parámetros importantes podrían quedar excluidos del modelo [9]. Entre los principales *modelos empíricos* para la estimación energética en la cara trasera de paneles solares bifaciales se encuentra el creado por el grupo Solar World. Este modelo será explicado más a detalle en el siguiente capítulo.

Los *factores de forma* estiman, usando consideraciones geométricas, la fracción de la irradiancia dispersa o reflejada desde las superficies adyacentes hasta la parte trasera del módulo fotovoltaico. Es uno de los métodos más estudiados, sus modelos tienen una complejidad computacional y una velocidad de procesamiento medias. Entre las ventajas de su uso se encuentra la relativa facilidad de implementación en sistemas computacionales, la capacidad de proveer más información sobre la irradiancia reflejada (en comparación con los *modelos empíricos*), y que estos modelos se adaptan muy bien a las configuraciones individuales. Por el contrario, entre sus desventajas se encuentra la falta de

descripción de los efectos de dispersión o difusión, el hecho de que las configuraciones complejas pueden complicar el análisis geométrico, y que el modelado se debe realizar para cada celda individual [9]. Entre los casos de estudio y aplicación más importantes sobre este tipo de modelado se encuentra la herramienta PVsyst, sobre la cual se explicará en el capítulo siguiente.

Por último se tiene el modelado por *trazado de rayos*. A diferencia de los otros dos métodos, este ofrece la posibilidad de recrear escenarios complejos de simulación para la evaluación de sistemas fotovoltaicos bifaciales. Estos escenarios pueden incluir las configuraciones de montaje y los efectos del autosombreado, así como variaciones en el módulo solar como la distancia entre las celdas, características del recubrimiento de vidrio, y otros efectos ópticos. Sin embargo, al tomar en cuenta muchos más aspectos que los modelos anteriores, esto se ve reflejado en una mayor demanda de recursos computacionales y tiempo de simulación. Entre las ventajas de usar el modelado por *trazado de rayos* está la posibilidad de crear estimaciones más realistas, una mejor resolución respecto a los efectos del autosombreado, y la adaptación a configuraciones más complejas.

Sin embargo, entre sus desventajas está la gran demanda computacional, que la complejidad alta de estos modelos puede hacer que no sean aptos para simular un panel individual, y su dificultad general de implementación [9]. Uno de sus casos de aplicación más importantes es el software creado por el Laboratorio Nacional de Energías Renovables de los Estados Unidos (NREL), el cual se explicará en el capítulo siguiente.

2.3. Comparación de herramientas

Para el desarrollo de la herramienta evaluativa se decidió trabajar con el lenguaje de programación Python, por encima de otros lenguajes y herramientas disponibles para desarrollar evaluaciones de sistemas fotovoltaicos. Las opciones principales que se manejan como alternativas a Python (solución 1) son Matlab (solución 2) y C++ (solución 3).

En general, se estableció que es ideal que la herramienta se pueda desarrollar con un presupuesto bajo, y por este motivo fue preferible el uso de una plataforma de código abierto. Adicionalmente, se deseó desarrollar una herramienta de evaluación flexible, veloz, ágil e intuitiva, que tome datos y archivos generados a partir de simulaciones y estimaciones energéticas realizadas por otras herramientas de software, para así generar gráficas y comparaciones entre modelos. Además, en busca de facilidad de uso para el usuario, se consideró deseable la capacidad de generar una interfaz gráfica de usuario (GUI).

Para realizar este análisis y comparación entre las soluciones propuestas se generó una matriz de Pugh. Se establecieron diversos criterios para la evaluación, así como un peso (entre 1.5 y 3) según la importancia que tienen para el desarrollo de la herramienta:

- Facilidad de acceso a recursos fotovoltaicos (**Peso = 3**): Este criterio corresponde a la disponibilidad *online* de bibliotecas, herramientas y *toolboxes* compatibles, si son gratuitas o de pago.
- Capacidad de entrelazarse con otras herramientas o lenguajes de programación (**Peso = 3**): Corresponde a la capacidad del lenguaje o herramienta de trabajar con datos de entrada provenientes de diferentes fuentes.
- Facilidad de graficación (**Peso = 3**): Se considera como la disponibilidad de recursos para graficación, así como qué tan sencillo es utilizarlos para personas con diferentes grados de experiencia en programación.
- Código abierto (**Peso = 2**): Considera si la herramienta o lenguaje de programación se maneja dentro de las normas de código abierto, lo que se puede traducir en una reducción de costos económicos.
- Demanda de capacidad de cómputo (**Peso = 2**): Corresponde a la demanda de recursos de memoria, energía y tiempo sobre un computador (portátil o de escritorio) por parte del lenguaje o herramienta.
- Facilidad para crear una interfaz gráfica de usuario (GUI)(**Peso = 1.5**): Considera la disponibilidad de bibliotecas, *toolboxes* o herramientas para la generación de una interfaz interactiva para el usuario, y la facilidad de su implementación para el desarrollador.

Para esta matriz se utilizó como referencia la propuesta de solución 3 (lenguaje C++). Las propuestas se evaluaron con un valor de +1 si se presentan como una mejor opción, un 0 si es igual, y un -1 si representa un peor opción respecto a la referencia. El siguiente cuadro (2.1) presenta la matriz de Pugh considerando los pesos atribuidos a cada característica. Estos pesos se multiplican por el valor asignado de +1, 0 o -1.

Cuadro 2.1: Matriz de Pugh (considerando los pesos)

Criterios	Pesos	Solución 1	Solución 2	Solución 3
Facilidad de acceso a recursos fotovoltaicos	3	+3	-3	=
Capacidad de entrelazarse con otras herramientas o lenguajes de programación	3	+3	+3	=
Facilidad de graficación	3	+3	+3	=
Código abierto	2	+2	-2	=
Demanda de capacidad de cómputo	2	0	-2	=
Facilidad para crear una interfaz gráfica de usuario (GUI)	1,5	+1,5	+1,5	=
Suma positiva		+12,5	+7,5	0
Suma negativa		0	-7	0
Suma total		+12,5	+0,5	0

Al analizar lo generado por la matriz de Pugh, es posible observar que la solución 1 (Python) resultó ser la más viable para el desarrollo de este proyecto. Esto se debe, principalmente, al ser un lenguaje de código abierto, gratuito, flexible y con gran disponibilidad de recursos.

Otra opción considerada dentro de las soluciones es Scilab (software para análisis numérico, con un lenguaje de programación de alto nivel para cálculo científico, desarrollado por Scilab Enterprises). Sin embargo, esta no se comparó a la par de las otras opciones debido a que varias de las bibliotecas para el modelado de paneles solares se encuentran aún en proceso de desarrollo.

Un aspecto que vale la pena mencionar respecto a la consideración de C++ en la selección de las soluciones, es que la mayoría de las herramientas de software que utilizan técnicas de modelado por trazadores de rayos están desarrolladas en este lenguaje. Sin embargo, Python demostró tener características más deseables para esta aplicación en particular.

2.4. Python

Python es un lenguaje de programación de propósito general de alto nivel y de código abierto (su código fuente y otros derechos que normalmente son exclusivos para quienes poseen los derechos de autor, son publicados bajo una licencia de código abierto o forman parte del dominio público), el cual es utilizado en aplicaciones como ciencia de datos, desarrollo de software y web, automatización y visualización de datos, cuya filosofía de diseño se enfoca en la legibilidad y en una sintaxis que permita definir conceptos mediante pocas líneas de código (en comparación con otros lenguajes de programación similares). Es uno de los lenguajes de programación más populares en la actualidad, debido a que es amigable con los programadores principiantes, pero al mismo tiempo ofrece versatilidad para veteranos [31]. Fue desarrollado por Guido Van Rossum, en 1991. Este lenguaje soporta múltiples paradigmas de programación, incluyendo orientación a objetos, programación funcional e imperativa, entre otros. Se caracteriza, además, por la capacidad de utilizar elementos modulares diseñados en otros lenguajes de programación. Presenta un sistema de tipo dinámico, administración automática de memoria y tiene una biblioteca estándar extensa y variada. [32]

Este lenguaje mantiene una comunidad activa de desarrolladores, tanto profesionales como aficionados, que constantemente crea nuevas bibliotecas, herramientas y funcionalidades. Entre estas bibliotecas existen algunas diseñadas especialmente para la simulación y visualización de datos en sistemas fotovoltaicos, como por ejemplo pvlib python [33], y otras enfocadas en la creación de gráficas a partir de datos contenidos en listas o *arrays*, como Matplotlib [34]. Adicionalmente, Python puede trabajar de forma cruzada con otras herramientas de software y lenguajes de programación.

Estas características son de gran utilidad para el desarrollo del proyecto, ya que dan una mayor claridad al desarrollador, a otras personas que deseen realizar cambios o adaptaciones al código, y a los usuarios finales del programa.

Las secciones desarrolladas en este capítulo ayudan a la comprensión general de la estructura de los sistemas fotovoltaicos, la diferencia entre los módulos monofaciales y bifaciales, se habla de la penetración de las tecnologías bifaciales en el mercado global, los desafíos presentes para la incorporación de estas tecnologías, algunas herramientas de software utilizadas para la estimación energética de sistemas bifaciales, se desarrolla la justificación del uso del lenguaje Python para el desarrollo de una nueva herramienta de evaluación, y se mencionan características y otros aspectos importantes sobre este lenguaje de programación. En el siguiente capítulo se explicarán detalladamente los elementos utilizados para la creación de la herramienta de evaluación de este proyecto.

Capítulo 3

Arquitectura de herramienta de evaluación

En el presente capítulo se detallan los modelos de estimación de energética de sistemas fotovoltaicos elegidos como base para el desarrollo de la herramienta de evaluación. Adicionalmente, se explica la arquitectura implementada en la creación de la herramienta, las partes necesarias para su funcionamiento, y los escenarios simulados mediante las herramientas de software complementarias.

3.1. Modelos de estimación seleccionados

Como se menciona en la sección 2.2, entre los modelos de estimación energética en sistemas bifaciales más utilizados se encuentran los modelos empíricos, de factor de forma y de trazado de rayos. Diversas organizaciones y grupos en el área de sistemas fotovoltaicos (mencionados en la sección 1.3) se han dedicado a diseñar métodos y herramientas de software para calcular y simular diferentes parámetros relacionados con la estimación energética de paneles bifaciales. Para el desarrollo de la herramienta de evaluación de este proyecto se decidió trabajar con un método por cada modelo mencionado, basado en su frecuencia de utilización y distintos niveles de complejidad, con el objetivo de abarcar diferentes aproximaciones de la estimación energética. Para el modelo empírico se eligió el método desarrollado por *Solar World*, para el factor de forma se seleccionó el software *PVSyst*, y para el trazado de rayos se decidió trabajar con la herramienta *Bifacial Radiance* (creada por NREL).

3.1.1. Solar World

Este modelo empírico fue propuesto al analizar los resultados de varias simulaciones experimentales de trazado de rayos utilizando el software Radiance [35], junto con la base de datos meteorológica Meteonorm. Utiliza como entradas el albedo, la altura normalizada del panel respecto al suelo (*normalized clearance height*), y la proporción de cobertura del suelo (*ground coverage ratio*), para estimar la ganancia bifacial energética (*bifacial gain in energy [BGE]*) para un módulo fotovoltaico, la cual es una relación entre la producción energética de un panel bifacial respecto a un panel monofacial.

El modelo propuesto está representado por la ecuación 3.1, donde α representa el albedo, φ_{Pmp} es el factor de bifacialidad del módulo, s es un factor de consideración por efectos de sombreado en la cara posterior del panel, a , b y c son factores empíricos, h es la *clearance height*, que representa la distancia entre el punto más bajo del módulo y

la superficie donde este se encuentra, y gcr es el *ground coverage ratio*, el cual consiste en una relación entre el área del arreglo fotovoltaico y el área de superficie donde están instalados los paneles.

$$BGE[\%] = \alpha * \varphi_{Pmp} * s [a * (1 - \sqrt{gcr}) (1 - e^{b*h*gcr}) + c * (1 - gcr^4)] \quad (3.1)$$

Tras un proceso de optimización, se sustituyen los factores s , a , b y c para dar como resultado la ecuación 3.2.

$$BGE[\%] = \alpha * \varphi_{Pmp} * 0,95 [1,95 * (1 - \sqrt{gcr}) (1 - e^{8,691*h*gcr}) + 0,125 * (1 - gcr^4)] \quad (3.2)$$

3.1.2. PVsyst

PVsyst es una herramienta de software desarrollada por la Universidad de Ginebra en 1992. Implementa un modelo de factor de vista al aplicar la misma metodología utilizada en la estimación energética en módulos monofaciales, para posteriormente agregar la energía producida por la parte trasera de un panel bifacial (al considerar la radiación directa, difusa y reflejada por el suelo), y así obtener la estimación energética total. Este software presenta la posibilidad de simular diferentes escenarios basados en ubicación geográfica, distintas bases de datos meteorológicas, configuraciones de inclinación, altura y orientación de los módulos, considerar rastreadores solares, así como incluir una amplia biblioteca de modelos comerciales y genéricos de paneles solares e inversores, entre otras funcionalidades. Adicionalmente, ofrece la posibilidad de exportar los resultados de simulación en archivos CSV. Es un software de pago, pero presenta la posibilidad de descargar una versión de prueba gratuita. Las figuras 3.1 y 3.2 presentan una muestra de la interfaz gráfica del programa.

3.1. MODELOS DE ESTIMACIÓN SELECCIONADOS

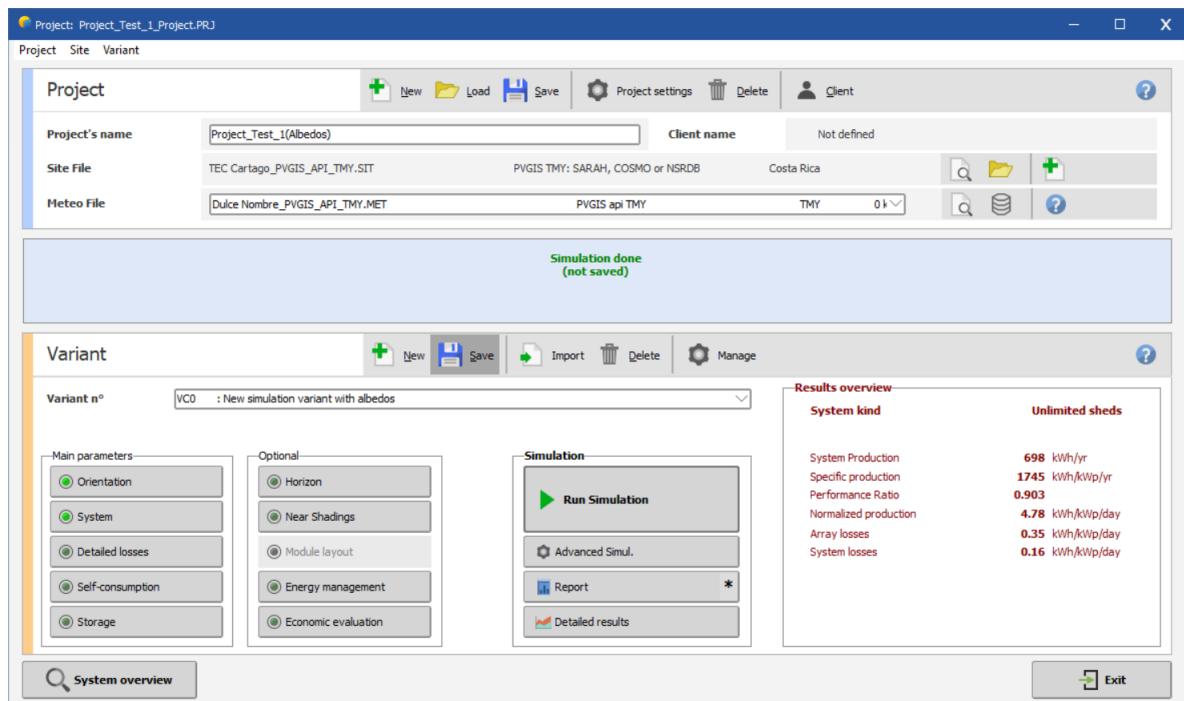


Figura 3.1: Interfaz gráfica de PVSyst, página principal de un proyecto

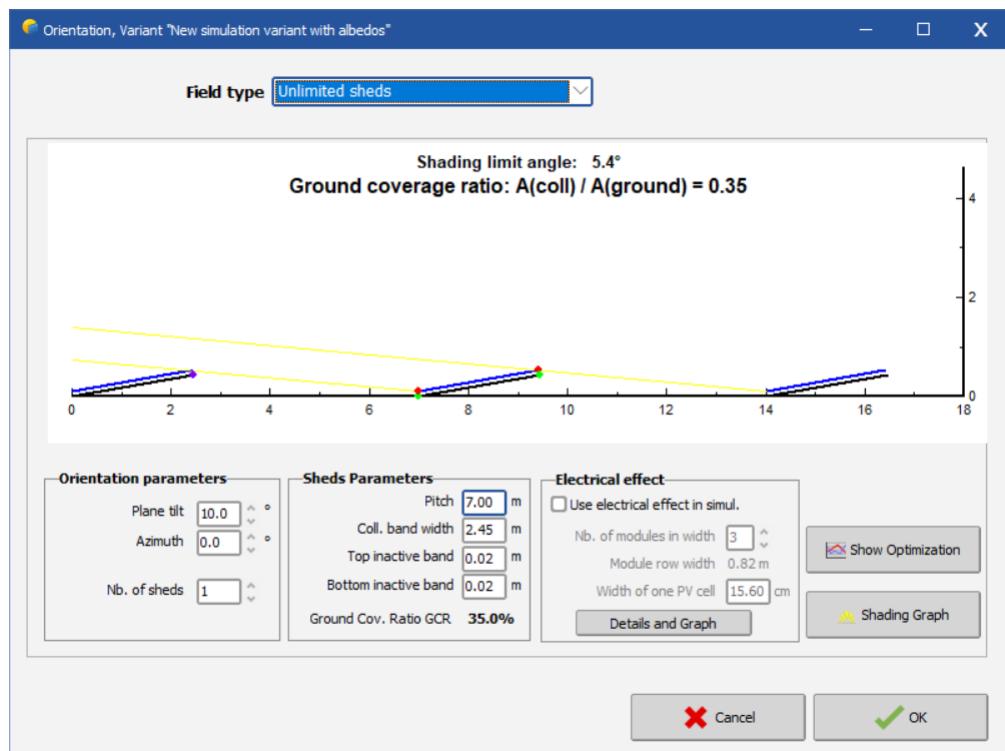


Figura 3.2: Interfaz gráfica de PVSyst, configuración de conjunto fotovoltaico

3.1.3. NREL Bifacial Radiance

El laboratorio de energías renovables de los Estados Unidos (NREL) desarrolló una integración de código abierto en *Python* para el software Radiance, que sirve para generar simulaciones de estimación energética en módulos bifaciales. Esta herramienta le permite al usuario configurar diversos parámetros de la simulación como ubicación geográfica (latitud y longitud), módulos de inclinación fija o con seguidores solares, simulaciones anuales, mensuales o diarias, la posibilidad de agregar módulos personalizados, ajustar el factor de bifacialidad, y variar parámetros del sitio como GCR, albedo y *clearance height*. La interfaz gráfica principal de esta herramienta se puede observar en la figura 3.3.

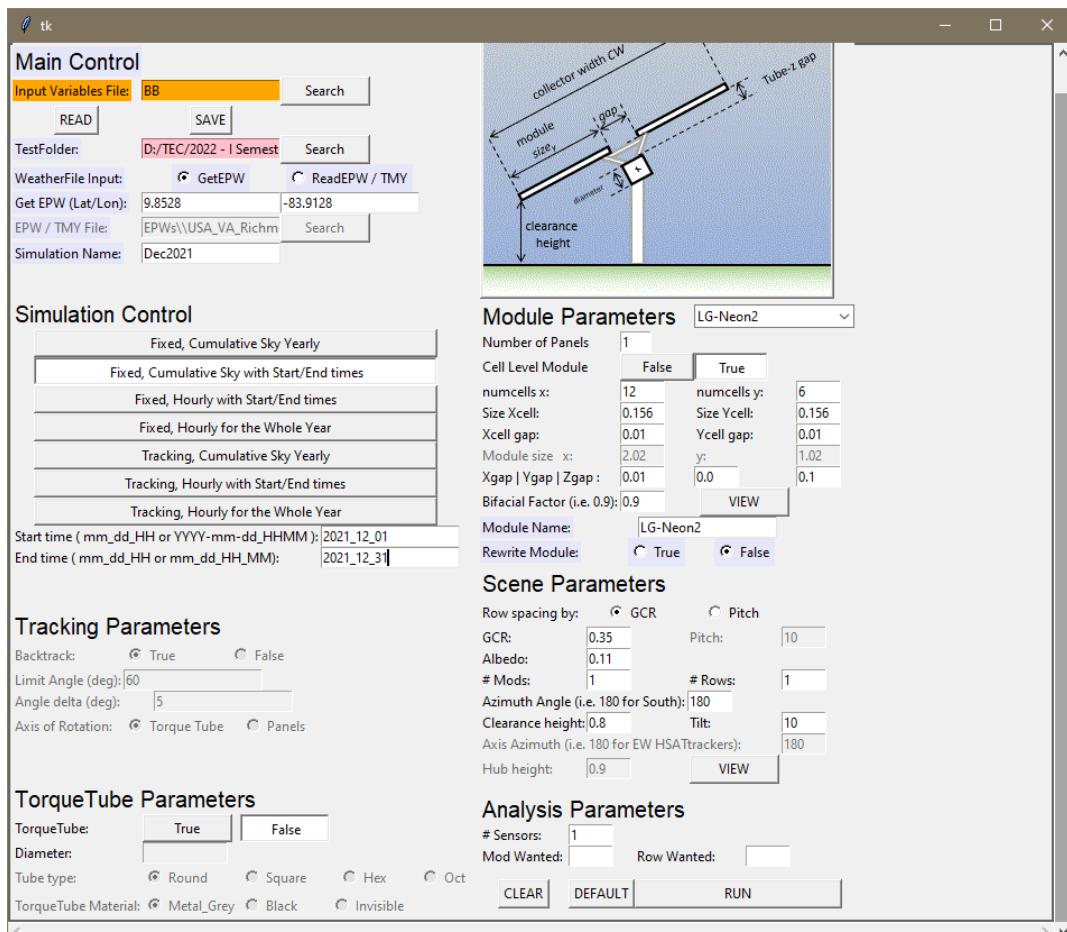


Figura 3.3: Interfaz gráfica de NREL Bifacial Radiance

3.2. Diseño de la herramienta

Para esta herramienta en particular se buscó trabajar con archivos externos en formato CSV y datos numéricos ingresados por el usuario, para lo que se utilizaron diferentes estructuras de datos. Adicionalmente, se implementó la funcionalidad de crear gráficas comparativas y tablas, todo esto alrededor de una interfaz gráfica de usuario (GUI) principal, en conjunto con GUIs secundarias.

Con el fin de cumplir con estas tareas se decidió trabajar con una metodología de programación modular [36], donde un programa principal se divide en partes más pequeñas (módulos o subprogramas), y cada una de ellas se encarga de llevar a cabo tareas concretas según su funcionalidad. El módulo principal inicialmente tiene el control del programa, y puede llamar al resto de los módulos auxiliares para ejecutar tareas y cederles el control temporalmente de acuerdo con sus necesidades. Esta decisión se tomó con la intención de crear un código fácil de manejar, leer, y revisar en caso de errores. Adicionalmente, un aspecto muy importante que se desea es facilitar a un usuario con conocimiento técnico la posibilidad de modificar los módulos o crear nuevos, para así agregar otros modelos de estimación energética, nuevos casos de graficación, o funcionalidades adicionales.

En la figura 3.4, se muestra un diagrama de bloques que explica de forma general el funcionamiento del programa. En resumen, inicialmente el usuario debe simular los casos de estudio que quiere analizar utilizando las herramientas de software de *PVSyst* y *NREL*, generando como salidas archivos en formato CSV con los datos de interés. Una vez hecho esto, se pueden tomar tres caminos, que pueden funcionar en paralelo: cargar dichos archivos CSV a la herramienta, ingresar datos reales medidos en campo, o ingresar las variables necesarias para realizar el cálculo del modelo empírico propuesto por *Solar World*. Estos datos se alimentan a distintos algoritmos que se encargan de revisar la extensión de los archivos CSV, extraer su información, organizarla según cada caso, realizar cálculos para generar las estimaciones energéticas, o estimar errores que indican la desviación entre los datos simulados y los datos medidos en campo. Toda esta información se puede graficar y tabular, según solicite el usuario, para su posterior análisis.

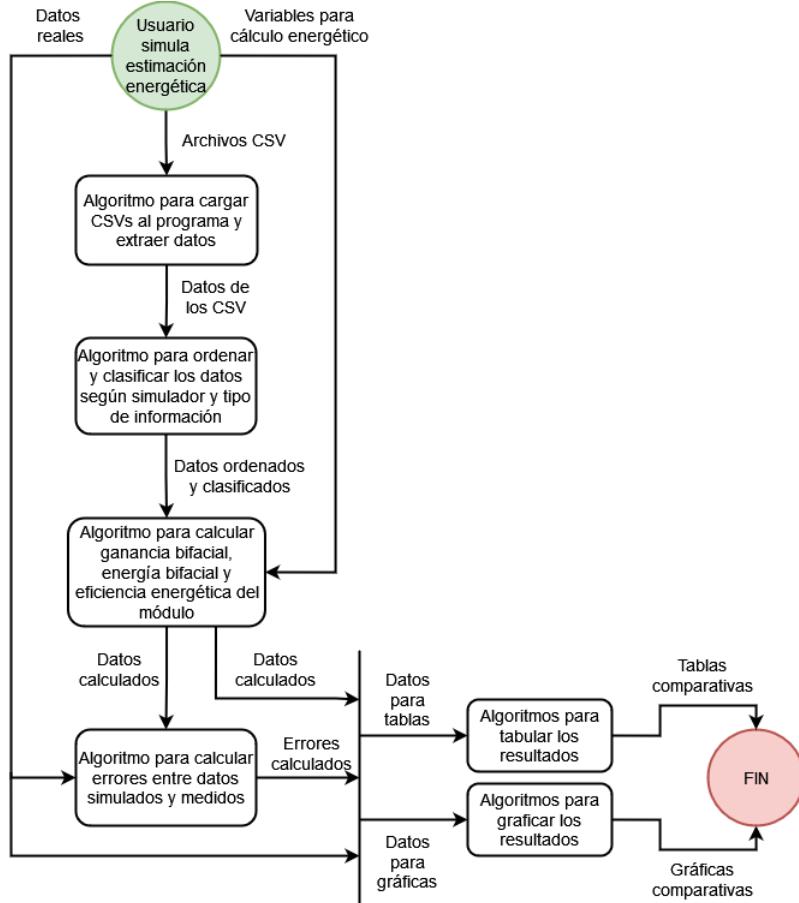


Figura 3.4: Diagrama de bloques de la herramienta diseñada

Los siguientes dos diagramas están construidos a partir del lenguaje UML (*Unified Modeling Language*). El UML está diseñado como un lenguaje general para modelar sistemas y explicar su funcionalidad sin importar en qué lugar del mundo se hayan desarrollado. Este lenguaje de modelado dispone de un conjunto amplio de elementos a graficar, como componentes, clases, casos de uso, objetos, flujos de trabajo, etc., así como las relaciones entre estos. Dentro de un marco general, su objetivo es describir el comportamiento, estructura, límites y objetos que forman parte del sistema [16].

En la figura 3.5 se puede observar un diagrama UML de los componentes de la herramienta diseñada. Este diagrama ilustra las dependencias entre los módulos del software y archivos o datos externos. Los módulos que componen el sistema son:

- ***gui.py***: Módulo principal del programa. Encargado de mostrar la GUI principal y llamar a los módulos auxiliares, así como mostrar sus GUIs secundarias correspondientes.
- ***globals.py***: Se encarga de recibir y almacenar datos de salida de todos los demás módulos del programa, y enviarlos como entradas a los módulos que lo soliciten.

- ***extraction.py***: Encargado de abrir la ventana de selección de archivos CSV para los casos de *PVSyst* y *NREL*, verificar la extensión de dichos archivos, abrirlos y extraer toda su información.
- ***sorting.py***: Su trabajo es recibir la información extraída de los archivos en *extraction.py*, aislar y recuperar la información relevante, y organizarla según cada tipo de variable y caso de simulación.
- ***empirical.py***: Módulo encargado de realizar el cálculo de la ganancia bifacial para el modelo empírico de *Solar World*, así como calcular algunas variables adicionales para los demás modelos, como la ganancia bifacial, la energía bifacial y la eficiencia energética del módulo.
- ***statistics.py***: Se encarga de hacer los cálculos para los errores de desviación entre los datos simulados y los medidos en campo.
- ***plotter.py***: Módulo para graficar los casos según solicitud del usuario.
- ***tables.py***: Encargado de crear tablas con la información calculada y extraída de los archivos, para complementar la visualización de las gráficas.

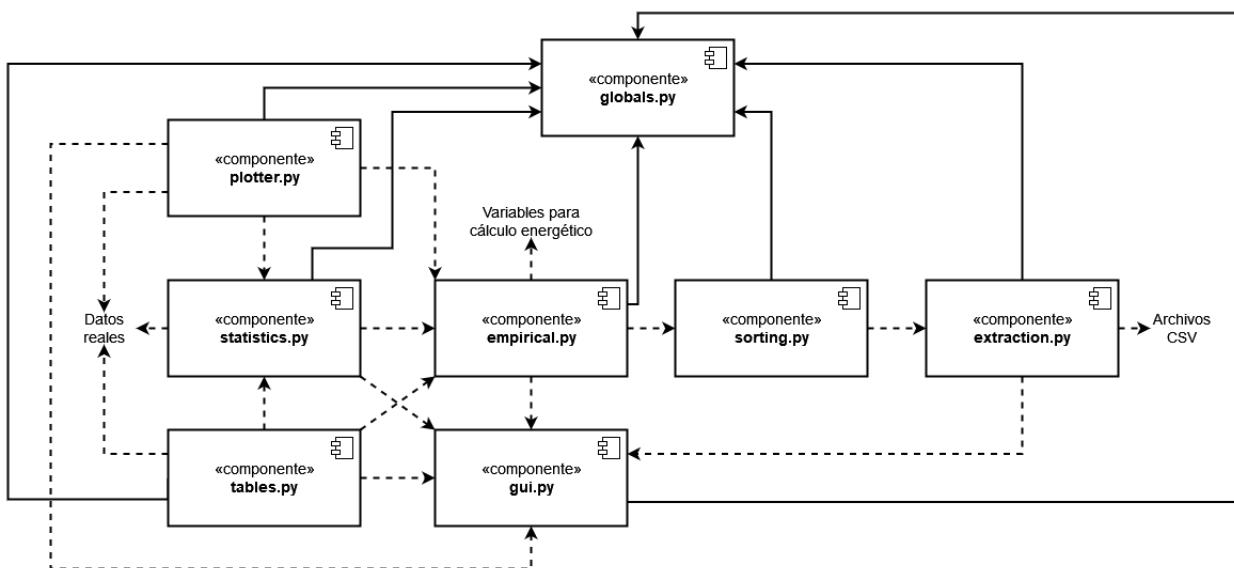


Figura 3.5: Diagrama de componentes de la herramienta diseñada

Por otra parte, en la figura 3.6 se puede observar un diagrama de casos de uso del programa. Este es un diagrama de alto nivel que presenta las interacciones entre los usuarios involucrados con el sistema y los principales procesos en los que puede participar. Estas relaciones son llamadas “casos de uso”, las cuales pueden ser de inclusión (*include*), donde un caso implica directamente la ejecución de otro proceso, o de extensión (*extend*), donde se deben cumplir ciertas condiciones para que ese caso suceda. Tanto el

include como el *extend* se representan dentro de comillas angulares (« »). Los procesos donde el usuario influye directamente están conectados con una línea negra (sin flecha), mientras que las interacciones entre procesos están representadas por flechas punteadas.

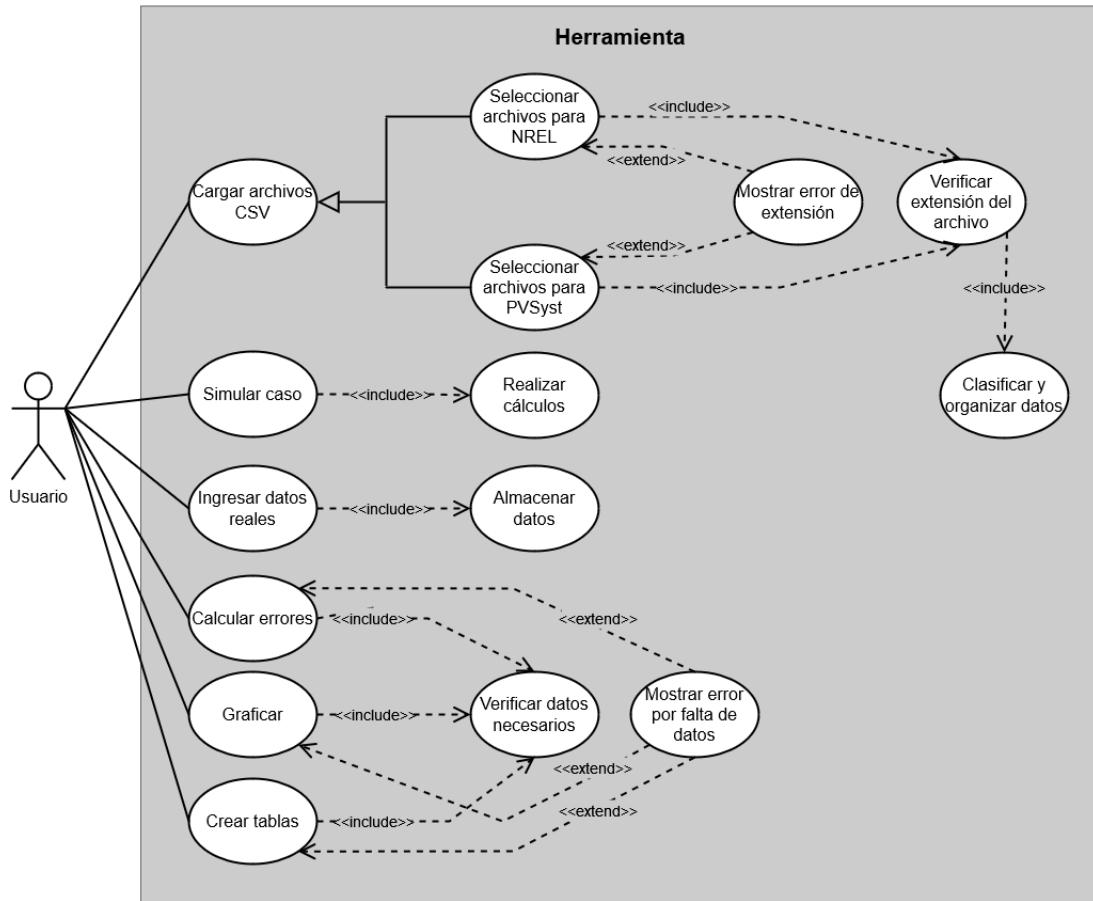


Figura 3.6: Diagrama de casos de uso de la herramienta diseñada

La figura 3.7 muestra la GUI principal del programa. Se puede observar un menú desplegable donde se tienen las opciones de herramientas de software para las que el usuario puede cargar la información simulada, o cargar los datos de mediciones de campo. También se encuentran botones para seleccionar qué herramientas y datos se desean trabajar, un botón para generar las gráficas de datos simulados o reales, y otro botón para calcular y graficar los errores estadísticos, así como un botón dedicado a la generación de tablas de datos y otro para tablas de errores. Se decidió configurar la herramienta en inglés para mejorar la accesibilidad a nivel internacional, y aumentar la posibilidad de cooperación científica. Por otra parte, la selección de las variables de ganancia bifacial (*bifacial gain*), energía monofacial (*bifacial energy*) y eficiencia energética del módulo (*module energy efficiency*) se hizo en base a que la ganancia bifacial es una salida en común en dos de los tres modelos que se están comparando (*NREL* y *Solar World*), la energía bifacial y eficiencia del módulo son salidas del modelo de *PVsyst*, y a partir de estos datos se pueden calcular de forma sencilla todas las variables de interés para los

tres modelos. Estos cálculos se explicarán en el siguiente capítulo.

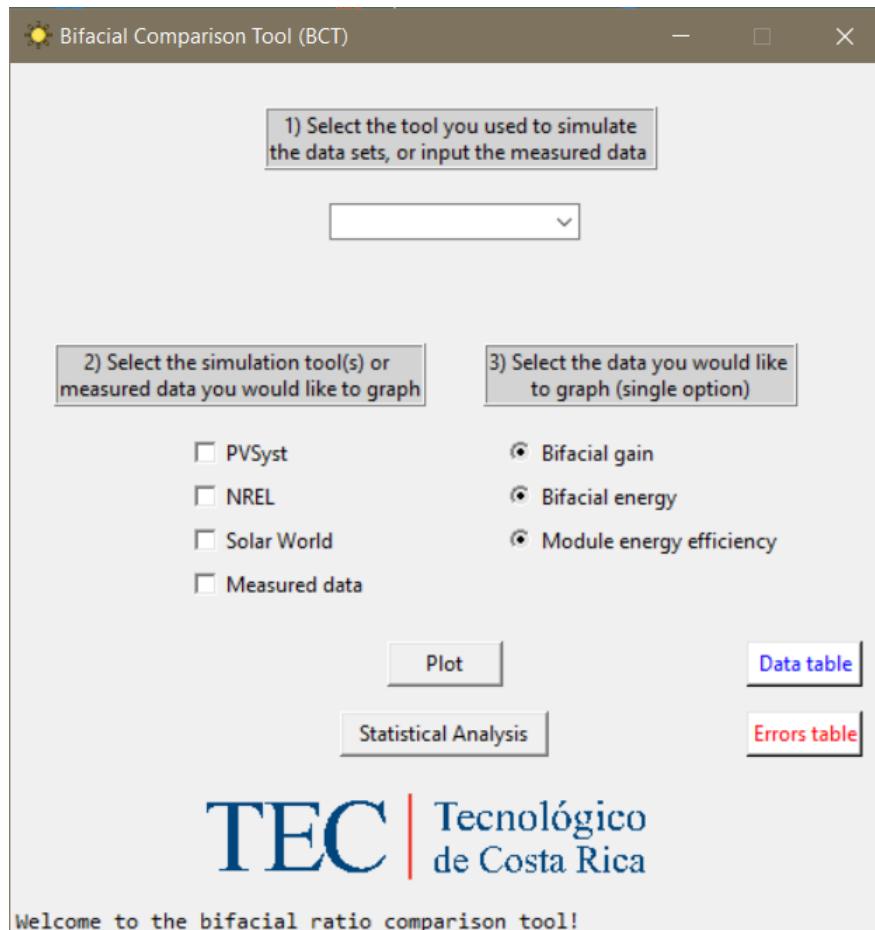


Figura 3.7: GUI principal del programa

Cada herramienta de simulación tiene su propia GUI independiente para cargar los datos necesarios, ya sea en formato CSV o ingresándolos de forma manual. Si el usuario selecciona la opción de *PVsyst* en el menú desplegable (ubicado en la parte superior de la ventana principal) podrá encontrar la GUI de la figura 3.8, donde puede cargar datos de energía monofacial, energía bifacial, eficiencia del módulo y la potencia nominal del módulo fotovoltaico utilizado en las simulaciones y/o mediciones en campo.

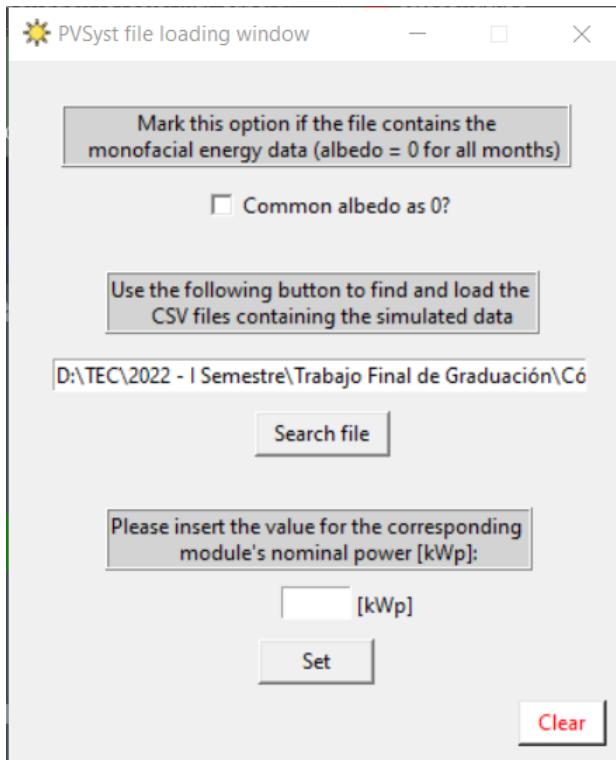


Figura 3.8: GUI para el caso de PVsyst

Al seleccionar la opción de *NREL Bifacial Radiance* en el menú desplegable, se puede utilizar la GUI de la figura 3.9, donde es posible cargar los archivos que contienen las simulaciones correspondientes a este modelo de estimación energética.

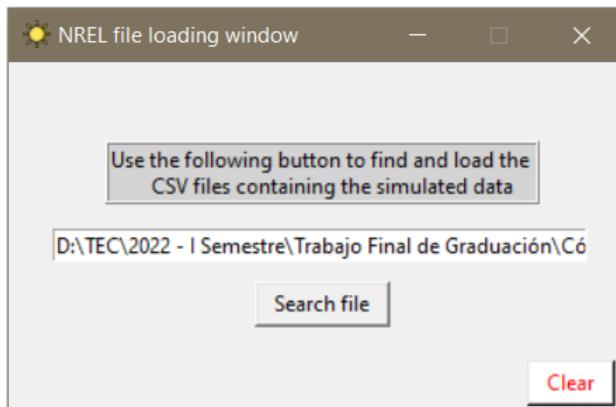


Figura 3.9: GUI para el caso de NREL

Si se selecciona la opción de *Solar World* en este mismo menú, se desplegará la interfaz de la figura 3.10. Aquí el usuario puede ingresar manualmente los parámetros necesarios para el cálculo de la ganancia bifacial para este modelo: valores de albedo (variables o constantes para cada mes), factor de bifacialidad, GCR y altura del módulo

respecto al suelo (o techo).

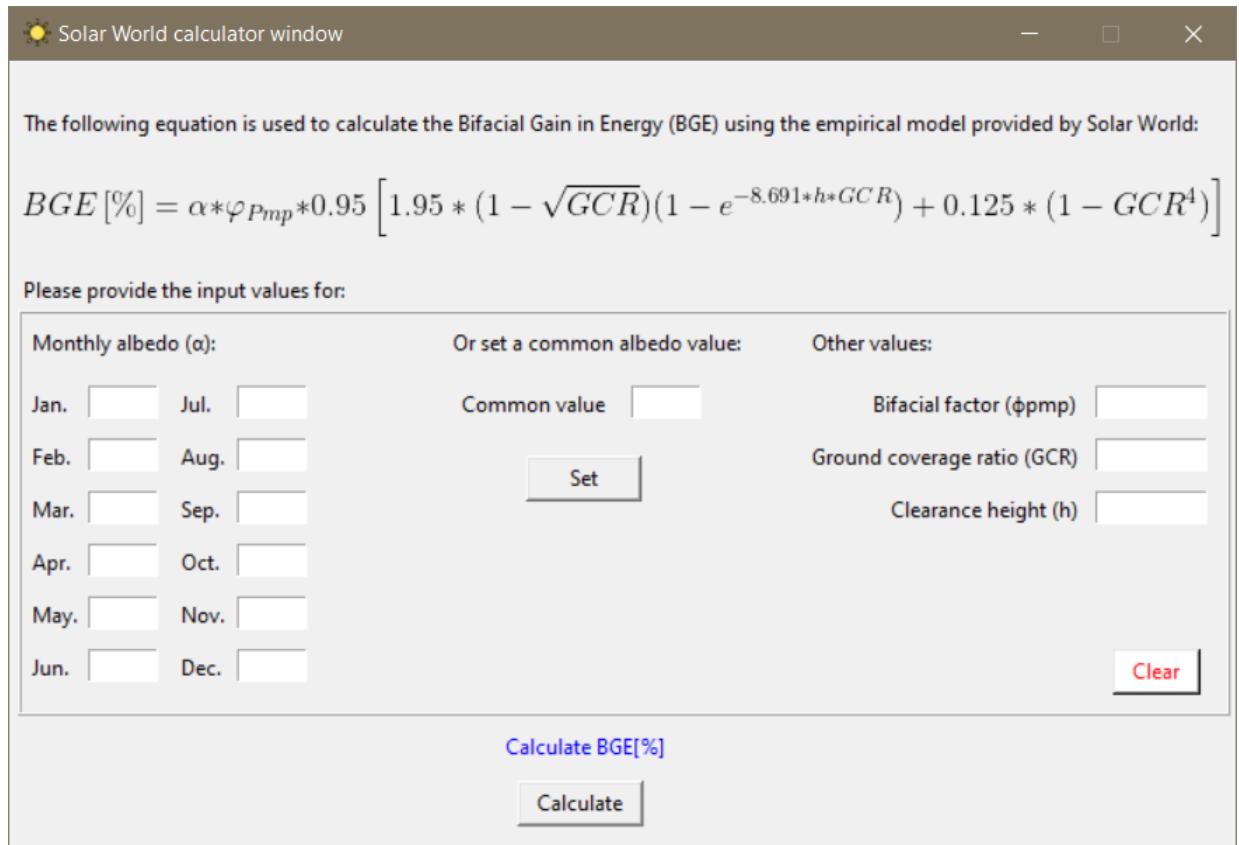


Figura 3.10: GUI para el caso de Solar World

La última opción en este menú desplegable corresponde a los datos medidos en campo (o datos reales). La interfaz correspondiente se encuentra en la figura 3.11, aquí el usuario puede ingresar datos de ganancia y energía bifacial, ambos mensuales. También se puede observar que todas las ventanas auxiliares tienen un botón con el texto *Clear*. Este sirve para reiniciar los datos que se han cargado para el modelo que se esté trabajando en ese momento, en caso de que el usuario se equivoque al cargarlos.

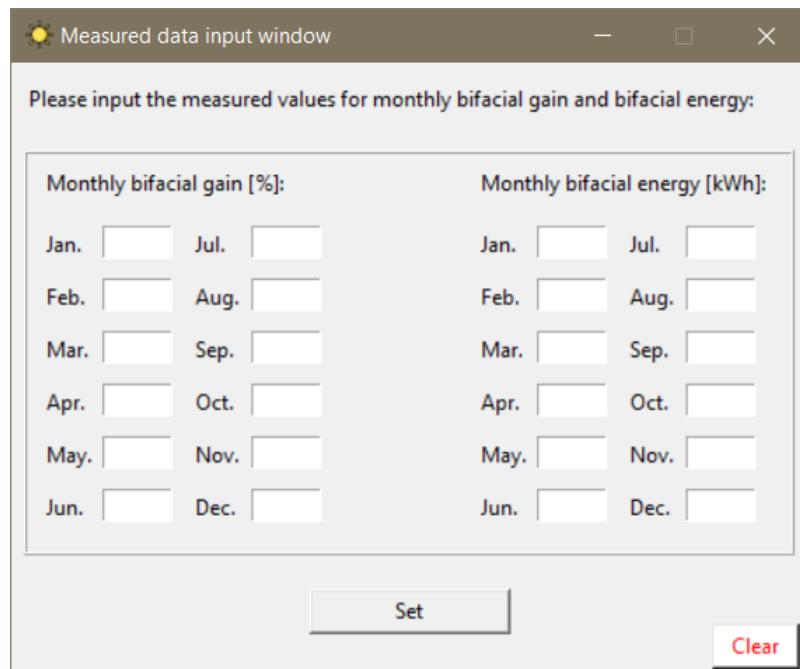


Figura 3.11: GUI para el caso de datos medidos

Estos procesos están organizados con la intención de habilitar que el usuario pueda cargar rápidamente los datos más relevantes para los cálculos energéticos correspondientes a cada modelo.

3.3. Entorno de desarrollo

Como se mencionó anteriormente, el lenguaje seleccionado para el desarrollo de la herramienta de evaluación es *Python*, debido a las diversas ventajas que implica su uso. Para la utilización de este lenguaje es necesario tomar en cuenta la gran variedad de bibliotecas y recursos disponibles, desarrollados tanto oficialmente como por la comunidad de usuarios.

3.3.1. Características del sistema

A continuación se mencionan las características de la máquina física utilizada para el desarrollo del proyecto, así como las versiones de herramientas de software, visibles en el cuadro 3.1.

Cuadro 3.1: Características del sistema

Características de máquina física	
Memoria RAM	8 GB
Procesador	Intel Core i7-4700
Sistema operativo	Windows 10
Herramientas de software	
<i>Python</i>	Versión 3.10.2
<i>PVSyst</i>	Versión 7.2
<i>NREL Bifacial Radiance</i>	Versión 5.2
<i>PyCharm Community Edition</i>	Versión 2021.3.2

3.3.2. Bibliotecas utilizadas

A continuación se mencionan las bibliotecas y módulos de *Python* que se utilizaron, junto con sus funcionalidades específicas:

- ***tkinter* (versión 8.6) [37]** : Este paquete forma parte de las bibliotecas estándar de *Python*, y no necesita una instalación independiente ya que viene incluido en los paquetes de instalación de *Python*. Es uno de los *toolkits* más comúnmente utilizados para la creación de GUIs, debido a su disponibilidad y compatibilidad para diversos sistemas operativos, su amplia documentación oficial, soporte técnico, y gran variedad de módulos y funciones. Los paquetes y *widgets* de esta biblioteca utilizados en la herramienta son:
 - ***Tk***: Es un subpaquete que habilita la implementación y manipulación de *widgets* altamente personalizables en la GUI. Su clase *Tk()* se puede usar para inicializar la construcción de una ventana, usualmente utilizada como principal, que sirve como base de la interfaz gráfica.
 - ***Label***: Consiste en una “etiqueta” donde se puede colocar texto o una imagen, y ubicarla en cualquier parte de una ventana *Tk*.

- **TopLevel**: Clase que permite crear una ventana secundaria con las mismas características que las creadas con la clase *Tk()*, ligada a la ventana principal y funcionando en paralelo con esta.
 - **Checkbutton**: Widget para crear botones cuadrados donde el usuario puede elegir, entre varias opciones, marcar una o más a la vez.
 - **Entry**: Widget que permite al usuario escribir texto directamente en una casilla rectangular e ingresarla al programa, con la opción de configurar previamente un texto para mostrar por defecto.
 - **Button**: Widget para crear un botón rectangular personalizable con texto, color, imagen o tamaño, y que se puede configurar para ejecutar una o más funciones cuando se presiona.
 - **MessageBox**: Widget que habilita la creación de un mensaje *pop-up* al ejecutar alguna acción en la interfaz, como presionar un botón. Este mensaje puede ser informativo o de alerta, según se configure.
 - **PhotoImage**: Widget que permite cargar imágenes en formatos *GIF*, *PGM*, *PPM*, y *PNG* sobre una *label* en la GUI.
 - **Combobox**: Clase perteneciente al módulo *ttk*, que permite crear un *widget* para una lista desplegable con múltiples opciones, llamada también *dropdown menu*.
 - **Radiobutton**: Este *widget* permite crear botones redondos, donde se puede dar al usuario la opción de elegir entre varias opciones, pero solamente seleccionar una de ellas a la vez.
 - **FileDialog**: Módulo que permite abrir una ventana emergente donde el usuario puede seleccionar y cargar archivos externos al programa, de extensión variable y configurable.
- **os [38]**: Este módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo, como abrir archivos, manipular y obtener rutas del sistema, entre otras opciones. Al igual que *tkinter*, este módulo forma parte de las bibliotecas estándar de *Python*, por lo que no requiere una instalación individual.
 - **pandas (versión 1.4.2) [39]**: Es una herramienta de código abierto para análisis y manipulación de datos, rápida, flexible y fácil de usar, creada en base a *Python*. Fue desarrollada por la empresa AQR Capital Management en 2008, y en 2009 pasó a ser de código abierto, por lo que hasta hoy en día es principalmente administrada por una comunidad internacional de usuarios y desarrolladores. Al ser una biblioteca externa es necesario hacer una instalación adicional en el sistema.

Entre sus funciones cuenta con objetos para manipulación de datos con indexación integrada, herramientas para leer y escribir datos entre diferentes estructuras y formatos como CSV, archivos de texto, Microsoft Excel, SQL y HDF5, así como

muchas otras funcionalidades y características que permiten un manejo ágil de grandes cantidades de datos.

- **NumPy** (versión 1.22) [40]: Es una biblioteca fundamental para computación científica en Python, que brinda la posibilidad de crear arreglos (*arrays*) n-dimensionales, objetos derivados como *masked arrays* y matrices, y rutinas para operar sobre estos objetos. *NumPy* es un proyecto de código abierto creado en 2005, y mantenido por una comunidad científica de usuarios en la plataforma GitHub.
- **matplotlib** (versión 3.5.1) [41]: Es una biblioteca amplia que permite crear visualizaciones gráficas estáticas, animadas e interactivas en *Python*, las cuales pueden ser implementadas en GUIs. Es ampliamente utilizada por su versatilidad, facilidad de uso e instalación simple.

Fue desarrollada por John Hunter y publicada en la revista *Computing in Science and Engineering* [34] de la IEEE en el año 2007, y actualmente es de código abierto y mantenida públicamente por desarrolladores de la comunidad de GitHub.

- **tkinterTable** [42]: Módulo creado por Damien Farrel y publicado en la plataforma GitHub. Contiene un conjunto de clases basadas en la biblioteca *tkinter* que permiten crear tablas dinámicas e interactivas en conjunto con una GUI. Estas tablas se pueden editar después de ser creadas directamente sobre la ventana que las contiene, y se pueden exportar como archivos externos en formato CSV.

Se decidió trabajar con estas bibliotecas debido a que cumplen con las necesidades de desarrollo para el proyecto, son livianas, de fácil instalación, además de que son de código abierto, están bien documentadas y mantenidas por la comunidad, lo que permite que el usuario pueda informarse sobre su utilización y agregar funcionalidades a la herramienta si así lo desea.

3.3.3. Instalación de bibliotecas y herramientas

De las bibliotecas utilizadas, dos de ellas (*tkinter* y *os*) vienen incluidas por defecto en la instalación de *Python*. Las demás bibliotecas necesitan instalación independiente, lo que se puede hacer desde la terminal (en este caso de Windows) mediante comandos sencillos. Según el caso, se debe insertar en la terminal:

- **pandas**: pip install pandas
- **NumPy**: pip install numpy
- **matplotlib**: python -m pip install -U matplotlib
- **tkinterTable**: pip install tkintertable

Al final de la instalación en cada proceso debería aparecer un aviso del resultado, exitoso o no, en la terminal. Más documentación y recursos para cada biblioteca se pueden encontrar en [39], [40], [41] y [42] respectivamente.

Para el caso de las herramientas de software utilizadas para simulaciones, *PVSyst* solamente puede ser instalado en sistema operativo Windows. Como se mencionó anteriormente, este programa es de pago en su versión completa, pero se puede descargar gratuitamente una versión de prueba por 30 días. Más información disponible sobre este recurso en [43].

La instalación de *NREL Bifacial Radiance* requiere más tiempo y cuidado, ya que depende del entorno de desarrollo desde donde se trabaje con *Python*. En este caso se trabajó con el IDE *Pycharm Community Edition*, lo que requirió configuraciones específicas. Una guía de instalación de la herramienta, grabada por parte del equipo que la desarrolló, se puede encontrar en [44], y las últimas actualizaciones e información sobre los lanzamientos de este software se pueden encontrar en su página de GitHub, disponible en [45].

3.4. Casos simulados

La herramienta de evaluación diseñada en este proyecto pretende tomar datos simulados previamente mediante herramientas de software externas, así como realizar cálculos internos, para comparar estas simulaciones entre sí y contra datos medidos físicamente. Para esto es necesario definir qué caso simular usando cada software, con el propósito de posteriormente validarla contra datos reales. Se decidió utilizar como ubicación geográfica de las simulaciones el campo de pruebas en exteriores del *Anhalt Photovoltaic Performance and Lifetime Laboratory* (APOLLO), de la Universidad de Ciencias Aplicadas de Anhalt, ubicado en Bernburg, Alemania. Esto debido a que se tienen datos tomados en esta ubicación durante un período de 1 año, mediante un experimento realizado en [9]. La instalación para este experimento se puede observar en la figura 3.12, donde se colocaron un módulo bifacial y un módulo monofacial sobre un soporte metálico.



Figura 3.12: Instalaciones experimentales, Bernburg, Alemania [9]

Las herramientas de software elegidas necesitan varios parámetros para ejecutar sus simulaciones (además de latitud y longitud), como el albedo, el ángulo de inclinación del panel, factor de bifacialidad, altura del panel respecto al suelo, y relación de cobertura del suelo (*ground coverage ratio* [GCR]). Con excepción del albedo, se decidió utilizar todos los parámetros sin variaciones (es decir, constantes), y con sus valores óptimos para la generación de energía en la ubicación de Bernburg, según los resultados obtenidos en [9]. Estos parámetros se pueden encontrar en el cuadro 3.2. Es importante recalcar que el *azimuth* de 180° corresponde a una orientación hacia el sur geográfico, esto en notación americana.

Cuadro 3.2: Parámetros constantes para las simulaciones

Parámetro	Valor	Unidad
Latitud	51.773	°
Longitud	11.763	°
Inclinación	35	°
Fact. Bifacialidad	90	%
Alt. respecto al suelo	1.5	m
GCR	0.5	-
Azimuth	180	°

Para aproximar las simulaciones lo más posible a los datos medidos en campo, se decidió simular usando valores de albedo variables para cada mes. Muchos sistemas de simulación energética disponibles actualmente utilizan albedo constante, lo que puede resultar en predicciones menos cercanas a la realidad. El efecto de variar el albedo en simulaciones de este tipo en lugar de usar albedo constante para todo el año se pueden analizar en [46]. En resumen, a lo largo del año las condiciones en el área de estudio varían, como el ángulo del sol, la irradiancia según la estación, cambios en la cobertura vegetativa de suelo (césped fresco o seco, nieve, etc.), lo que afecta el albedo. Los valores de albedo usados como referencia fueron medidos en el centro de APOLLO, en el experimento realizado en [46] y [9], tomando los valores promedio para cada mes. Estos se pueden observar en el cuadro 3.3.

Cuadro 3.3: Valores de albedo promedio correspondientes a cada mes (adaptación de tabla tomada de [9])

Mes	Albedo promedio [-]
Enero	0.113
Febrero	0.109
Marzo	0.149
Abril	0.146
Mayo	0.186
Junio	0.160
Julio	0.176
Agosto	0.173
Septiembre	0.141
Octubre	0.127
Noviembre	0.113
Diciembre	0.113

Los simuladores utilizados también usan como parámetro un módulo fotovoltaico especificado por el usuario. En este caso se decidió trabajar con uno de la línea LG NeON2

BiFacial, específicamente el modelo LG400N2T-J5, el cual es utilizado en el centro fotovoltaico del SESLab, en Cartago. Este panel bifacial cuenta con una potencia nominal de 400 Wp, 72 celdas bifaciales, y unas dimensiones de 2024 x 1024 mm. Su ficha técnica se puede encontrar en [47]. Se decidió trabajar con este panel por su referencia en la utilización de los sistemas del SESLab, la similitud con uno de los paneles disponibles en el simulador de PVsyst (el LG400N2T-A5 [48]), y la posibilidad de agregar sus características en los archivos del programa de NREL para poder ser simulado. En el experimento realizado en Bernburg se utilizó el panel modelo LG300N1T-G4 [49], sin embargo este no se encuentra entre los modelos comerciales disponibles para simular en PVsyst, por lo que se decidió no utilizarlo en este caso.

Es importante recalcar que todas las simulaciones fueron ejecutadas para un solo módulo bifacial. De igual forma, se trabajó con una configuración en modo *landscape* (horizontal), ya que esto produce un menor sombreado de la parte trasera del panel, lo que ayuda con la producción energética [9]. De igual forma, en las simulaciones se consideró el caso como ideal, sin tomar en cuenta autosombreado ni pérdidas.

Para la validación posterior de las simulaciones es necesario tener una referencia de valores reales medidos en el campo, los cuales serán ingresados en la herramienta diseñada. Para esto se utilizan las mediciones tomadas en [9] para energía bifacial y ganancia bifacial, visibles en la figura 3.13 y cuadro 3.4.

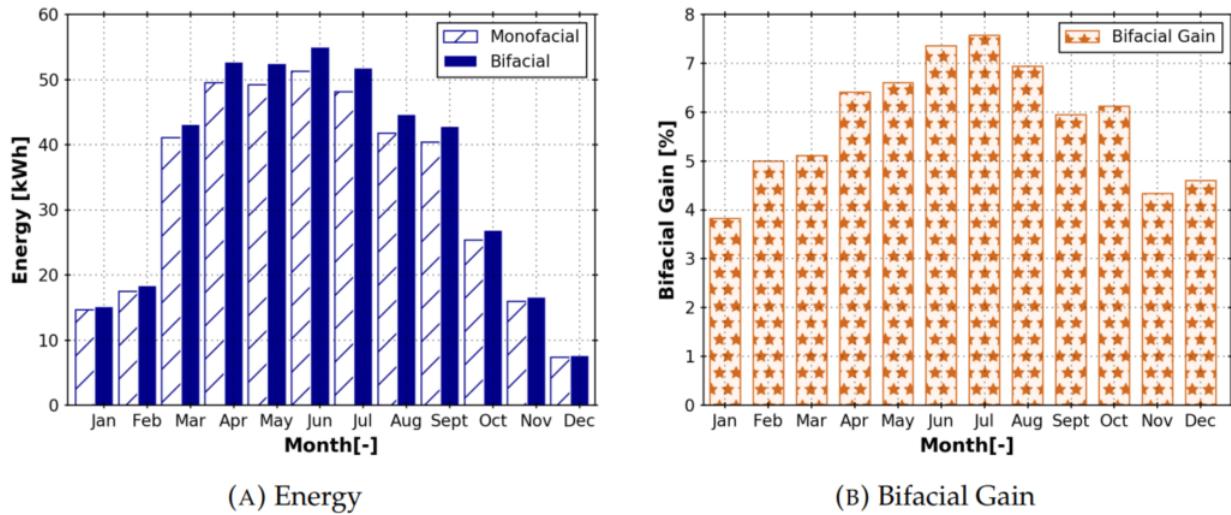


Figura 3.13: Mediciones tomadas en Bernburg, Alemania [9]

Cuadro 3.4: Valores de referencia para BGE y energía bifacial, tomados de [9]

Mes	BGE [%]	Energía bifacial [kWh]
Enero	3.8	15.0
Febrero	5.0	19.0
Marzo	5.1	42.0
Abril	6.4	52.0
Mayo	6.6	51.9
Junio	7.4	54.5
Julio	7.6	51.5
Agosto	6.9	45.0
Septiembre	5.9	44.0
Octubre	6.1	28.0
Noviembre	4.3	18.0
Diciembre	4.6	8.0

3.4.1. Simulación en PVsyst

Para el caso de PVsyst es necesario ejecutar dos casos de simulación: uno con albedo constante igual a 0 para todos los meses (emulando un sistema monofacial), para así obtener la energía monofacial de referencia, y otro con valores de albedo variables (donde se obtiene la energía bifacial y la eficiencia energética del panel), de acuerdo con el cuadro 3.3. Ambos casos comparten, además de una misma ubicación geográfica (usando la base de datos meteorológico PVGis), una configuración de inclinación del panel, azimuth, número de paneles y GCR (en base al cuadro 3.2), como se puede observar en la figura 3.14. Para esta herramienta el azimuth se establece en 0° para indicar el sur geográfico, ya que utiliza notación europea.

Ambos casos también comparten una misma configuración de módulo fotovoltaico e inversor, visible en la figura 3.15. La diferencia entre ellos, como se mencionó anteriormente, radica en los valores de albedo mensuales. La figura 3.16 muestra el caso de albedo cero, y la figura 3.17 el caso de albedo variable.

3.4. CASOS SIMULADOS

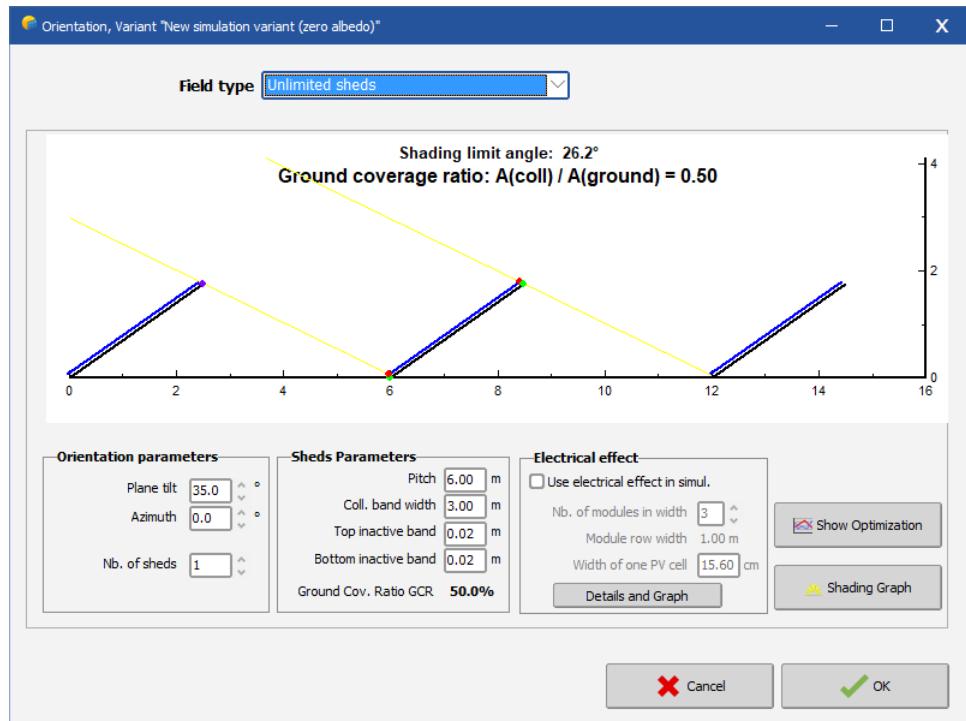


Figura 3.14: Orientación e instalación del panel fotovoltaico en PVsyst

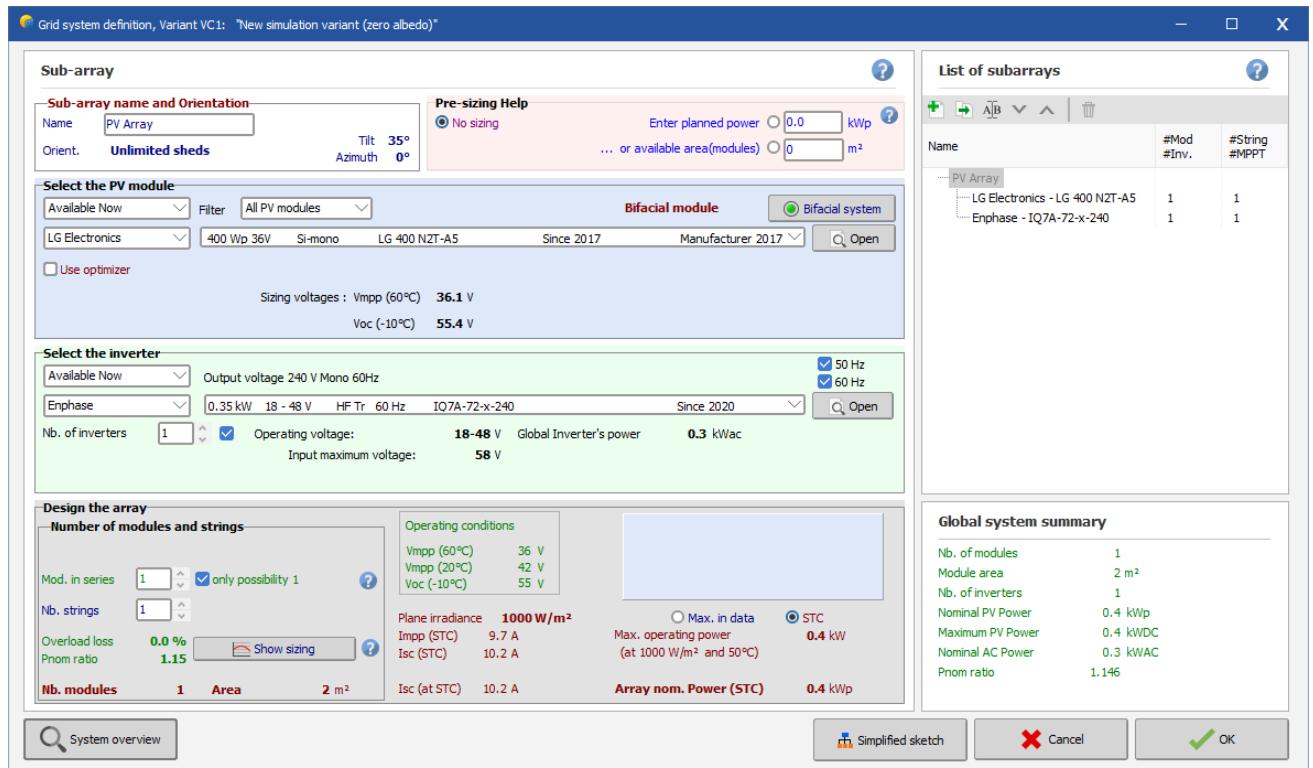


Figura 3.15: Configuración de modulo e inversor en PVsyst

3.4. CASOS SIMULADOS

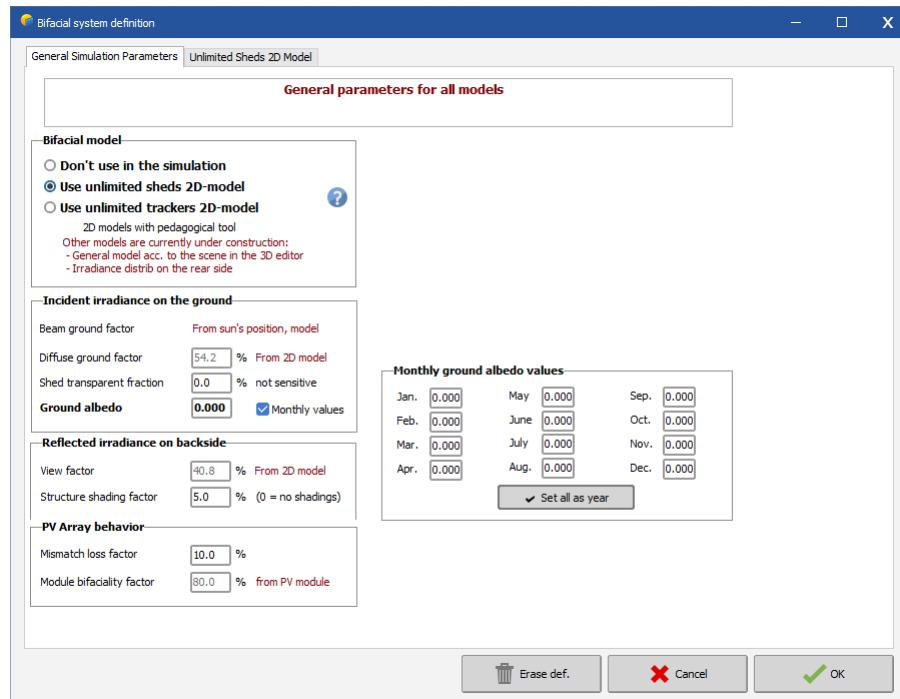


Figura 3.16: Configuración de albedo cero en PVSyst

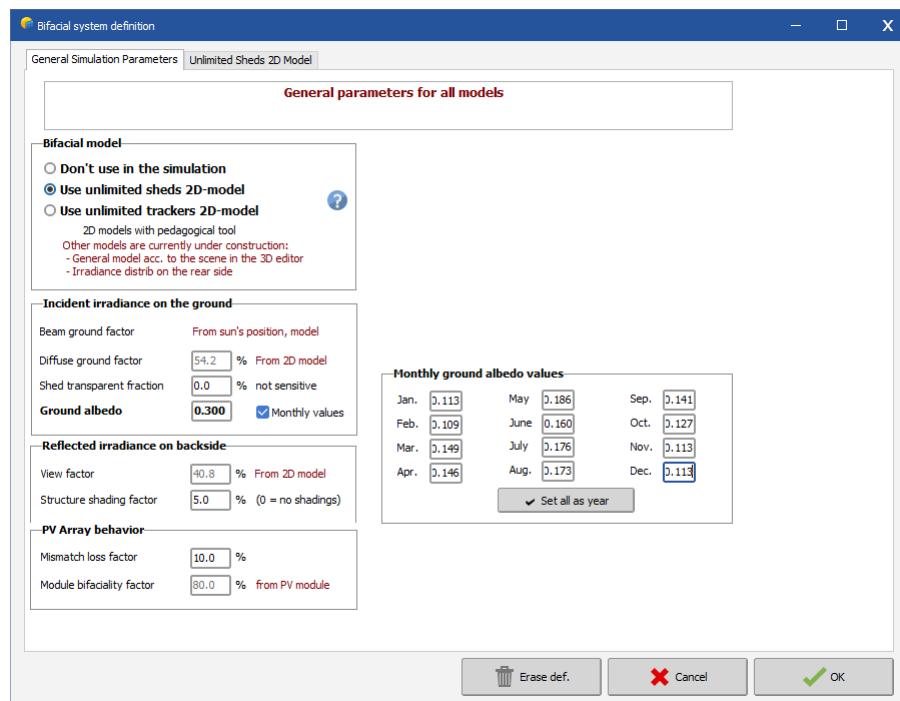


Figura 3.17: Configuración de albedo variable en PVSyst

Los resultados de estas simulaciones se exportan como archivos CSV, para ser utilizados.

zados posteriormente por la herramienta de evaluación diseñada.

3.4.2. Simulación en NREL Bifacial Radiance

Para esta herramienta es necesario ejecutar una simulación para cada mes del año que se desea evaluar, tomando en cuenta la cantidad de días por mes. En este caso se quiso evaluar un año completo, por lo que se ejecutaron 12 simulaciones diferentes, variando únicamente el valor de albedo. Se deben establecer las coordenadas geográficas deseadas (latitud y longitud), para que así el programa cargue el archivo meteorológico correspondiente. Es importante mencionar que estos archivos que contienen la información de recurso solar provienen de una base de datos propia de *NREL*, en archivos de formato EPW.

Una vez definido esto, es necesario configurar los parámetros del módulo. En este caso el módulo que se deseaba simular no estaba disponible por defecto en el programa, pero se cuenta con la posibilidad de agregar sus características a un archivo llamado *module.json* (que forma parte del paquete del programa), para utilizarlo en simulaciones. El código agregado a este archivo se puede observar en la figura 3.18. Entre los parámetros del módulo se puede elegir una orientación vertical (*portrait*) u horizontal (*landscape*), así como el factor de bifacialidad.

```
"LG-Neon2": {
    "bifil": 1,
    "cellModule": {
        "centerRB": null,
        "numcellsx": 6,
        "numcellsy": 12,
        "xcell": 0.150,
        "xcellgap": 0.01,
        "ycell": 0.150,
        "ycellgap": 0.01
    },
    "glass": false,
    "modulefile": null,
    "modulomaterial": "black",
    "numpanels": 1,
    "offsetfromaxis": 0,
    "scenex": 2.02,
    "sceney": 1.02,
    "scenez": 0.1,
    "text": "! genbox black cellPVmodule 0.150 0.150 0.02 | xform -t -0.4049999999999997 -0.805 0 -a 6 -t 0.162 0 0 -a 12 -t 0 0.162 0 -a 1 -t 0 1.02 0",
    "x": 2.02,
    "xgap": 0.01,
    "y": 1.02,
    "ygap": 0.0,
    "z": 0.02,
    "zgap": 0.1
}
```

Figura 3.18: Código para agregar un módulo al programa de *NREL*, mediante el archivo *module.json*

Posteriormente, se deben configurar los parámetros de la escena. Aquí se define el valor de GCR, albedo, número de módulos y filas, azimuth (que en esta herramienta se define en 180° para referirse al sur), *clearance height* e inclinación del módulo. Todos los parámetros, a excepción del albedo, se mantienen constantes para cada simulación

(configurados según el cuadro 3.2). El valor del albedo sí se cambia para cada mes, de acuerdo con el cuadro 3.3. La configuración en la interfaz del programa de todos estos parámetros mencionados, para el caso particular de enero, se puede observar en la figura 3.19. Cada una de estas simulaciones se guarda automáticamente en un archivo CSV, el cual contiene, entre otra información, la ganancia bifacial correspondiente al período de tiempo definido.

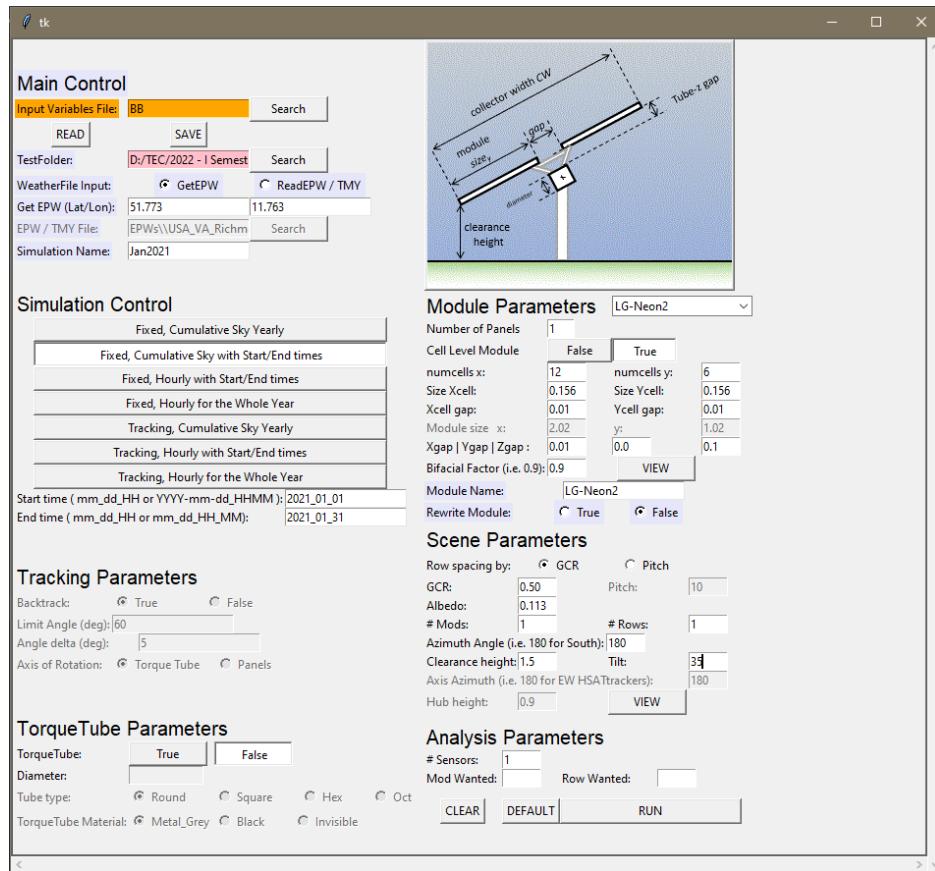


Figura 3.19: Configuración de parámetros en herramienta NREL Bifacial para el mes de enero del año 2021

3.4.3. Simulación en Solar World

Para ejecutar una simulación del modelo empírico creado por Solar World es necesario ejecutar la herramienta diseñada en este proyecto y establecer los valores solicitados para el cálculo de la ganancia bifacial, los cuales se encuentran en la interfaz visible en la figura 3.10. Una explicación más detallada de este caso de simulación se puede encontrar en el siguiente capítulo.

3.5. Métodos desarrollados

En esta sección se describen de forma detallada los métodos que conforman los módulos del programa, mencionados en la sección 3.2.

3.5.1. Interfaces gráficas en `gui.py`

Para este módulo se necesita importar las bibliotecas `tkinter` y `os`, así como invocar a los módulos `extraction`, `globals`, `empirical`, `statistics`, `plotter` y `tables`.

Este módulo, como se mencionó anteriormente, se encarga de todo el apartado gráfico de la herramienta, tanto para la ventana principal como para las auxiliares. Inicialmente, se crea la ventana principal mediante la biblioteca `tkinter`, y se configura su geometría, se crean *labels* para las instrucciones de uso y nombres de botones, además de crear el menú *dropdown* y los botones para cada función, utilizando los métodos en el apartado de `tkinter` mencionados en la sección 3.3.2. La figura 3.20 muestra el código para crear la estructura básica de la ventana principal.

```

17     """
18     Root configuration
19     """
20     root = Tk()
21     root.geometry("500x500")
22     root.title("Bifacial Comparison Tool (BCT)")
23     root.resizable(0,0)
24     root.iconbitmap("sun.ico")
25     root.propagate(0)

```

Figura 3.20: Creación de la estructura básica de la ventana principal

Para el diseño del menú *dropdown*, se crea la función *changevalues* para asignar las opciones de selección a este *widget*, e inmediatamente se crea el menú mediante el método `ttk.Combobox`, como se puede observar en la figura 3.21.

```

    """
    This function is used to set the values for the dropdown menu, which are initially set as empty to display no
    options before the user clicks the menu
    """

def changevalues():
    dropdown["values"]=[ "PV Syst", "NREL Bifacial Radiance", "Solar World", "Measured data"]

dropdown = ttk.Combobox(state="readonly", values=[], postcommand=changevalues)
dropdown.place(x=182, y=80)
dropdown.current(0)
dropdown.bind("<<ComboboxSelected>>", callbackFunc)

```

Figura 3.21: Creación del menú desplegable

Una vez creado este menú, se le asigna una tarea mediante la función *callbackFunc* (figura 3.22). Esta función se encarga de realizar una acción para cada opción seleccionada por el usuario, donde se llama a una función correspondiente para cada simulador, la cual crea su ventana individual con todos sus elementos. Estas funciones se explicarán con mayor detalle más adelante.

```
"""
Drop down menu with the names for the simulation tools

callbackFunc function sets the global variable toolname as the value obtained at the dropdown menu, prints the selected option in the console and returns that toolname, as well as calling the functions to create new windows.

"""

def callbackFunc(event):
    globals.toolname = dropdown.get()
    print("{} tool has been selected".format(globals.toolname))
    if globals.toolname == "PVsyst":
        pvsystWindow()

    elif globals.toolname == "NREL Bifacial Radiance":
        nrelWindow()

    elif globals.toolname == "Solar World":
        empInputWindow()

    elif globals.toolname == "Measured data":
        realData()

    return globals.toolname
```

Figura 3.22: Función asignada al menú

Seguidamente, se crean los *checkboxes* mediante el método *Checkbutton* de *tkinter* y se les asigna una variable de número entero entre 1 y 0, para representar un valor booleano (figura 3.23), y se crean funciones asignadas a cada botón para registrar la selección que hace el usuario, y enviarla al módulo *globals.py* (figura 3.24).

```
"""
Integer values for the buttons, that work as booleans """
pvsyst = IntVar()
nrel = IntVar()
solarworld = IntVar()
realdata = IntVar()

"""
Check buttons for the plotter """
check1 = Checkbutton(root, text="PVsyst", variable=pvsyst, onvalue=1, offvalue=0, command=setpvs)
check1.place(x=100, y=210)

check2 = Checkbutton(root, text="NREL", variable=nrel, onvalue=1, offvalue=0, command=setnrel)
check2.place(x=100, y=235)

check3 = Checkbutton(root, text="Solar World", variable=solarworld, onvalue=1, offvalue=0, command=setsw)
check3.place(x=100, y=260)

check4 = Checkbutton(root, text="Measured data", variable=realdata, onvalue=1, offvalue=0, command=lambda: [setreal(), changeState()])
check4.place(x=100, y=285)
```

Figura 3.23: Creación de botones de *check*

```
"""
Check buttons for the plotter
The following functions store the value of the checkbox variables to a corresponding global variable
"""

""" This functions get the value coming from the check buttons """
def setpvs():
    globals.pvscheck = pvsyst.get()
    print(globals.pvscheck)

def setnrel():
    globals.nrelcheck = nrel.get()
    print(globals.nrelcheck)

def setsw():
    globals.swcheck = solarworld.get()
    print(globals.swcheck)

def setreal():
    globals.realcheck = realdata.get()
    print(globals.realcheck)

def changeState():
    if globals.realcheck == 1:
        radio3.config(state=DISABLED)
    else:
        radio3.config(state=NORMAL)
```

Figura 3.24: Obtención de la selección de los botones de *checkbox*

Por último, para la configuración de la GUI principal, se crean los *radiobuttons*, se diseña una función (*select*) para obtener la selección del usuario y enviarla al módulo *globals.py*, y se crean los botones para graficar, ejecutar el análisis estadístico, crear tablas de simulaciones y de errores. Todos estos botones tienen asignadas funciones creadas en otros módulos, que se explicarán posteriormente. Todo este proceso se puede observar en la figura 3.25.

```
""" Radio buttons for the plotter, and function to set the global variable as the button selection """
def select():
    globals.radioselect = option.get()
    print(globals.radioselect)

option = StringVar()

radio1 = Radiobutton(root, text="Bifacial gain", variable=option, value="BifacialGain", command=select, state=NORMAL)
radio1.place(x=280, y=210)

radio2 = Radiobutton(root, text="Bifacial energy", variable=option, value="BifacialEnergy", command=select, state=NORMAL)
radio2.place(x=280, y=235)

radio3 = Radiobutton(root, text="Module energy efficiency", variable=option, value="ModuleEnergy", command=select, state=NORMAL)
radio3.place(x=280, y=260)

""" Button to plot """
plotButton = Button(root, text=" Plot ", command=plotter.plot)
plotButton.config(width=8)
plotButton.place(x=215, y=330)

""" Statistical analysis button """
statButton = Button(root, text=" Statistical Analysis ", command=lambda: [statistics.BGerrors(), statistics.BErrors(),
                                                               plotter.plotErrors()])
statButton.place(x=188, y=370)

""" Button to create simulations tables """
simTableButton = Button(root, text=" Data table ", command=tables.gen_table_sim)
simTableButton.config(width=8)
simTableButton.config(background="white", foreground="blue")
simTableButton.place(x=420, y=330)

""" Button to create errors tables """
errorTableButton = Button(root, text=" Errors table ", command=tables.gen_table_errors)
errorTableButton.config(width=8)
errorTableButton.config(background="white", foreground="red")
errorTableButton.place(x=420, y=370)
```

Figura 3.25: Creación y asignación de funciones para botones adicionales de la interfaz

En este mismo módulo, se crearon funciones individuales para el manejo de la información de cada uno de los modelos de simulación, así como para los datos reales.

Se creó la función *pvsystWindow* para crear la ventana correspondiente a este modelo (mediante el método Toplevel de *tkinter*, para generar ventanas adicionales dependientes de la principal), así como para generar etiquetas, botones y cajas de entrada de texto, de forma muy similar a la ventana principal. La configuración inicial de esta ventana, dentro de la función *pvsystWindow*, se puede observar en la figura 3.26.

```
def pvsystWindow():
    """ Create the main window """
    pvsystroot = Toplevel()
    pvsystroot.geometry("350x400")
    pvsystroot.title("PVsyst file loading window")
    pvsystroot.resizable(0, 0)
    pvsystroot.iconbitmap("sun.ico")
    pvsystroot.propagate(0)
```

Figura 3.26: Configuración básica de la ventana correspondiente a PVsyst

Dentro de esta ventana de PVsyst se utiliza el método `os.getcwd()` de la biblioteca `os` para obtener el directorio del archivo que está cargando el usuario y mostrarlo mediante un cuadro de texto. Así mismo, se crea la función `updatefilepathpv` para actualizar dicho directorio cada vez que se carga un archivo nuevo. La sección de código correspondiente a esto se encuentra en la figura 3.27.

```
""" Entry box to display the path of the path of the CSV file, or type said path """
currentpath1 = StringVar(pvsystroot, value=os.getcwd())
entrybox1 = Entry(pvsystroot, textvariable=currentpath1)
entrybox1.config(width=50)
entrybox1.place(x=25, y=170)

""" Update the filepath display """
def updatefilepathpv():
    entrybox1.delete(0, END)
    entrybox1.insert(END, globals.pvfilepath)
```

Figura 3.27: Configuración del directorio de los archivos para PVsyst

Entre los procesos importantes de esta ventana, se crea un botón para cargar los archivos y se le asigna la función `loadpv`, la cual llama una función específica del módulo `extraction` llamada `loadpvsyst` (explicada más adelante). De forma similar, se crea el entorno (caja de texto, etiquetas y botón) para asignar la potencia del panel utilizado, y se crea la función `setpower` para tomar el valor escrito por el usuario en la caja de texto y enviarlo al módulo `globals.py` para usarlo en cálculos posteriormente. Todo este proceso en el código del programa se puede ver en la figura 3.28.

```
""" Calls the external loadfile function """
def loadpv():
    extraction.loadpvsyst()

""" Button to load file """
loadButton1 = Button(pvsystroot, text=" Search file ", command=lambda: [loadpv(), updatefilepathpv()])
loadButton1.place(x=140, y=200)

""" Label for module power label """
powLoadLabel = Label(pvsystroot, text="""Please insert the value for the corresponding
module's nominal power [kWp]:""")
powLoadLabel.config(bg="light gray", justify="center", relief="ridge")
powLoadLabel.place(x=55, y=255)

""" Entry box for the module power """
entrybox2 = Entry(pvsystroot)
entrybox2.config(width=6, justify="center")
entrybox2.place(x=155, y=300)

""" Units label for module power """
unitsLabel = Label(pvsystroot, text="""[kWp]""")
unitsLabel.config(justify="center")
unitsLabel.place(x=195, y=300)

""" This function stores the module's power from the entry box in a global variable """
def setpower():
    globals.modpower = float(entrybox2.get())
    print(globals.modpower)
    MessageBox.showinfo("Success", "Saved module power: {} kW".format(globals.modpower))

    return globals.modpower

""" Button to set module power """
setButton1 = Button(pvsystroot, text=" Set ", command=setpower)
setButton1.config(width=8)
setButton1.place(x=142, y=330)
```

Figura 3.28: Código para llamar función de carga de archivos y asignación de potencia

Para crear la ventana específica del caso de *NREL* se diseñó la función *nrelWindow*. Esta tiene prácticamente las mismas funciones que la ventana de *PVSyst* (pero en una ventana de menor tamaño), incluyendo las funciones para cargar archivos (*load*) y mostrar el directorio de los archivos cargados (*updatefilepath*). La principal diferencia es que en esta ventana no es necesario cargar la potencia del panel, y aquí el botón asignado para cargar archivos está vinculado con una función específica para cargar datos de *NREL*, la cual se encuentra en el módulo *extraction.py*. El código para la configuración de esta ventana, y la función *load*, se encuentran en las figuras 3.29 y 3.30.

```
"""
This function creates an external window for the NREL case (ray tracing)
"""

def nrelWindow():
    """ Create the main window """
    thirdroot = Toplevel()
    thirdroot.geometry("350x200")
    thirdroot.title("NREL file loading window")
    thirdroot.resizable(0, 0)
    thirdroot.iconbitmap("sun.ico")
    thirdroot.propagate(0)
```

Figura 3.29: Código para configurar ventana auxiliar de NREL

```
""" Calls the external loadfile function """
def load():
    extraction.loadfile()
```

Figura 3.30: Código para llamar función de carga de archivos en NREL

La ventana para el caso de *Solar World* difiere respecto al diseño de las dos anteriores. Para crearla, se diseñó la función *empInputWindow*. Dentro de esta función se inicializa y configura la ventana de la misma forma que en los casos anteriores, pero con dimensiones distintas. Aquí se crean cajas de texto y etiquetas para las entradas de albedo, factor de bifacialidad, GCR y *clearance height*, las cuales son llenadas manualmente por el usuario. También existe la posibilidad de asignar un mismo valor de albedo a todos los meses de forma rápida, sin tener que pasar por las casillas de cada mes. Para esto se crea la función *setcommonalb*, donde se escribe el valor deseado en una casilla específica, y al presionar el botón *Set* se le asigna ese valor a todas las demás casillas de albedo. El método se puede observar en la figura 3.31.

```
""" Function to set common albedo to all entry boxes """
def setcommonalb():
    entries = [entryJan, entryFeb, entryMar, entryApr, entryMay, entryJun, entryJul, entryAug, entrySep, entryOct,
               entryNov, entryDec]

    for ent in entries:
        ent.delete(0, END)
        ent.insert(END, float(entrycommon.get()))
```

Figura 3.31: Función *setcommonalb* de la ventana de Solar World

Una vez que el usuario llena todas las casillas con la información necesaria, si presiona el botón *Caculate BGE[%]* se llamará a la función *entries*. Esta se creó para tomar los valores ingresados en todas las casillas y enviarlos a la función *empmodel* del módulo *empirical.py*, y así calcular la ganancia bifacial del modelo empírico. Este método se puede encontrar en la figura 3.32.

```
"""This function retrieves the values from the entry boxes and calls the function located in empirical.py module"""
def entries():
    albedos = [float(entryJan.get()), float(entryFeb.get()), float(entryMar.get()), float(entryApr.get()),
               float(entryMay.get()), float(entryJun.get()), float(entryJul.get()), float(entryAug.get()),
               float(entrySep.get()), float(entryOct.get()), float(entryNov.get()), float(entryDec.get())]

    print("Albedos:", albedos)

    bifgain = float(entryBif.get())
    GCR = float(entryGCR.get())
    clearheight = float(entryClear.get())

    for albvalue in albedos:
        empirical.empmodel(albvalue, bifgain, GCR, clearheight)

    MessageBox.showinfo("Success!", "Bifacial gain has been calculated successfully")
```

Figura 3.32: Función *entries* de la ventana de Solar World

Por último en el código para las interfaces gráficas tenemos lo correspondiente a los datos reales. Para manejar esto se creó la función *realData*, la cual se activa al seleccionar la opción de “*Measured data*” en el menú *dropdown* principal. Aquí también se inicializa y se configura la ventana de forma similar a las otras ventanas auxiliares, también se crean etiquetas y cajas de entrada de texto, en este caso para la ganancia bifacial y energía bifacial respecto a cada mes. Una vez que el usuario ha ingresado los datos y presiona el botón *Set*, se llama a las funciones creadas bajo los nombres *entriesBG* y *entriesBE*, las cuales obtienen los datos de ganancia y energía bifacial, respectivamente, de las casillas de texto, y las guarda en una lista dentro del módulo *globals.py* para usarlas en otros procedimientos. Los métodos usados por estas funciones se encuentra en el segmento de código de la figura 3.33.

```
"""This functions retrieve the values from the entry boxes and calls and stores them in globals.py module"""
def entriesBG():
    realBGs = [float(entryJan.get()), float(entryFeb.get()), float(entryMar.get()), float(entryApr.get()),
               float(entryMay.get()), float(entryJun.get()), float(entryJul.get()), float(entryAug.get()),
               float(entrySep.get()), float(entryOct.get()), float(entryNov.get()), float(entryDec.get())]

    for value in realBGs:
        globals.realBifacialGain.append(value)

    print("Real Bifacial Gains:", globals.realBifacialGain)
    MessageBox.showinfo("Success!", "Values have been loaded successfully")

def entriesBE():
    realBEs = [float(entryJan2.get()), float(entryFeb2.get()), float(entryMar2.get()), float(entryApr2.get()),
               float(entryMay2.get()), float(entryJun2.get()), float(entryJul2.get()), float(entryAug2.get()),
               float(entrySep2.get()), float(entryOct2.get()), float(entryNov2.get()), float(entryDec2.get())]

    for value in realBEs:
        globals.realBifacialEnergy.append(value)

    print("Real Bifacial Energies:", globals.realBifacialEnergy)
```

Figura 3.33: Funciones *entriesBG* y *entriesBE*, correspondientes a la ventana de datos reales

3.5.2. “Variables globales” en `globals.py`

El módulo `globals.py` inicialmente contiene variables vacías, sin valores asignados, en formato de *strings*, números enteros, de punto flotante y listas. Estas variables se llaman en los diferentes módulos del programa para ser sobreescritas con valores específicos conforme el usuario utiliza la herramienta, para que puedan ser utilizadas en diferentes partes del programa. Funciona como un módulo exclusivo para guardar una especie de variables globales. Algunas de estas variables se pueden observar en la sección de código de la figura 3.34. Este módulo es el único que no requiere la importación de bibliotecas ni otros módulos del programa.

```
""" This module contains all the global variables used across the different modules. """  
  
""" Name of the simulation tool used """  
toolname = ""  
  
""" File path for PVsyst csv """  
pvfilepath = ""  
  
""" PVsyst data lists """  
modpower = float  
pvsMonoEnergy = []  
pvsBifEnergy = []  
pvsEfficiency = []  
pvsBifGain = []  
  
""" Check for the usage of monofacial energy """  
monoCheck = int
```

Figura 3.34: Variables inicializadas como vacías en el módulo globals.py

3.5.3. Abrir archivos y extraer información en extraction.py

El módulo extraction.py utiliza las bibliotecas *tkinter* y *pandas*, e importa los módulos *globals* y *sorting*. Se diseñó para contener dos funciones principales que se encargan del manejo inicial de datos:

- ***loadpvsyst***: Esta función se diseñó para trabajar específicamente con el caso de *PVsyst* y el formato de sus archivos CSV. Se encarga de abrir una ventana mediante el método *FileDialog.askopenfilename* de la biblioteca *tkinter*, para que el usuario busque en el sistema los archivos con extensión “.csv” que deseé cargar. La función guarda en una variable de texto todo la ruta del archivo, incluyendo su nombre, y a partir de ahí revisa si la extensión concuerda con el formato de archivo deseado. Si no se cumple esta condición se muestra un mensaje de error, pero si se cumple se procede a leer y extraer toda la información del archivo para guardarla en un *dataframe* mediante *pandas*, específicamente con el método *read_csv*. En este caso es necesario omitir las dos primeras filas de información en los archivos, ya que estos corresponden a etiquetas de información que no son relevantes para los cálculos, y que generan un error en la lectura. Una vez hecho esto, se muestra un mensaje de carga exitosa de la información, y se manda el *dataframe* con la información a una función llamada *sort* que pertenece al módulo *sorting.py*. El código correspondiente a esta función se puede encontrar en la figura 3.35.

```
def loadpvsyst():
    pvsfilepath = FileDialog.askopenfilename(title="Open file", filetypes=((("Comma-separated values", "*.csv"),
                                                                           ("All files", "*.*")))
                                             )
    globals.pvfilepath = pvsfilepath

    if pvsfilepath == "":
        pass
    elif pvsfilepath != '' and pvsfilepath[-4:] != ".CSV":
        MessageBox.showerror("Error!", "Incorrect file extension")
    else:
        temporarylist = pvsfilepath.split("/")
        pvsfilename = temporarylist[-1]
        dataf = pd.read_csv(pvsfilepath, sep=",", skiprows=2, # , on_bad_lines='skip'
                            print(dataf)
        MessageBox.showinfo("Success!", """
        File: {}
        Simulation tool: {}

        Loaded successfully!
        """.format(pvsfilename, globals.toolname))

    return sorting.sort(dataf)
```

Figura 3.35: Código para leer archivos y extraer información de PVsyst en el módulo extraction.py

- **loadfile**: Esta función está diseñada para manejar específicamente la información del caso de *NREL Bifacial Radiance*. Trabaja de manera similar a *loadpvsyst*, pero en este caso no es necesario omitir líneas del archivo CSV, además de que solo se extrae la última columna de la información que contiene, ya que solo esta es de interés para el trabajo. Su código correspondiente se puede ver en la figura 3.36. Aquí también se envía el *dataframe* con la información extraída a la función *sort* del módulo *sorting.py*

```
"""
NREL case: This function opens a pop-up window to enable the user to look for the desired CSV files, and then separates
the useful information and stores it in a pandas dataframe, returning said dataframe to a different function
"""

def loadfile():
    path = FileDialog.askopenfilename(title="Open file", filetypes=((("Comma-separated values", "*.csv"),
                                                                    ("All files", "*.*"))))
    globals.filepath = path

    if path == "":
        pass
    elif path != '' and path[-4:] != ".csv":
        MessageBox.showerror("Error!", "Incorrect file extension")
    else:
        templist = path.split("/")
        filename = templist[-1]
        df = pd.read_csv(path)
        print(df)
        info = df.iloc[:, -1:]
        MessageBox.showinfo("Success!", """
File: {}
Simulation tool: {}

Loaded successfully!
""".format(filename, globals.toolname))

    return sorting.sort(info)
```

Figura 3.36: Código para leer archivos y extraer información de NREL en el módulo *extraction.py*

3.5.4. Organizar datos en *sorting.py*

El módulo *sorting.py* solo necesita la importación del módulo *globals.py*. Este contiene únicamente una función específica llamada *sort*, la cual recibe los *dataframes* provenientes de *extraction.py* y revisa en una variable llamada *toolname* del módulo *globals* si la herramienta que se está trabajando es la de *PVSyst* o la de *NREL*.

Si se está trabajando con la herramienta de *PVSyst*, se procede a revisar si el usuario especificó que el archivo que está cargando contiene la información de energía monofacial. Si es así, se extrae la información de la columna del archivo CSV bajo el nombre “EArray”, correspondiente a la energía monofacial.

Si el usuario no especifica que el archivo contiene datos de energía monofacial, se procede a revisar si el documento contiene de igual forma una columna llamada “EArray”, en este caso correspondiente a la energía bifacial, y si no, se busca que contenga una columna llamada “Ya”, la cual contiene la información de eficiencia energética del módulo. Los nombres de estas columnas se asignan por defecto en el simulador de *PVSyst*. La sección de código de la función *sort* correspondiente al caso de *PVSyst* se puede encontrar en la figura 3.37.

```
def sort(data):
    """ Extracts the information for PVsyst """
    if globals.toolname == "PVsyst":
        if globals.monoCheck == 1:
            try:
                monoenergy = data.loc[1:12, 'EArray'] # Change the column label if necessary
                monoenergyList = monoenergy.tolist()
                for element in monoenergyList:
                    globals.pvsMonoEnergy.append(float(element))
                print("Mono. energy list:", globals.pvsMonoEnergy)
            except:
                pass
        else:
            try:
                energy = data.loc[1:12, 'EArray'] # Change the column label if necessary
                energyList = energy.tolist()
                for element in energyList:
                    globals.pvsBifEnergy.append(float(element))
                print("Bif. energy list:", globals.pvsBifEnergy)
            except:
                efficiency = data.loc[1:12, 'Ya'] # Change the column label if necessary
                efficiencyList = efficiency.tolist()
                templist = []
                daysPerMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

                for i in efficiencyList:
                    templist.append(float(i))

                effPerMonth = [i * j for i, j in zip(templist, daysPerMonth)]

                for element in effPerMonth:
                    globals.pvsEfficiency.append(round(element, 4))
                print("Eff. list:", globals.pvsEfficiency)
```

Figura 3.37: Código de la función *sort* correspondiente al caso de PVsyst

Por otra parte, el caso de *NREL* es más simple. Si se cumple la condición de que la variable *globals.toolname* es igual a “NREL Bifacial Radiance”, se toma el dato de ganancia bifacial contenido en el *dataframe* que llegó como salida de *empirical.py* y se guarda en dos variables del módulo *globals*: primero como valor porcentual al multiplicarlo por 100, y segundo como el formato que tiene el dato extraído directamente del archivo. Todo este proceso se puede observar en el código de la figura 3.38.

```
""" Extracts the information for NREL Bifacial Radiance """
if globals.toolname == "NREL Bifacial Radiance":
    percent = data.iloc[0, 0] * 100
    globals.nrelBifRatio.append(round(data.iloc[0, 0], 4))
    globals.nrelPercentBG.append(round(percent, 4))

    print("NREL back/front ratio:\n")
    print(globals.nrelBifRatio)
    print("NREL percent BGE:\n")
    print(globals.nrelPercentBG)
```

Figura 3.38: Código de la función *sort* correspondiente al caso de NREL Bifacial Radiance

3.5.5. Cálculo de variables energéticas en *empirical.py*

El módulo *empirical.py* requiere la importación de la biblioteca *numpy* y del módulo *globals*. Se diseñó para realizar el cálculo de la ganancia bifacial, tanto para el modelo empírico de *Solar World* como para el modelo de *PVSyst*, además de la energía bifacial y eficiencia energética para los modelos de *NREL* y *Solar World*.

Para el cálculo de la ganancia bifacial de *Solar World* se crea la función *empmodel*, donde se toma como referencia la ecuación 3.2. Aquí se calcula el valor de BGE y se guarda tanto como valor porcentual como decimal en variables ubicadas en el módulo *globals.py*.

Para calcular la ganancia bifacial para el caso de *PVSyst* se diseña la función *bifGainCalc*, en base a la ecuación 4.1. Para calcular la energía bifacial se crea la función *bifEnergyCalc* (en base a la ecuación 4.2), y para el cálculo de la eficiencia energética del módulo se diseña la función *modEffCalc* (en base a la ecuación 4.3). El código correspondiente a todas las funciones del módulo *empirical.py* se puede encontrar en la figura 3.39.

```
"""
Function to calculate the empirical model's bifacial gain
"""

def empmodel(alb, bif, gcr, h):
    bge = alb * bif * 0.95 * (1.95 * (1 - np.sqrt(gcr)) * (1 - np.exp(-8.691 * h * gcr)) + 0.125 * (1 - gcr**4))
    print("\nBGE[%] = {}".format(bge))
    percent = bge * 100
    globals.empiricalList.append(round(percent, 4))
    globals.swPercentBG.append(round(percent, 4))
    print(globals.empiricalList)
    print(globals.swPercentBG)
    return percent


"""
Function to calculate the general bifacial gain
"""

def bifGainCalc(list1, list2):
    bgresult = [((i-j) / j)*100 for i, j in zip(list1, list2)]
    return bgresult


"""
Function to calculate general bifacial energy
"""

def bifEnergyCalc(monolist, bglist):
    beresult = [i + (i*j) for i, j in zip(monolist, bglist)]
    return beresult


"""
Function to calculate general module energy efficiency
"""

def modEffCalc(biflist, pow):
    templist = []

    for ele in biflist:
        templist.append(ele/pow)
    return templist
```

Figura 3.39: Código de la funciones contenidas en el módulo empirical.py

3.5.6. Estimación de errores en statistics.py

Para el módulo *statistics.py* es necesario importar las bibliotecas *numpy* y *tkinter*, así como el módulo *globals*. Aquí se crearon cuatro funciones relacionadas con el cálculo de los errores de RMSE y MBE.

La función llamada *rmse* se basa en la ecuación 4.4, y la función *mbe* se diseñó a partir de la ecuación 4.5. Ambas usan métodos de *Numpy*, específicamente para calcular la raíz cuadrada y la media de los números, y se usan como parámetros de las funciones los valores simulados (predicciones) y los valores reales (objetivos). El código para estas funciones se encuentra en la figura 3.40.

```
"""
Root mean square error (RMSE) - Predictions and targets need to be arrays
"""

def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())


"""
Mean bias error (MBE)
"""

def mbe(predictions, targets):
    return np.mean(predictions - targets)
```

Figura 3.40: Código para estimar RMSE y MBE en statistics.py

Estas funciones son invocadas a través de otras dos funciones específicas: *BGerrors* para calcular los errores para la ganancia bifacial, y *BEerrors* para los de la energía bifacial. En estas dos funciones se llama a las funciones *rmse* y *mbe* enviando como entradas la ganancia y energía de cada uno de los simuladores de modelos, para así calcular los errores y guardarlos en variables del módulo *globals.py*. El código correspondiente a estas funciones se puede observar en la figura 3.41.

```
"""
Function to calculate the bifacial gain errors
"""

def BGerrors():
    try:
        globals.rmsBGpvsys = [rmse(i,j) for i,j in zip(np.array(globals.pvsBifGain),np.array(globals.realBifacialGain))]
        globals.rmsBGnrel = [rmse(i,j) for i,j in zip(np.array(globals.nrelPercentBG),np.array(globals.realBifacialGain))]
        globals.rmsBGsw = [rmse(i,j) for i,j in zip(np.array(globals.swPercentBG),np.array(globals.realBifacialGain))]

        globals.mbeBGpvsys = [mbe(i,j) for i,j in zip(np.array(globals.pvsBifGain),np.array(globals.realBifacialGain))]
        globals.mbeBGnrel = [mbe(i,j) for i,j in zip(np.array(globals.nrelPercentBG),np.array(globals.realBifacialGain))]
        globals.mbeBGsw = [mbe(i,j) for i,j in zip(np.array(globals.swPercentBG),np.array(globals.realBifacialGain))]

    except:
        MessageBox.showerror("Missing data",
                            "Please load all the corresponding CSV files and calculations for all cases"
                            "of Bifacial Gain and Bifacial Energy")




"""
Function to calculate the bifacial energy errors
"""

def BEerrors():
    try:
        globals.rmsBEpvsys = [rmse(i, j) for i, j in zip(np.array(globals.pvsBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.rmsBEnrel = [rmse(i, j) for i, j in zip(np.array(globals.nrelBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.rmsBEsw = [rmse(i, j) for i, j in zip(np.array(globals.swBifEnergy), np.array(globals.realBifacialEnergy))]

        globals.mbeBEpvsys = [mbe(i, j) for i, j in zip(np.array(globals.pvsBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.mbeBEnrel = [mbe(i, j) for i, j in zip(np.array(globals.nrelBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.mbeBEsw = [mbe(i, j) for i, j in zip(np.array(globals.swBifEnergy), np.array(globals.realBifacialEnergy))]

    except:
        MessageBox.showerror("Missing data",
                            "Please load all the corresponding CSV files and calculations for all cases"
                            "of Bifacial Gain and Bifacial Energy")
```

Figura 3.41: Código para estimar errores de ganancia y energía bifacial en statistics.py

3.5.7. Creación de gráficas en plotter.py

El módulo *plotter.py* depende de las bibliotecas *tkinter* y *matplotlib*, y se relaciona con los módulos *globals* y *empirical*. En este módulo se crearon funciones generales para graficar desde una hasta cuatro variables en un mismo plano (funciones *plotOne* (figura 3.42), *plotTwo* (figura 3.43), *plotThree* (figura 3.44) y *plotFour* (figura 3.45)), usando como entradas de las funciones los valores para el eje Y, las etiquetas de ese eje, y el título de la gráfica. También se diseñó una función donde se crean todas las variaciones de gráficas para simulaciones y datos reales (función *plot*), así como otra para todos los casos de errores (función *plotErrors*).

```
def plotOne(var1, label1, labelY, varTitle):
    # Graph
    plt.close()
    plt.figure("{}".format(varTitle))
    plt.plot(months, var1, marker="o", color="#c0392b", label=label1)
    plt.xlabel("Month [-]")
    plt.ylabel(labelY)
    plt.title(varTitle)

    plt.legend()
    plt.tight_layout()
    plt.grid()

    plt.show()
```

Figura 3.42: Código para graficar 1 variable en plotter.py

```
def plotTwo(var1, var2, label1, label2, labelY, varTitle):
    # Graph
    plt.close()
    plt.figure("{}".format(varTitle))

    plt.plot(months, var1, marker="o", color="#c0392b", label=label1)
    plt.plot(months, var2, marker="o", color="#3498db", label=label2)

    plt.xlabel("Month [-]")
    plt.ylabel(labelY)
    plt.title(varTitle)

    plt.legend()
    plt.tight_layout()
    plt.grid()

    plt.show()
```

Figura 3.43: Código para graficar 2 variables en plotter.py

```
def plotThree(var1, var2, var3, label1, label2, label3, labelY, vartitle):
    # Graph
    plt.close()
    plt.figure("{}".format(vartitle))

    plt.plot(months, var1, marker="o", color="#c0392b", label=label1)
    plt.plot(months, var2, marker="o", color="#3498db", label=label2)
    plt.plot(months, var3, marker="o", color="#27ae60", label=label3)

    plt.xlabel("Month [-]")
    plt.ylabel(labelY)
    plt.title(vartitle)

    plt.legend()
    plt.tight_layout()
    plt.grid()
    plt.show()
```

Figura 3.44: Código para graficar 3 variables en plotter.py

```
def plotFour(var1, var2, var3, var4, label1, label2, label3, label4, labelY, vartitle):
    # Graph
    plt.close()
    plt.figure("{}".format(vartitle))

    plt.plot(months, var1, marker="o", color="#c0392b", label=label1)
    plt.plot(months, var2, marker="o", color="#3498db", label=label2)
    plt.plot(months, var3, marker="o", color="#27ae60", label=label3)
    plt.plot(months, var4, marker="o", color="#7f8c8d", label=label4, linestyle="--")

    plt.xlabel("Month [-]")
    plt.ylabel(labelY)
    plt.title(vartitle)

    plt.legend()
    plt.tight_layout()
    plt.grid()
    plt.show()
```

Figura 3.45: Código para graficar 4 variables en plotter.py

La función *plot* está diseñada en base a los diferentes casos de graficación, llamando a las funciones anteriores para graficar según el número de modelos y las variables energéticas que se quieran visualizar. La figura 3.46 muestra el código para el caso donde el usuario selecciona (en la GUI principal) que quiere graficar los tres modelos simulados junto con los datos reales, ya sea para ganancia o energía bifacial. Lo mismo se tiene dentro de la misma función para las diferentes combinaciones entre simuladores, con o sin los datos reales, mediante condiciones *if* y *elif* anidadas.

```
def plot():
    # Four variables scenario
    # PVsyst, NREL, Solar World and real data
    if globals.pvscheck == 1 and globals.nrelcheck == 1 and globals.swcheck == 1 and globals.realcheck == 1:
        if globals.radioselect == "BifacialGain":
            try:
                globals.pvsBifGain = empirical.bifGainCalc(globals.pvsBifEnergy, globals.pvsMonoEnergy)
                plotFour(globals.pvsBifGain, globals.nrelPercentBG, globals.swPercentBG, globals.realBifacialGain, "PVsyst",
                         "NREL", "Solar World", "Measured data", "Bifacial gain [%]", "Bifacial gain values per month")
            except:
                MessageBox.showerror("Missing data", "Please load all the corresponding CSV files")

        elif globals.radioselect == "BifacialEnergy":
            try:
                globals.nrelBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.nrelBifRatio)
                globals.swBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.empiricalList)
                plotFour(globals.pvsBifEnergy, globals.nrelBifEnergy, globals.swBifEnergy, globals.realBifacialEnergy,
                         "PVsyst", "NREL", "Solar World", "Measured data", "Bifacial energy [kWh]", "Bifacial energy values per month")
            except:
                MessageBox.showerror("Missing data", "The monofacial energy from PVsyst CSVs and the bifacial gains for"
                                     " each simulation tool are needed to calculate the bifacial energy")
```

Figura 3.46: Código para graficar los tres modelos junto con los datos reales en plotter.py

La función para graficar los errores, llamada *plotErrors*, funciona de forma similar a *plot*. Aquí también se revisa la selección del usuario en la interfaz principal, pero no se considera el caso de la selección de los valores reales (porque ya se está tomando en cuenta implícitamente), ni la selección de la eficiencia energética del panel, ya que no se tienen mediciones de referencia para esta variable. Otra diferencia es que esta función configura y grafica directamente según el caso, sin llamar a otras funciones auxiliares. Una sección del código de *plotErrors* para el caso de selección de los modelos de *PVsyst*, *NREL* y *Solar World* para la variable de ganancia bifacial se muestra en la figura 3.47. La función considera también las demás combinaciones entre modelos de simulación y variables de energía y ganancia bifacial utilizando condiciones *if* y *elif* anidadas.

```
def plotErrors():
    try:
        plt.close("all")
        # PVsyst, NREL and Solar World
        if globals.pvscheck == 1 and globals.nrelcheck == 1 and globals.swcheck == 1:
            x = months
            if globals.radioselect == "BifacialGain":

                y1 = globals.rmsBGpvsys
                y2 = globals.rmsBGNrel
                y3 = globals.rmsBGSW

                y4 = globals.mbeBGpvsys
                y5 = globals.mbeBGNrel
                y6 = globals.mbeBGSW

                plt.figure("Errors for Bifacial Gain in Energy (BGE)")
                plt.subplot(2, 1, 1)
                plt.plot(x, y1, marker="o", label="PVsyst")
                plt.plot(x, y2, marker="o", label="NREL")
                plt.plot(x, y3, marker="o", label="Solar World")
                plt.title("Bifacial Gain in Energy (BGE)")
                plt.ylabel("RMSE [%]")
                plt.legend()
                plt.tight_layout()
                plt.grid()

                plt.subplot(2, 1, 2)
                plt.plot(x, y4, marker="o", label="PVsyst")
                plt.plot(x, y5, marker="o", label="NREL")
                plt.plot(x, y6, marker="o", label="Solar World")
                plt.xlabel("Month [-]")
                plt.ylabel("MBE [%]")
                plt.legend()
                plt.tight_layout()
                plt.grid()

            plt.show()
```

Figura 3.47: Código para graficar los errores de ganancia bifacial para los tres modelos de simulación en *plotter.py*

3.5.8. Generación de tablas en *tables.py*

Por último, se tiene el módulo *tables.py*. Este utiliza las bibliotecas *tkinter*, *tkinterTable* y *numpy*, y se relaciona con los módulos *globals.py*, *empirical.py* y *statistics.py*.

El diseño de este módulo consiste en funciones para configurar y crear las tablas según la cantidad de columnas que se necesite (entre 1 y 6, llamadas *onecolumns*, *twocolumns*, *threecolumns*, *fourcolumns* y *sixcolumns*, omitiendo el 5 ya que no se usa este caso), así como una función para generar las tablas para simulaciones (función *gen_table_sim*) y otra para generar tablas de errores (función *gen_table_errors*).

La figura 3.48 muestra el código correspondiente a la función *threecolumns*, donde se tiene como entradas las listas con datos energéticos y los nombres para cada una de las columnas. Estas listas se convierten en un *array* usando un método de *numpy* (para poder ser operadas por la bibliotecas *tkintertable*), se le aplica la transpuesta para

organizar los datos para una mejor visualización, y posteriormente se configuran los datos que conforman cada una de las filas y columnas de la tabla. Las demás funciones para configurar las tablas funcionan de manera similar, simplemente variando la cantidad de columnas.

```
def threecolumns(col1, col2, col3, colname1, colname2, colname3):
    array = np.array([col1, col2, col3])
    tarray = np.transpose(array)

    master = Toplevel()
    tframe = Frame(master)
    tframe.pack()

    data = {'row1': {colname1: float(tarray[0, 0]), colname2: float(tarray[0, 1]), colname3: float(tarray[0, 2])},
            'row2': {colname1: float(tarray[1, 0]), colname2: float(tarray[1, 1]), colname3: float(tarray[1, 2])},
            'row3': {colname1: float(tarray[2, 0]), colname2: float(tarray[2, 1]), colname3: float(tarray[2, 2])},
            'row4': {colname1: float(tarray[3, 0]), colname2: float(tarray[3, 1]), colname3: float(tarray[3, 2])},
            'row5': {colname1: float(tarray[4, 0]), colname2: float(tarray[4, 1]), colname3: float(tarray[4, 2])},
            'row6': {colname1: float(tarray[5, 0]), colname2: float(tarray[5, 1]), colname3: float(tarray[5, 2])},
            'row7': {colname1: float(tarray[6, 0]), colname2: float(tarray[6, 1]), colname3: float(tarray[6, 2])},
            'row8': {colname1: float(tarray[7, 0]), colname2: float(tarray[7, 1]), colname3: float(tarray[7, 2])},
            'row9': {colname1: float(tarray[8, 0]), colname2: float(tarray[8, 1]), colname3: float(tarray[8, 2])},
            'row10': {colname1: float(tarray[9, 0]), colname2: float(tarray[9, 1]), colname3: float(tarray[9, 2])},
            'row11': {colname1: float(tarray[10, 0]), colname2: float(tarray[10, 1]), colname3: float(tarray[10, 2])},
            'row12': {colname1: float(tarray[11, 0]), colname2: float(tarray[11, 1]), colname3: float(tarray[11, 2])}
        }

    table = TableCanvas(tframe, data=data)
    # table.redraw()
    table.show()

    master.mainloop()
```

Figura 3.48: Código para generar una tabla de tres columnas mediante la función *threecolumns* en tables.py

La función *gen_table_sim* está diseñada para revisar la selección hecha por el usuario de modelos y variables energéticas, de manera similar a la forma en que se generan las gráficas. La diferencia es que en este caso se invoca a las funciones generadoras de tablas según el número de columnas, enviándoles como entradas los datos específicos para cada modelo. La figura 3.49 muestra el código para el caso de los tres modelos de simulación junto con los datos reales (tanto para la energía como para la ganancia bifacial), mientras que la figura 3.50 presenta el caso de los tres modelos sin datos reales, pero considerando también la eficiencia energética del panel.

```

def gen_table_sim():
    # Four variables scenario
    # PVsyst, NREL, Solar World and real data
    if globals.pvscheck == 1 and globals.nrelcheck == 1 and globals.swcheck == 1 and globals.realcheck == 1:
        if globals.radioselect == "BifacialGain":
            try:
                globals.pvsBifGain = empirical.bifGainCalc(globals.pvsBifEnergy, globals.pvsMonoEnergy)
                fourcolumns(globals.pvsBifGain, globals.nrelPercentBG, globals.swPercentBG,
                            globals.realBifacialGain, "PVsyst",
                            "NREL", "Solar World", "Measured data")
            except:
                MessageBox.showerror("Missing data", "Please load all the corresponding CSV files")

        elif globals.radioselect == "BifacialEnergy":
            try:
                globals.nrelBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.nrelBifRatio)
                globals.swBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.empiricalList)
                fourcolumns(globals.pvsBifEnergy, globals.nrelBifEnergy, globals.swBifEnergy,
                            globals.realBifacialEnergy,
                            "PVsyst", "NREL", "Solar World", "Measured data")
            except:
                MessageBox.showerror("Missing data", "The monofacial energy from PVsyst CSVs and the bifacial gains for"
                                     " each simulation tool are needed to calculate the bifacial energy")

```

Figura 3.49: Código de la función *gen_table_sim* para los tres modelos de simulación y datos reales en tables.py

```

# Three variables scenarios
# PVsyst, NREL and Solar World
elif globals.pvscheck == 1 and globals.nrelcheck == 1 and globals.swcheck == 1:
    if globals.radioselect == "BifacialGain":
        try:
            globals.pvsBifGain = empirical.bifGainCalc(globals.pvsBifEnergy, globals.pvsMonoEnergy)
            threecolumns(globals.pvsBifGain, globals.nrelPercentBG, globals.swPercentBG, "PVsyst", "NREL",
                         "Solar World")
        except:
            MessageBox.showerror("Missing data", "Please load all the corresponding CSV files")

    elif globals.radioselect == "BifacialEnergy":
        try:
            globals.nrelBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.nrelBifRatio)
            globals.swBifEnergy = empirical.bifEnergyCalc(globals.pvsMonoEnergy, globals.empiricalList)
            threecolumns(globals.pvsBifEnergy, globals.nrelBifEnergy, globals.swBifEnergy, "PVsyst", "NREL",
                         "Solar World")
        except:
            MessageBox.showerror("Missing data", "The monofacial energy from PVsyst CSVs and the bifacial gains for"
                                 " each simulation tool are needed to calculate the bifacial energy")

    elif globals.radioselect == "ModuleEnergy":
        try:
            globals.nrelEfficiency = empirical.modEffCalc(globals.nrelBifEnergy, globals.modpower)
            globals.swEfficiency = empirical.modEffCalc(globals.swBifEnergy, globals.modpower)
            threecolumns(globals.pvsEfficiency, globals.nrelEfficiency, globals.swEfficiency, "PVsyst",
                         "NREL",
                         "Solar World")
        except:
            MessageBox.showerror("Missing data",
                                 "The bifacial energy from each simulation tool and the module power"
                                 " at PVsyst are needed to calculate the module energy efficiency")

```

Figura 3.50: Código de la función *gen_table_sim* para los tres modelos de simulación en tables.py

Por último, la función *gen_table_errors* permite evaluar, al igual que la función anterior, la selección de modelos de simulación y variables energéticas, en este caso para invocar el cálculo de los errores RMSE y MBE, e insertarlos en tablas. En la figura 3.51 se muestra el código para el caso de los tres modelos de simulación a la vez, tanto para la ganancia como para la energía bifacial. De igual forma, esta misma función contiene casos para cada una de las combinaciones de simulación.

```
# PVsyst, NREL and Solar World
if globals.pvscheck == 1 and globals.nrelcheck == 1 and globals.swcheck == 1:
    if globals.radioselect == "BifacialGain":
        globals.rmsBGpvsyst = [statistics.rmse(i, j) for i, j in
                               zip(np.array(globals.pvsBifGain), np.array(globals.realBifacialGain))]
        globals.rmsBNrel = [statistics.rmse(i, j) for i, j in
                            zip(np.array(globals.nrelPercentBG), np.array(globals.realBifacialGain))]
        globals.rmsBGsw = [statistics.rmse(i, j) for i, j in
                           zip(np.array(globals.swPercentBG), np.array(globals.realBifacialGain))]

        globals.mbeBGpvsyst = [statistics.mbe(i, j) for i, j in
                               zip(np.array(globals.pvsBifGain), np.array(globals.realBifacialGain))]
        globals.mbeBNrel = [statistics.mbe(i, j) for i, j in
                            zip(np.array(globals.nrelPercentBG), np.array(globals.realBifacialGain))]
        globals.mbeBGsw = [statistics.mbe(i, j) for i, j in
                           zip(np.array(globals.swPercentBG), np.array(globals.realBifacialGain))]

        sixcolumns(globals.rmsBGpvsyst, globals.rmsBNrel, globals.rmsBGsw, globals.mbeBGpvsyst,
                   globals.mbeBNrel, globals.mbeBGsw, "RMSE PVsyst", "RMSE NREL", "RMSE Solar World",
                   "MBE PVsyst", "MBE NREL", "MBE Solar World")

    elif globals.radioselect == "BifacialEnergy":
        globals.rmsBEpvsyst = [statistics.rmse(i, j) for i, j in
                               zip(np.array(globals.pvsBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.rmsBEnrel = [statistics.rmse(i, j) for i, j in
                            zip(np.array(globals.nrelBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.rmsBEsw = [statistics.rmse(i, j) for i, j in
                           zip(np.array(globals.swBifEnergy), np.array(globals.realBifacialEnergy))]

        globals.mbeBEpvsyst = [statistics.mbe(i, j) for i, j in
                               zip(np.array(globals.pvsBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.mbeBEnrel = [statistics.mbe(i, j) for i, j in
                            zip(np.array(globals.nrelBifEnergy), np.array(globals.realBifacialEnergy))]
        globals.mbeBEsw = [statistics.mbe(i, j) for i, j in
                           zip(np.array(globals.swBifEnergy), np.array(globals.realBifacialEnergy))]

        sixcolumns(globals.rmsBEpvsyst, globals.rmsBEnrel, globals.rmsBEsw, globals.mbeBEpvsyst,
                   globals.mbeBEnrel, globals.mbeBEsw, "RMSE PVsyst", "RMSE NREL", "RMSE Solar World",
                   "MBE PVsyst", "MBE NREL", "MBE Solar World")
```

Figura 3.51: Código de la función *gen_table_errors* para los tres modelos de simulación en *tables.py*

En las secciones anteriores se detallan aspectos sobre los modelos de estimación energética utilizados en el proyecto, arquitectura de la herramienta de evaluación creada, especificaciones sobre el entorno de desarrollo, y los casos de simulación trabajados en las herramientas de software complementarias, así como los métodos diseñados. En el siguiente capítulo se hablará sobre la ejecución de la herramienta de evaluación creada, pruebas de funcionamiento y verificación de los resultados obtenidos.

Capítulo 4

Pruebas y verificación de la herramienta

En este capítulo se desarrolla un caso de ejecución del programa, desde su llamada inicial, pasando por los valores de entrada, las salidas que genera el programa, y por último se analizan y comparan los resultados contra valores de referencia.

4.1. Ejecución del programa

Para que la herramienta de evaluación funcione correctamente, es necesario tomar en cuenta que:

1. Todas las bibliotecas necesarias estén instaladas en el sistema.
2. Todos los archivos que conforman el programa se encuentren en una misma carpeta.

El programa se puede inicializar desde el IDE de preferencia por el usuario, ejecutando el módulo llamado *gui.py*, o llamando a la terminal de Windows (o al *Windows Power Shell*) desde la carpeta donde se encuentren todos los archivos del programa y escribiendo el comando `python gui.py`

En la figura 4.1 se observa el *Windows Power Shell* con el comando necesario para ejecutar el archivo desde la carpeta en el sistema donde está guardado. Si no existe algún problema con la instalación, se debería abrir la GUI principal del programa (figura 3.7).

```
PS D:\TEC\2022 - I Semestre\Trabajo Final de Graduación\Códigos Python\BifacialTool\BifacialTool_v2> python gui.py
```

Figura 4.1: Comando inicial del programa desde *Windows Power Shell*

4.2. Entradas del programa

4.2.1. Carga de archivos de PVsyst

Si el usuario selecciona la opción de *PVsyst* del menú desplegable en la GUI principal (figura 4.2), se abrirá una GUI auxiliar para este modelo de estimación. Una vez en esta ventana (figura 4.3), el usuario puede marcar el botón con la leyenda “*Common albedo as 0?*” y presionar el botón “*Search file*” para buscar en el sistema y cargar el archivo CSV que contiene la información de energía monofacial, o quitar la selección del primer botón y presionar “*Search file*” nuevamente para cargar los archivos correspondientes a la energía bifacial y eficiencia del panel. La ventana para buscar y cargar archivos se puede observar en la figura 4.4. En esta misma ventana auxiliar el usuario puede escribir el valor de la potencia nominal del panel que utilizó para las simulaciones, en el cuadro que se encuentra en la parte inferior, y seguidamente presionar el botón “*Set*” para guardar este valor en el programa. En el caso de demostración que se estará desarrollando se usa una potencia nominal de 0.4 kWp (400 Wp), como se mencionó en la sección 3.4.

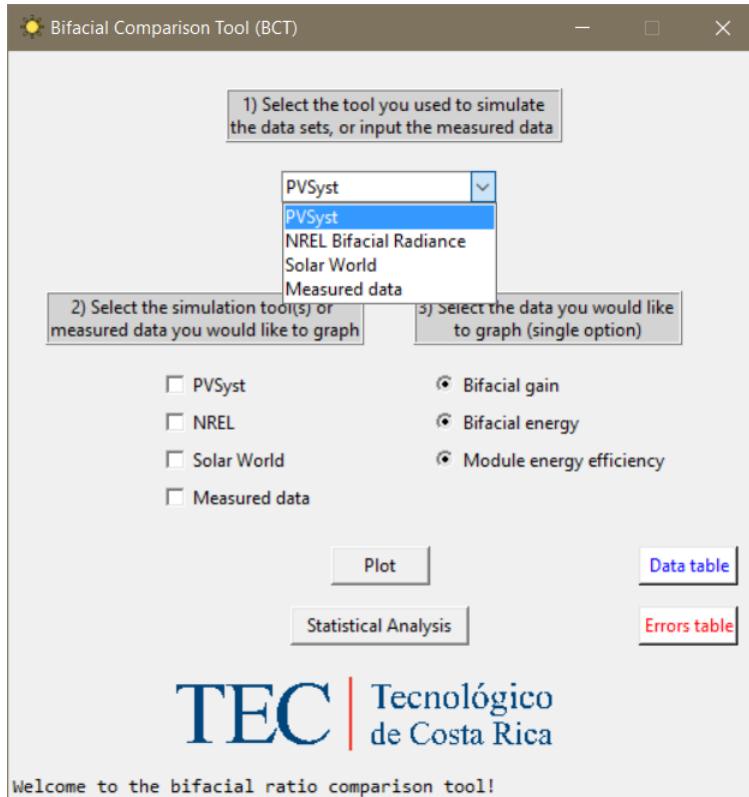


Figura 4.2: GUI principal con selección de PVsyst

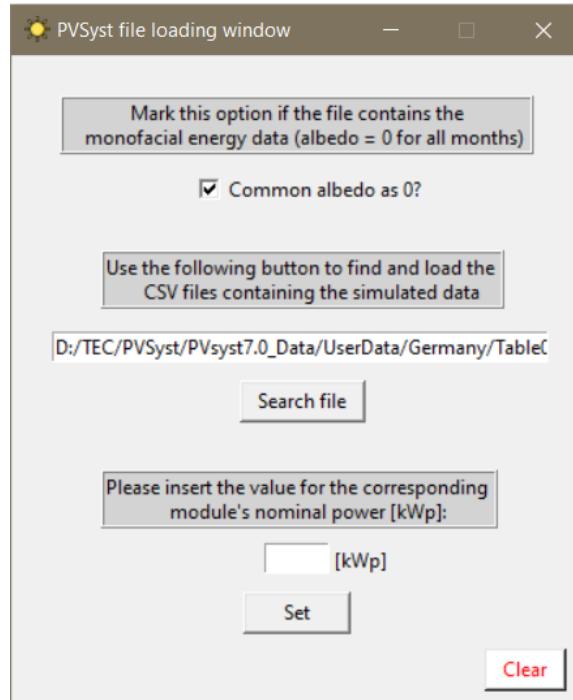


Figura 4.3: GUI auxiliar con opciones para PVsyst

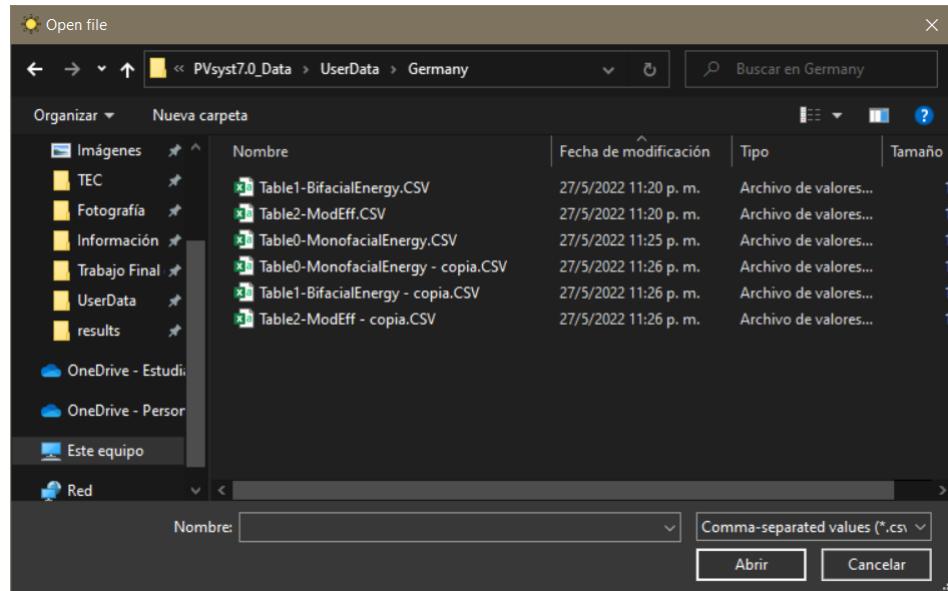


Figura 4.4: Ventana para cargar archivos CSV en caso de PVsyst

Si no existe algún error en la carga del archivo, el usuario debería ver una notificación sobre la carga exitosa de este (figura 4.5), lo mismo a la hora de establecer el valor de potencia nominal del panel (figura 4.6).

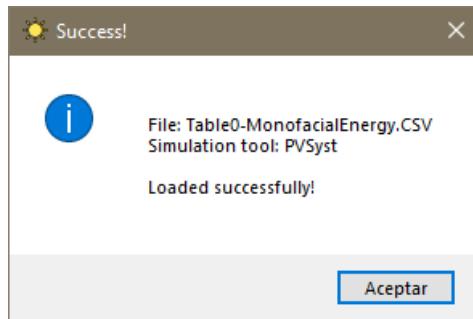


Figura 4.5: Notificación de carga exitosa de CSV en PVsyst

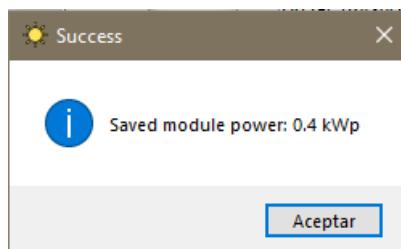


Figura 4.6: Notificación de potencia asignada correctamente en PVsyst

4.2.2. Carga de archivos de NREL Bifacial Radiance

Si el usuario selecciona la opción *NREL Bifacial Radiance* del menú desplegable en la ventana principal (figura 4.7) podrá acceder a la GUI auxiliar para este modelo de estimación energética (figura 4.8). Una vez ahí, al presionar el botón “Search file” el programa abrirá una ventana adicional (figura 4.9) donde el usuario podrá buscar y cargar los archivos CSV correspondientes a las simulaciones de NREL para cada mes, y almacenar sus datos en el programa.

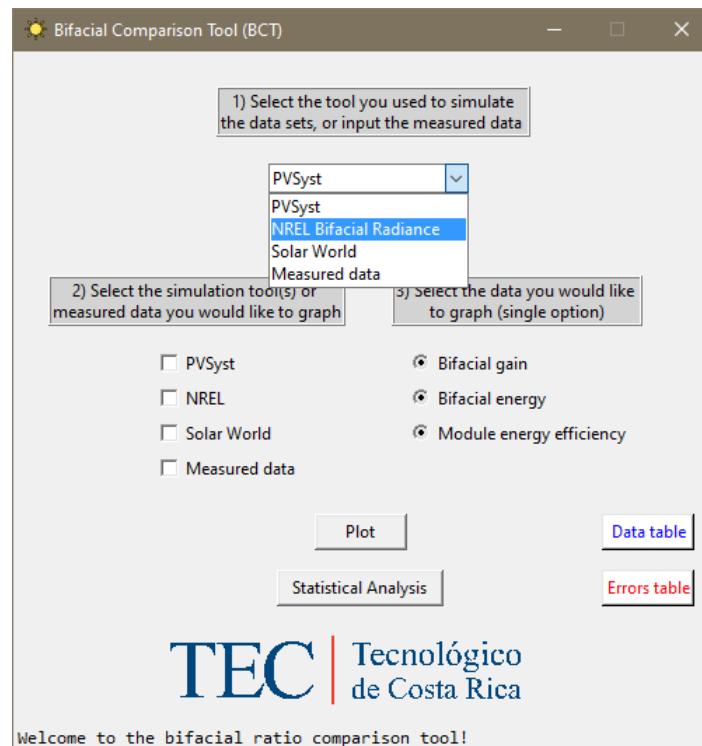


Figura 4.7: GUI principal con selección de NREL

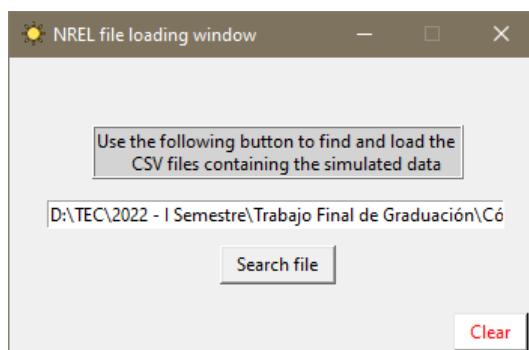


Figura 4.8: GUI auxiliar con opciones para NREL

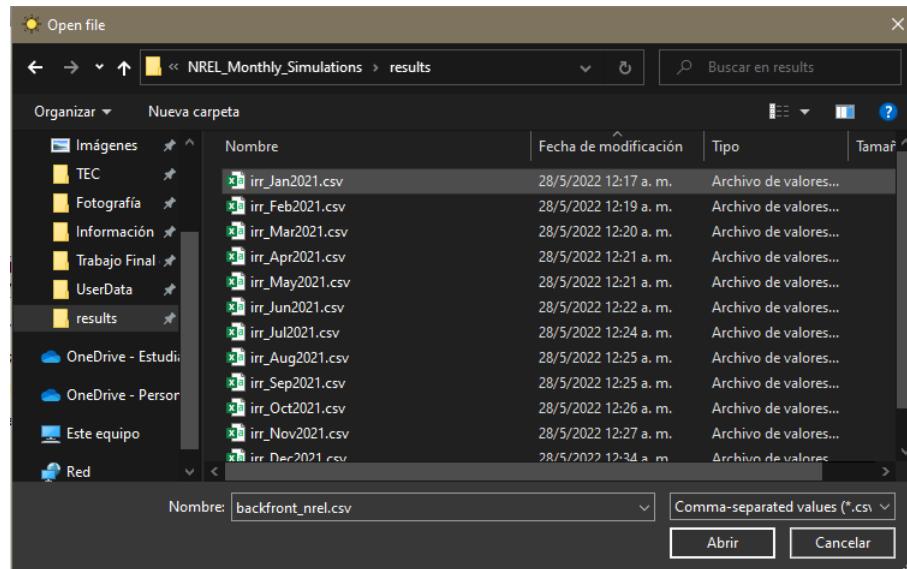


Figura 4.9: Ventana para cargar archivos CSV en caso de NREL

Si no existe algún error en la carga del archivo, el usuario debería ver una notificación sobre la carga exitosa de este (figura 4.10) cada vez que carga un CSV con la información que desea evaluar.

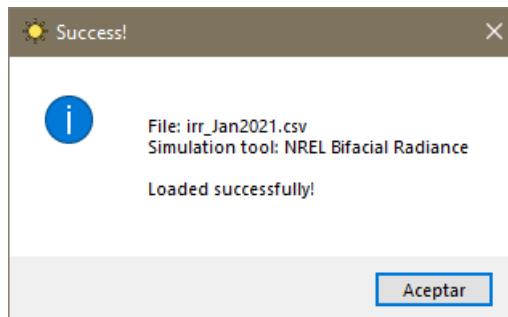


Figura 4.10: Notificación de carga exitosa de CSV en NREL

4.2.3. Carga de datos para Solar World

Cuando el usuario elige la opción de *Solar World* en el menú desplegable de la GUI principal (figura 4.11) el programa abrirá una GUI auxiliar para este caso (figura 4.12). Aquí el usuario puede ingresar valores mensuales de albedo (variables o constantes), y definir el factor de bifacialidad, GCR y *clearance height*. Para el caso de demostración se cargaron los valores de albedo del cuadro 3.3, y los demás valores según el cuadro 3.2. Cuando el usuario presiona el botón “Calculate” (ubicado en la parte inferior de la ventana) el programa se encarga de tomar los valores ingresados por el usuario y calcular la ganancia bifacial a partir de la ecuación 3.2, también visible en la ventana de la GUI para este caso.

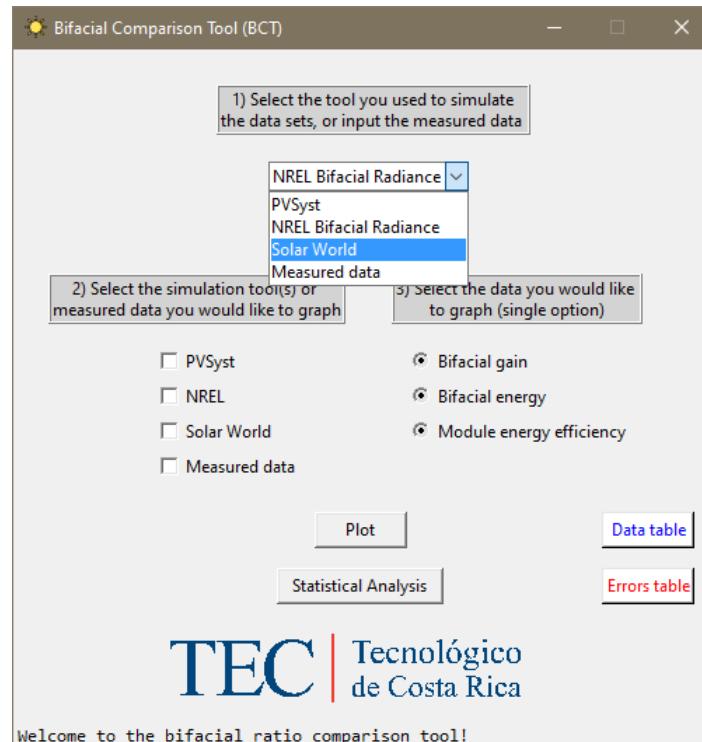


Figura 4.11: GUI principal con selección de Solar World

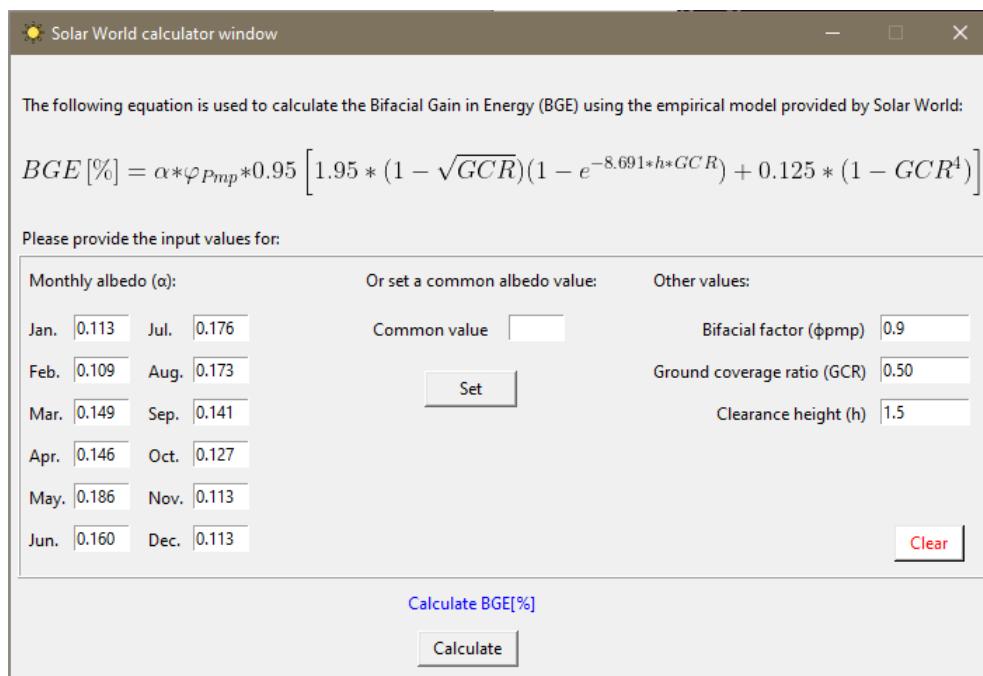


Figura 4.12: GUI auxiliar con opciones para Solar World

Si no hay alguna irregularidad con el ingreso de los datos (como por ejemplo usar una

coma para separar los decimales en lugar de un punto, o ingresar un valor no numérico), el usuario debería ver una notificación sobre el cálculo exitoso de la ganancia bifacial (figura 4.13).

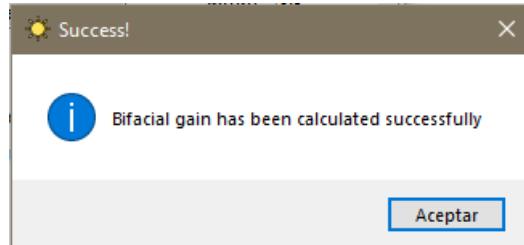


Figura 4.13: Notificación de cálculo exitoso en Solar World

4.2.4. Carga de datos reales

Cuando el usuario haga “click” en la opción de *Measured data* del menú *dropdown* de la GUI principal (figura 4.14), se desplegará una GUI adicional donde se pueden ingresar los datos medidos en campo, que serán utilizados como referencia para determinar la exactitud de las simulaciones anteriores. En la figura 4.15 se puede observar el caso de demostración, donde se ingresaron como referencia los valores de energía bifacial y ganancia bifacial medidos en el experimento realizado en [9], mostrados en la figura 3.13 de la sección 3.4.

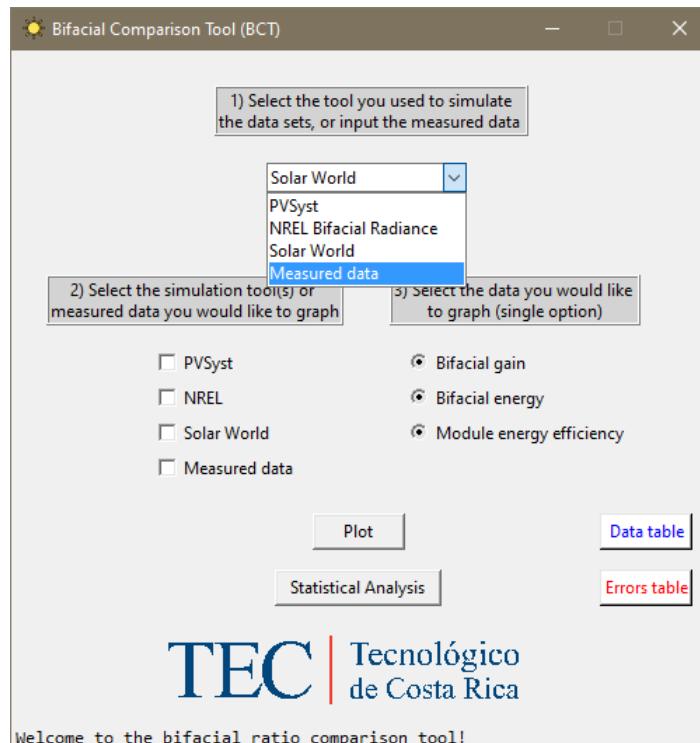


Figura 4.14: GUI principal con selección de datos medidos

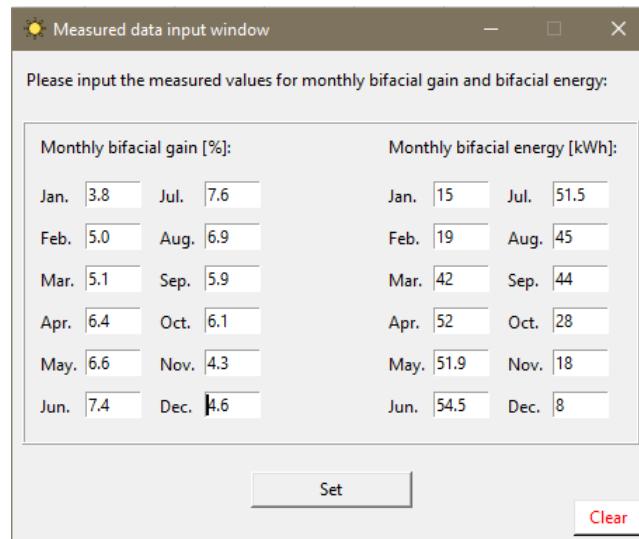


Figura 4.15: GUI auxiliar con espacios para introducir datos medidos

Al igual que en la ventana de Solar World, si los datos se ingresan sin irregularidades, al presionar el botón “*Set*” se debería desplegar una notificación indicando que los valores fueron cargados correctamente (figura 4.16).

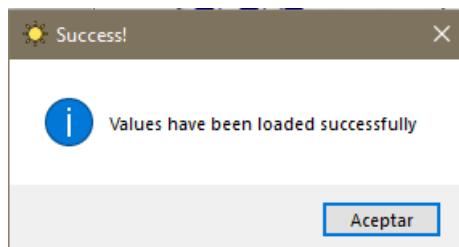


Figura 4.16: Notificación de carga exitosa de datos medidos

4.3. Salidas del programa

4.3.1. Gráficas y tablas entre simulaciones

Una vez cargados todos los archivos CSV y los datos para el modelo empírico, se puede proceder a la generación de gráficas y tablas para evaluar los resultados. A través de la GUI principal, la herramienta diseñada permite desplegar la información de uno, dos o tres modelos simulados a la vez, ya sea para mostrar los datos de ganancia bifacial, energía bifacial o eficiencia energética del módulo.

Para desplegar toda la información es necesario realizar algunos cálculos, los cuales se ejecutan a la hora de graficar o tabular los datos, ya que existe interdependencia entre algunas variables. Para el caso de *PVSyst*, la importación de los archivos CSV permite extraer la energía monofacial, energía bifacial y la eficiencia energética del módulo. La ganancia bifacial (BGE) no se encuentra directamente en los archivos CSV generados al ejecutar las simulaciones, por lo que se debe calcular mediante la ecuación 4.1, usando las energías monofacial y bifacial extraídas anteriormente de los archivos.

$$BGE[\%] = \left(\frac{Energia\ bifacial\ [kWh] - Energia\ monofacial\ [kWh]}{Energia\ monofacial\ [kWh]} \right) \times 100 \quad (4.1)$$

En los casos de *NREL* y *Solar World*, originalmente solo se obtiene la BGE mediante la importación de archivos CSV y el cálculo usando el modelo empírico, respectivamente. Para estimar la energía bifacial de ambos casos se necesita la energía monofacial de referencia (obtenida de los archivos de *PVSyst*), y la BGE correspondiente a cada uno de estos modelos. El cálculo se puede realizar utilizando la ecuación 4.2.

$$Energia\ bif.\ [kWh] = Energia\ mono.\ [kWh] + (Energia\ mono.\ [kWh] \times BGE\ [-]) \quad (4.2)$$

De la misma forma, para estos dos casos (*NREL* y *Solar World*) también es necesario realizar un cálculo para obtener la eficiencia energética del módulo, mediante la ecuación 4.3. Para esto se debe tener la energía bifacial (calculada en el punto anterior con la ecuación 4.2) y la potencia nominal del módulo utilizado en las simulaciones (dato que se ingresa en la GUI de *PVSyst*, de la figura 4.3).

$$Eficiencia\ mod.\ [kWh/kWp] = \frac{Energia\ bif.\ [kWh]}{Potencia\ nom.\ modulo\ [kWp]} \quad (4.3)$$

Una vez calculados estos valores es posible visualizarlos mediante gráficas y tablas. Por ejemplo, si en la GUI principal (figura 4.17) el usuario selecciona los tres modelos de estimación (sección central izquierda), elige la opción de *Bifacial gain* (sección central

derecha) y presiona el botón con la leyenda “*Plot*”, podrá generar una gráfica de esta variable en el caso de cada modelo para cada mes del año (figura 4.18a), mientras que si presiona el botón con la leyenda “*Data table*” podrá generar una tabla con los valores exactos presentados en la gráfica, para ayudar con la visualización (figura 4.18b).

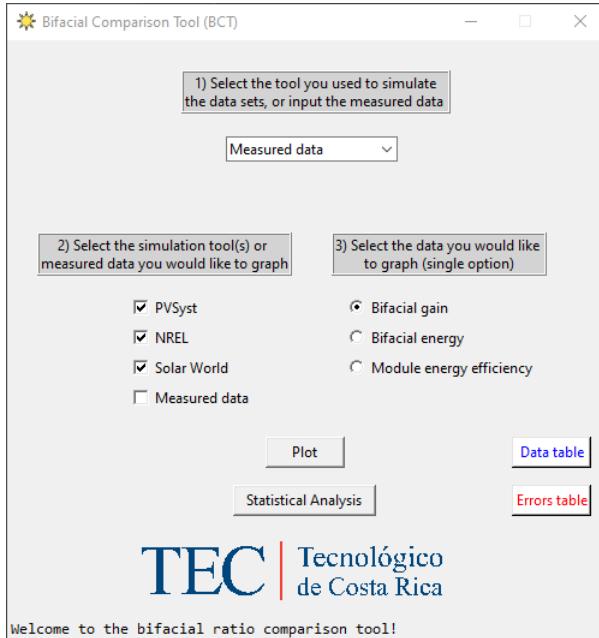


Figura 4.17: GUI principal con la selección de los tres modelos de estimación energética

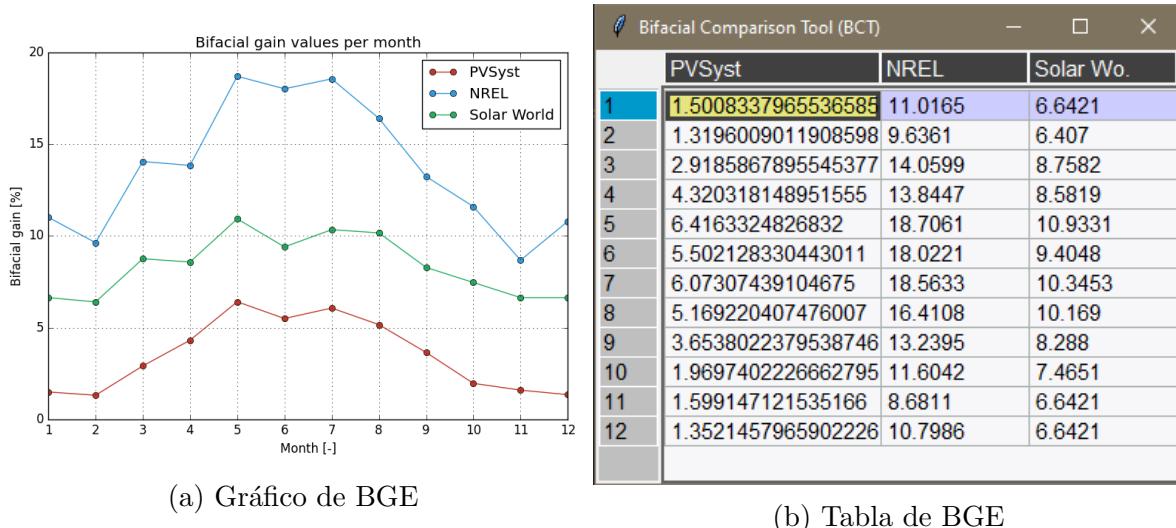
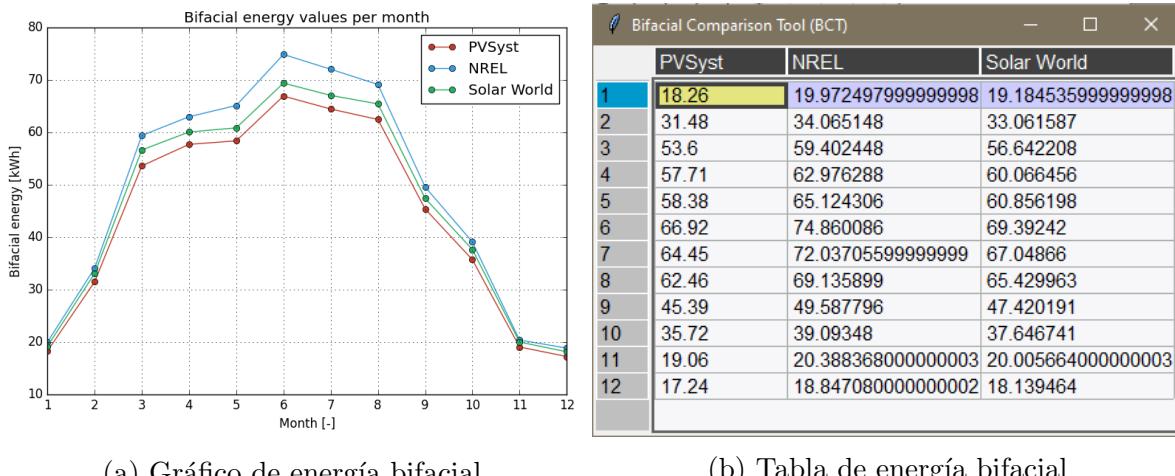


Figura 4.18: Ganancia bifacial simulada para un año, correspondiente a los tres modelos de simulación

Usando el mismo procedimiento, si el usuario selecciona la opción de *Bifacial energy* del menú principal podrá generar la gráfica de la figura 4.19a y la tabla de la figura

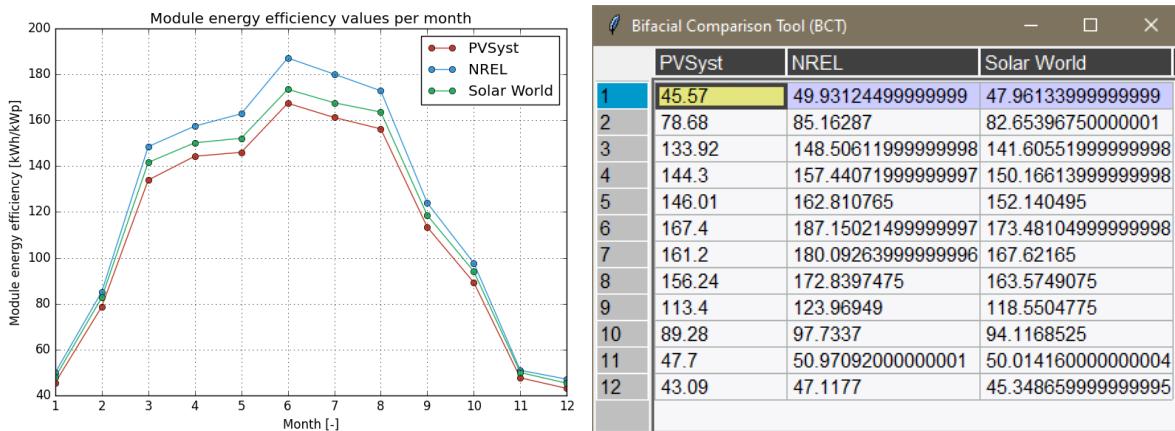
4.19b, mientras que si elige la opción *Module energy efficiency* puede crear la gráfica y tabla correspondientes a las figuras 4.20a y 4.20b.



(a) Gráfico de energía bifacial

(b) Tabla de energía bifacial

Figura 4.19: Energía bifacial simulada para un año, correspondiente a los tres modelos de simulación



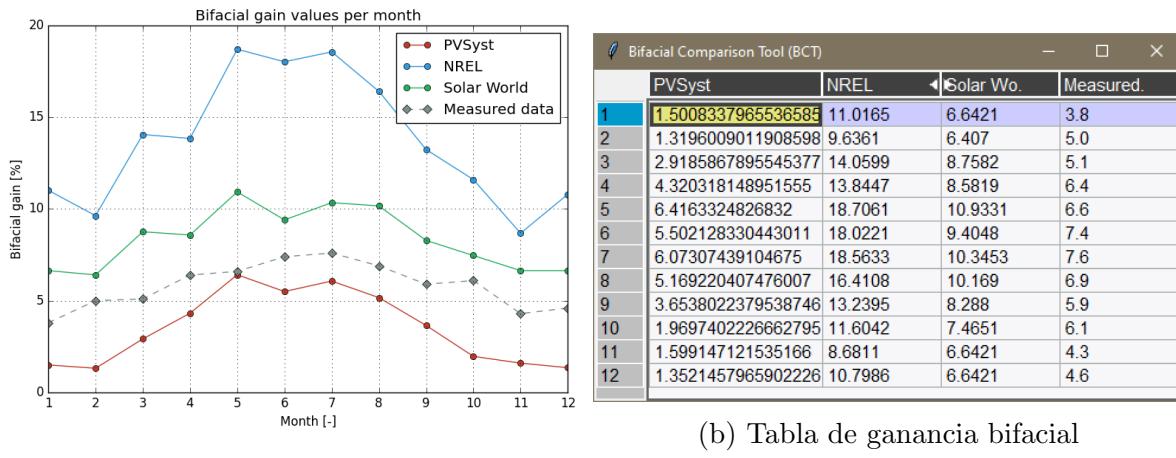
(a) Gráfico de eficiencia energética del módulo (b) Tabla de eficiencia energética del módulo

Figura 4.20: Eficiencia energética del modulo simulada para un año, correspondiente a los tres modelos de simulación

4.3.2. Gráficas y tablas con datos reales

De manera similar, la herramienta permite cargar datos reales medidas en campo correspondientes a la ganancia bifacial y la energía bifacial, para compararlos visualmente contra las simulaciones. Los datos utilizados como referencia se pueden encontrar en la sección 4.4 referente a la validación de resultados. La figura 4.21 muestra en una misma gráfica y tabla los valores simulados y medidos para la BGE, mientras que la figura 4.22

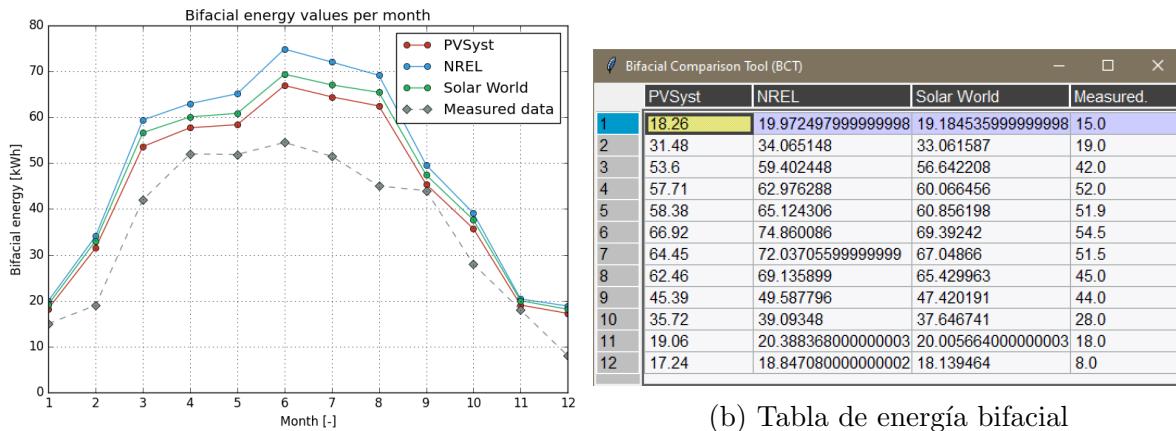
muestra la representación de la energía bifacial. Si el usuario así lo desea, también podría graficar y tabular la cantidad de modelos que desee, o solamente los datos medidos.



(a) Gráfico de ganancia bifacial

(b) Tabla de ganancia bifacial

Figura 4.21: Ganancia bifacial para un año, correspondiente a los tres modelos de simulación y a los datos reales



(a) Gráfico de energía bifacial

(b) Tabla de energía bifacial

Figura 4.22: Energía bifacial para un año, correspondiente a los tres modelos de simulación y a los datos reales

4.3.3. Gráficas y tablas de errores

La herramienta diseñada también ofrece la posibilidad de calcular valores estadísticos que permiten evaluar la exactitud de las simulaciones, es decir, qué tanto se acercan a los valores reales de referencia. Estos valores son el *Root Mean Squared Error* (RMSE) y el *Mean Bias Error* (MBE).

El RMSE es una medición de exactitud proporcional al tamaño del error cuadrático. Permite conocer un valor absoluto, porcentual, de qué tan lejos está un valor respecto a su referencia. Está definido por la ecuación 4.4, donde y_i es el valor simulado, \bar{y}_i es el valor esperado y N es la cantidad de datos en el conjunto.

$$RMSE_{absoluto} = \sqrt{\frac{\sum_{i=1}^N (y_i - \bar{y}_i)^2}{N}} \quad (4.4)$$

Por otra parte, el MBE captura el sesgo promedio en la predicción. Este error es utilizado principalmente para estimar el sesgo promedio en el modelo y decidir si se deben aplicar correcciones para mejorar el modelado. Un MBE positivo indica que la simulación está subestimando el valor de referencia, mientras que un valor negativo indica que se está sobreestimando. Se define mediante la ecuación 4.5.

$$MBE_{absoluto} = \frac{\sum_{i=1}^N (y_i - \bar{y}_i)}{N} \quad (4.5)$$

Después de ingresar todos los archivos y datos necesarios para las simulaciones y datos reales, además de calcular y graficar sus variables, en el menú principal el usuario puede seleccionar los modelos para los que desea calcular los errores y seleccionar si quiere ver la información para *bifacial gain* o *bifacial energy*. Al presionar el botón “*Statistical Analysis*” se calculan estos errores y se genera una gráfica para visualizarlos, mientras que si se presiona el botón “*Errors table*” se generará una tabla con dichos valores de RMSE y MBE. En la figura 4.23 se puede observar la gráfica para la ganancia bifacial, y su respectiva tabla se puede encontrar seccionada en dos partes en la figura 4.24, mientras que en la figura 4.25 está la gráfica para el caso de los errores para la energía bifacial, y en la figura 4.26 su tabla correspondiente (también dividida).

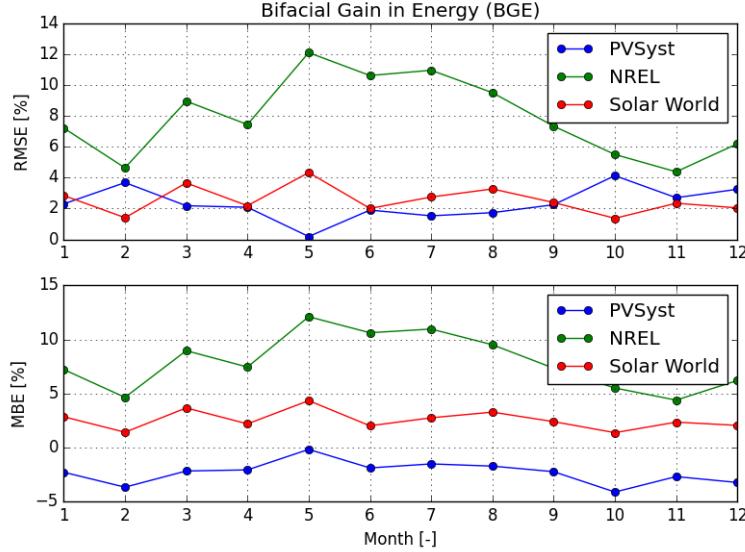


Figura 4.23: RMSE y MBE para la ganancia bifacial

	RMSE PVsyst	RMSE NREL	RMSE Solar W.		MBE PVsyst	MBE NREL	MBE Solar Wo.
1	2.299166203446341	7.216500000000001	2.842100000000003		-2.299166203446341	7.216500000000001	2.842100000000003
2	3.68039909880914	4.636100000000001	1.407		-3.68039909880914	4.636100000000001	1.407
3	2.181413210445462	8.959900000000001	3.658200000000001		-2.181413210445462	8.959900000000001	3.658200000000001
4	2.079681851048445	7.444699999999999	2.181899999999999		-2.079681851048445	7.444699999999999	2.181899999999999
5	0.18366751731679987	12.1061	4.3331		-0.18366751731679987	12.1061	4.3331
6	1.8978716695569897	10.622099999999998	2.0047999999999995		-1.8978716695569897	10.622099999999998	2.0047999999999995
7	1.52692560895325	10.963300000000002	2.745300000000003		-1.52692560895325	10.963300000000002	2.745300000000003
8	1.7307795925239935	9.510799999999998	3.269		-1.7307795925239935	9.510799999999998	3.269
9	2.2461977620461258	7.339499999999999	2.388		-2.2461977620461258	7.339499999999999	2.388
10	4.13025977733372	5.504200000000001	1.3651		-4.13025977733372	5.504200000000001	1.3651
11	2.7008528784648336	4.381100000000001	2.342100000000003		-2.7008528784648336	4.381100000000001	2.342100000000003
12	3.2478542034097773	6.198600000000001	2.042100000000005		-3.2478542034097773	6.198600000000001	2.042100000000005

(a) RMSE para BGE

(b) MBE para BGE

Figura 4.24: Tablas de RMSE y MBE para la ganancia bifacial, correspondientes a los tres modelos de simulación

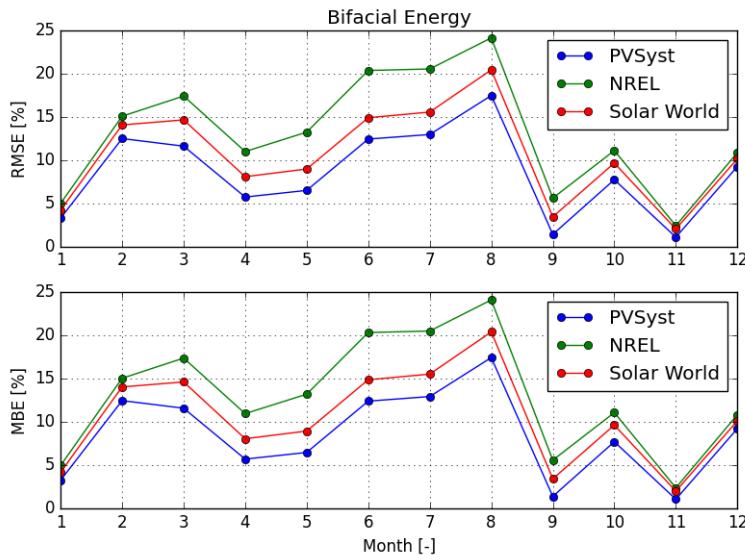


Figura 4.25: RMSE y MBE para la energía bifacial

	RMSE PVsyst	RMSE NREL	RMSE Solar W.		MBE PVsyst	MBE NREL	MBE Solar World
1	3.260000000000001e+16	4.972497999999998	4.184535999999998		3.260000000000001e+16	4.972497999999998	4.184535999999998
2	12.48	15.065148	14.061587000000003		12.48	15.065148	14.061587000000003
3	11.600000000000001	17.402448	14.64220799999997		11.600000000000001	17.402448	14.64220799999997
4	5.710000000000001	10.976287999999997	8.066456000000002		5.710000000000001	10.976287999999997	8.066456000000002
5	6.480000000000004	13.224306000000006	8.956198		6.480000000000004	13.224306000000006	8.956198
6	12.420000000000002	20.360085999999995	14.892420000000001		12.420000000000002	20.360085999999995	14.892420000000001
7	12.950000000000003	20.537055999999993	15.548659999999998		12.950000000000003	20.537055999999993	15.548659999999998
8	17.46	24.135898999999995	20.429963		17.46	24.135898999999995	20.429963
9	1.390000000000006	5.587795999999997	3.4201910000000026		1.390000000000006	5.587795999999997	3.4201910000000026
10	7.719999999999999	11.09348	9.64674099999999		7.719999999999999	11.09348	9.64674099999999
11	1.0599999999999987	2.3883680000000034	2.005664000000003		1.0599999999999987	2.3883680000000034	2.005664000000003
12	9.239999999999998	10.847080000000002	10.139464		9.239999999999998	10.847080000000002	10.139464

(a) RMSE para energía bifacial

(b) MBE para energía bifacial

Figura 4.26: Tablas de RMSE y MBE para la energía bifacial, correspondientes a los tres modelos de simulación

4.4. Análisis y validación de resultados simulados y medidos

Cada uno de los tres modelos de estimación utilizados para este proyecto permite tomar en cuenta diferentes parámetros para generar sus simulaciones, variando en su nivel de complejidad. Los parámetros tomados en cuenta para las simulaciones de demostración se pueden encontrar en el cuadro a continuación (4.1):

Cuadro 4.1: Parámetros utilizados para las simulaciones, según modelo simulado

	PVSyst	NREL	Solar World
Albedo	✓	✓	✓
GCR	✓	✓	✓
Fact. Bif.		✓	✓
Altura módulo	✓	✓	✓
Inclinación	✓	✓	
Azimuth	✓	✓	
Info. de módulo	✓	✓	
Info. de inversor	✓		
Núm. de módulos	✓	✓	
Datos meteorológicos	✓	✓	
Ubicación geográfica	✓	✓	
Sombreado	✓		
Pérdidas	✓		

En el caso de *PVSyst*, este simulador toma en cuenta valores pequeños de sombreado y pérdidas al trabajar con sistemas bifaciales.

Como se mencionó en la sección 3.4, se tomaron las mediciones de energía y ganancia bifacial capturadas en Bernburg (Alemania), presentes en el cuadro 3.4, como referencia de datos reales. Además de la diferencia en los paneles solares usados en las simulaciones de demostración y las mediciones del experimento de referencia, así como consideraciones de temperatura y aspectos estructurales del montaje del panel que no se tomaron en cuenta en las simulaciones propias del presente proyecto, para las simulaciones complementarias al experimento de [9] se utilizó una base de datos meteorológica personalizada a la ubicación del experimento, con datos más exactos sobre las condiciones solares y de nubosidad en esta locación durante el período de las pruebas, mientras que en las simulaciones propias se usaron las bases de datos disponibles por defecto en los programas.

4.4.1. Ganancia bifacial

Para el caso de ganancia bifacial, como se puede observar en la gráfica y tabla de la figura 4.21, los valores simulados en *PVSyst* son los más conservadores, ya que se encuentran por debajo de los valores de referencia, mientras que *Solar World* y *NREL* obtuvieron valores de ganancia bifacial superiores a la referencia.

Los valores de error RMSE de las figuras 4.23 y 4.24a permiten observar que la desviación de los datos para *PVSyst* varía entre 0.18 % y 4.13 % (con un promedio anual de 2.53 %), para *NREL* se encuentra entre 4.38 % y 12.11 % (promedio anual de 8.28 %), mientras que para *Solar World* está entre 1.36 % y 4.33 % (promedio anual de 2.68 %). Con estos valores de desviación, se puede sugerir que el modelo de *PVSyst* es el que mejor se aproxima a los datos reales en este caso (menor error cuadrático), seguido de *Solar World* y por último *NREL* (mayor error cuadrático). El cuadro 4.2 recopila estos valores de RMSE.

Cuadro 4.2: Valores menores y mayores de RMSE para la ganancia bifacial, en el caso de los tres simuladores

	PVSyst	NREL	Solar World
RMSE Menor [%]	0.18	4.38	1.36
RMSE Mayor [%]	4.13	12.11	4.33
Promedio anual [%]	2.53	8.28	2.68

Por otra parte, el análisis de MBE de las figuras 4.23 y 4.24b permite verificar cuándo un valor se está sobreestimando (negativo) o subestimando (positivo). En este caso solo los valores correspondientes a *PVSyst* se están subestimando. Los demás, como se mencionó anteriormente, están por encima de la referencia.

Estos resultados son esperables, ya que según [9] la herramienta de *NREL* tiende a sobreestimar el valor de ganancia bifacial. Esto se puede deber principalmente a que este modelo utiliza una base datos meteorológica propia de esta organización, la cual en este caso utilizó automáticamente un archivo correspondiente a Berlín como ubicación geográfica más cercana a la latitud y longitud provistas al simulador, a falta de datos específicos sobre la ubicación exacta. De igual forma, este simulador permite incluir solamente algunas características del módulo fotovoltaico que se quiere utilizar, sin tomar en cuenta su potencia nominal. También, debido a la complejidad de su modelado

usando trazado de rayos se pueden complicar las estimaciones al simular un solo módulo.

Para el caso de *PVSyst*, este simulador es el que permite ejecutar simulaciones más completas, pero proporcionalmente más complejas de configurar, ya que se pueden tomar en cuenta una mayor cantidad de variables (en comparación con los otros dos modelos), como la potencia nominal del módulo, y tiene mayor diversidad de bases de datos meteorológicas, lo que abre la posibilidad de probar con diferentes opciones. El hecho de que sus valores estimados se encuentren por debajo de los valores de referencia se puede deber a que, al tomar en cuenta más parámetros del módulo como sus temperaturas de operación y algunas pérdidas, así como incluir un inversor y una parte de sombreado estructural, esto podría afectar la estimación de energía en la cara posterior del panel.

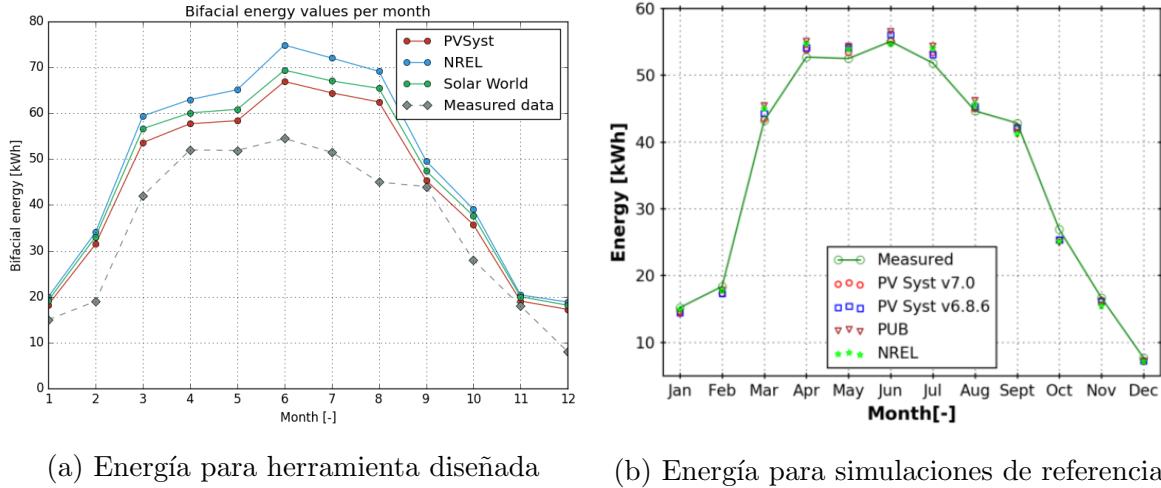
El modelo empírico de *Solar World* es el más simplificado de los tres, ya que no toma en cuenta tantas variables. Al no depender de datos meteorológicos ni de las características del panel (como su potencia nominal) permite crear aproximaciones relativamente conservadoras, independientes de pérdidas y sombreado. Sin embargo, esto podría limitar su uso ya que las mediciones reales dependen de más factores que en este caso no se toman en cuenta.

Es necesario considerar que la generación de gráficas y tablas para la ganancia bifacial (y sus errores asociados) es un proceso que no se ejecuta en el estudio de referencia de [9], por lo que no se pueden realizar comparaciones gráficas de estos valores, como las hechas en la siguiente sección para la energía bifacial. Sin embargo, sí es posible comparar los valores de ganancia bifacial simulados en este proyecto contra los medidos en dicho estudio.

4.4.2. Energía bifacial

Para la energía bifacial también se toman como referencia las mediciones de Bernburg. En este caso, al observar la figura 4.22 se puede ver cómo los datos simulados siguen aproximadamente la misma tendencia de los datos medidos, sin embargo, todos están siendo sobreestimados por los modelos de simulación. Esto se debe a que, como se mencionó en la sección 3.4, al utilizar un modelo de panel solar con mayor potencia (respecto al de las mediciones de campo) en todos los modelos de simulación, con excepción de *Solar World*, se puede esperar una mayor producción de energía bajo las mismas condiciones (irradiancia, nubosidad, etc.)

La figura 4.27 muestra una comparación lado a lado de la gráfica de energía bifacial obtenida mediante la herramienta diseñada contra el caso de las simulaciones generadas en [9], tomadas como referencia. Ambas gráficas tienen en común la simulación para las herramientas de *NREL* y *PVSyst*, así como los valores medidos. Se puede observar cómo los valores en común siguen aproximadamente la misma tendencia en ambas gráficas, sin embargo en el caso de la herramienta diseñada las simulaciones tienen magnitudes mayores, por las razones mencionadas anteriormente.



(a) Energía para herramienta diseñada

(b) Energía para simulaciones de referencia

Figura 4.27: Energía bifacial para modelos simulados de la herramienta diseñada vs simulaciones de referencia tomadas de [9]

Para este caso de la herramienta diseñada, los valores de desviación en RMSE varían para *PVsyst* entre 1.06 % y 17.46 % (promedio anual de 9.7 %), para *NREL* están entre 2.39 % y 24.14 % (promedio anual de 14.57 %), y por último para *Solar World* fluctúan entre 2.01 % y 20.43 % (promedio anual de 11.79 %). Al igual que para la ganancia bifacial, *PVsyst* tiene el menor error cuadrático y *NREL* tiene el mayor error, mientras que *Solar World* se encuentra en el medio. Estos valores están resumidos en el cuadro 4.3.

Cuadro 4.3: Valores menores y mayores de RMSE para la energía bifacial, en el caso de los tres simuladores.

	PVsyst	NREL	Solar World
RMSE Menor [%]	1.06	2.39	2.01
RMSE Mayor [%]	17.46	24.14	20.43
Promedio anual [%]	9.70	14.57	11.79

Para el análisis del MBE, todos los valores son negativos debido a que todos los simuladores sobreestimaron sus cálculos. Al comparar la figura 4.28 contra la figura 4.29 (tomada como referencia de [9] a partir de las mediciones y simulaciones de Bernburg), las cuales tienen en común los modelos de *NREL* y *PVsyst*, es posible observar que los

errores de RMSE en el caso de este proyecto tienen magnitudes mayores que los RMSE de referencia, lo cual está dentro del comportamiento esperado debido a las mismas razones que las diferencias en la ganancia bifacial.

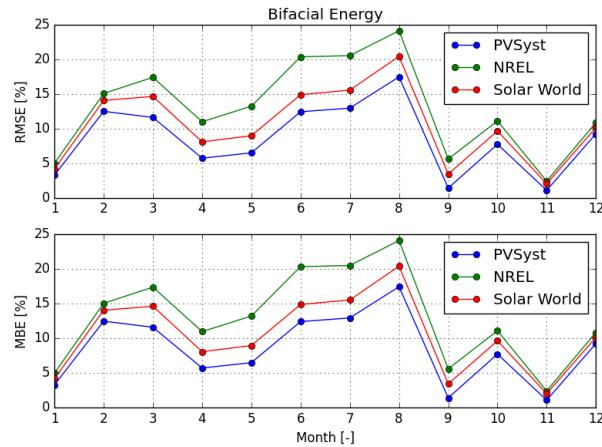


Figura 4.28: RMSE y MBE para la energía bifacial de la herramienta diseñada

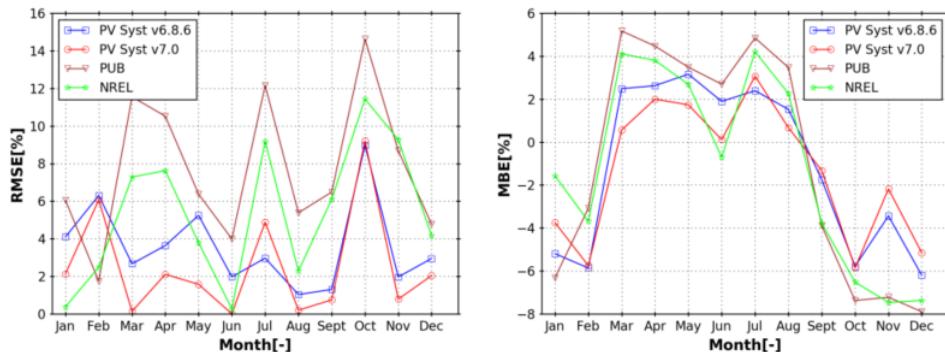


Figura 4.29: RMSE y MBE para la energía bifacial del caso de referencia [9]

4.5. Síntesis

El presente capítulo mostró el proceso para inicializar y ejecutar la herramienta desarrollada para un caso de simulación, tomando en cuenta sus entradas y salidas, así como una validación de la herramienta al incorporar datos reales.

Se ejecutaron simulaciones para los modelos de *PVSyst*, *NREL* y *Solar World*, permitiendo generar gráficas y tablas para hacer comparaciones entre sus valores de ganancia bifacial, energía bifacial y eficiencia energética del módulo utilizado en dichas simulaciones.

Posteriormente, al incorporar datos reales medidos en campo como referencia fue posible estimar la desviación entre los valores simulados y medidos mediante el RMSE y MBE, para ayudar con la determinación del modelo de simulación más exacto. El modelo de *PVSyst* presentó en promedio un error RMSE de 2.53 % para la BGE, y un 9.70 % en promedio para la energía bifacial, sugiriendo que puede ser el modelo más exacto entre las simulaciones. Por otra parte, el menos exacto aparenta ser el modelo de *NREL*, con un RMSE promedio de 8.28 % para la BGE y de 14.57 % en promedio para la energía bifacial. El modelo de *Solar World* se encuentra en un punto medio en ambos casos. Sin embargo, para tener certeza sobre esta exactitud es necesario realizar análisis estadísticos más profundos, lo cual no se encuentra en el enfoque del proyecto.

Estos resultados obtenidos presentan diferencias respecto a las simulaciones de referencia, lo que representa un comportamiento esperado, tanto de tendencias como de magnitudes. Las principales diferencias que se pueden encontrar entre el proyecto actual y el estudio de referencia son:

- Versiones más nuevas de los simuladores NREL y PVSyst para el presente proyecto.
- La base de datos meteorológica en el estudio de referencia fue creada específicamente para la locación de interés, mientras que en el presente proyecto se usaron bases de datos predeterminadas para los simuladores, las cuales usan ubicaciones aproximadas al lugar de interés, y pueden presentar datos inexactos.
- Las simulaciones de este proyecto utilizaron un modelo de panel solar diferente y de mayor potencia (400 Wp) que en el estudio de referencia (300 Wp).
- En el estudio de referencia no se hace un análisis de la eficiencia energética del módulo contra mediciones reales.
- En el estudio de referencia no se hace una comparación de tendencias y errores para la ganancia bifacial, y no se toma en cuenta el simulador de Solar World para la comparación de energía bifacial.

Como se puede observar, las diferencias son ocasionadas principalmente por las herramientas de simulación externas. De igual forma, este trabajo pretende complementar y

expandir la labor realizada por otras personas e instituciones para ejecutar evaluaciones de estimación energética en módulos fotovoltaicos bifaciales, con la meta de colaborar con la creación de conocimiento para permitir una adopción más rápida de estas tecnologías en los mercados de energías renovables.

Capítulo 5

Conclusiones y recomendaciones

5.1. Conclusiones

En general, se logró desarrollar una herramienta con una interfaz gráfica intuitiva y versátil, la cual permite recopilar y procesar información generada por fuentes externas, y representar los resultados de forma visual. Su arquitectura modular permite que un usuario con conocimiento técnico pueda modificar sus funcionalidades o agregar nuevos modelos de estimación energética para comparaciones contra datos reales. Esta herramienta está ideada para la cooperación científica en posibles estudios posteriores, como por ejemplo validaciones de tipo *round-robin*. Todo esto fue realizado mediante herramientas de código abierto, lo que facilita una posterior expansión de sus funcionalidades con una inversión económica baja o nula.

A nivel específico:

- Se diseñó un método para recopilar y extraer la información de archivos en formato CSV que contienen datos sobre las simulaciones ejecutadas para los casos de *PVSyst* y *NREL* mediante el módulo *extraction.py*, además de ordenar y clasificar la información relevante según simulador y tipo de dato usando el módulo *sorting.py*, así como permitirle a un usuario ingresar y recopilar datos para el modelo empírico de *Solar World* a través del módulo *empirical.py*.
- Se desarrolló la capacidad de calcular tres variables de estimación energética usando el módulo *empirical.py*, y generar gráficas y tablas mediante los módulos *plotter.py* y *tables.py*, que permiten comparar datos de ganancia bifacial, energía bifacial y eficiencia energética de un módulo entre herramientas de simulación en un mismo plano, para un periodo de 12 meses.
- Se logró validar la herramienta desarrollada mediante la incorporación de datos reales de energía y ganancia bifacial tomados de un experimento de medición en las instalaciones del centro APOLLO de la Universidad de Ciencias Aplicadas de Anhalt. Estos datos reales permitieron generar, mediante el módulo *statistics.py*, errores de *RMSE* y *MBE* para analizar (con un método de validación cruzada) la desviación entre los datos simulados y los datos reales, determinar si estos fueron sobreestimados o subestimados, y en conjunto con otros módulos de la herramienta generar gráficas y tablas representativas de dichos errores.

5.2. Recomendaciones

- Generar un paquete con un archivo ejecutable para facilitar su exportación y utilización por usuarios en diferentes sistemas físicos (computadoras), removiendo la necesidad de tener instalado en el equipo *Python* y las bibliotecas que componen a la herramienta.
- Ya que el código de la herramienta permite expandir sus funcionalidades, sería de utilidad agregar otros tipos de gráficas (según las necesidades del usuario) y la posibilidad de generar análisis estadísticos más profundos, para determinar con certeza el modelo de simulación más exacto.
- Realizar análisis en otras ubicaciones geográficas, con más *sets* de datos reales obtenidos en otros estudios, para comparar la exactitud de la herramienta diseñada mediante diferentes casos de uso.

Apéndice A

Abreviaturas

- **APOLLO:** Anhalt Photovoltaic Performance and Lifetime Laboratory
- **BGE:** Bifacial Gain in Energy
- **CENCE:** Centro Nacional de Control de Energía
- **CSV:** Comma-separated values
- **ERNC:** Energías renovables no convencionales
- **GB:** Gigabyte
- **GCR:** Ground Coverage Ratio
- **GUI:** Graphical User Interface
- **ICE** Instituto Costarricense de Electricidad
- **IEC:** International Electrotechnical Commission
- **IRENA:** International Renewable Energy Association.
- **ITRPV:** International Technology Roadmap for Photovoltaic
- **MBE:** Mean bias error
- **NREL:** National Renewable Energy Laboratory
- **PID:** Potential induced degradation
- **RAM:** Random Access Memory
- **RMSE:** Root mean square error
- **SESLab:** Laboratorio de Sistemas Electrónicos para la Sostenibilidad
- **UML:** Unified Modeling Language

Bibliografía

- [1] IRENA, “World energy transisitions outlook: 1.5°C pathway,” 2021.
- [2] ——, “Installed capacity trends,” Available at <https://www.irena.org/solar> (2021).
- [3] ICE, “Informe anual de la operación del sistema eléctrico nacional 2021,” 2021.
- [4] C. RL, “San carlos inaugura su primer parque solar,” Available at <http://www.conelectricas.com/san-carlos-inaugura-primer-parque-solar/> (2019).
- [5] H. S. Ortiz, “Tecnología solar fotovoltaica 1,” 2020.
- [6] D. of Energy, “Solar photovoltaic cell basics,” 2022. [Online]. Available: <https://www.energy.gov/eere/solar/solar-photovoltaic-cell-basics>
- [7] P. A. Gómez, “Combinación de energía solar fotovoltaica y energía minieólica como alternativas limpias para el parcial abastecimiento de una vivienda,” 2020.
- [8] S. Enkhhardt, “Frameless glass-glass solar modules made in europe have the best co2 footprint, fraunhofer ise says,” 2021. [Online]. Available: <https://www.pv-magazine.com/2021/09/24/frameless-glass-glass-solar-modules-made-in-europe-have-the-best-co2-footprint-fraunhofer-ise-says/>
- [9] H. S. Ortíz, “Validation of energy prediction models for bifacial photovoltaics module,” Master’s thesis, Kothen, Germany, 2021.
- [10] H. S. Ortiz, “Paneles solares bifaciales,” 2020.
- [11] ITRPV, “International technology roadmap for photovoltaic (itrpv)-2021 results,” International Technology Roadmap for Photovoltaic, Tech. Rep., 2022.
- [12] A. B. R. et al, “Ipcc special report global warming of 1.5°C-summary for teachers,” Intergovernmental Panel on Climate Change - Office for Climate Education, Tech. Rep., 2018.
- [13] MINAE, “Plan nacional de energía 2015-2030,” 2015.
- [14] V. Chacón, “Ice sepulta el proyecto hidroeléctrico diquís,” Available at <https://semanariouniversidad.com/pais/ice-sepulta-el-proyecto-hidroelectrico-diquis/> (2018).
- [15] B. Wire, “Bmr energy inicia la construcción de una planta solar en costa rica,” Available at <https://www.businesswire.com/news/home/20210519005776/es/> (2021).
- [16] T. Martínez, “Entorno de simulación para el análisis de fallas en instalaciones fotovoltaicas con python,” Cartago, Costa Rica, 2021.

- [17] J. E. Castillo-Aguilella and P. S. Hauser, "Multi-variable bifacial photovoltaic module test results and best-fit annual bifacial energy yield model," *IEEE Access*, vol. 4, pp. 498–506, 2016.
- [18] M. Kutzer, A. Fülle, A. Jahnke, J. Hahn, S. Wendt, D. Neuhaus, A. Witzig, and K. Kutzer, "Ertragssteigerung durch bifaziale modultechnologie," *Proc. 31st Symp. Photovolt. Sol. Energy*, pp. 1–10, 2016.
- [19] A. Mermoud and B. Wittmer, "Bifacial shed simulation with pvsyst," *Bifacial Workshop*, pp. 25–26, 2017.
- [20] X. Sun, M. R. Khan, C. Deline, and M. A. Alam, "Optimization and performance of bifacial solar modules: A global perspective," *Applied energy*, vol. 212, pp. 1601–1610, 2018.
- [21] B. Marion, S. MacAlpine, C. Deline, A. Asgharzadeh, F. Toor, D. Riley, J. Stein, and C. Hansen, "A practical irradiance model for bifacial pv modules," *2017 IEEE 44th Photovoltaic Specialist Conference (PVSC)*, pp. 1537–1542, 2017.
- [22] I. C. et al., "International standard iec 61724: Photovoltaic system performance monitoring—guidelines for measurements, data exchange and analysis," 1998.
- [23] M. A. Abella, "Sistemas fotovoltaicos," 2005.
- [24] M. Woodhouse, R. Jones-Albertus, D. Feldman, R. Fu, K. Horowitz, D. Chung, D. Jordan, and S. Kurtz, "On the path to sunshot: The role of advancements in solar photovoltaic efficiency, reliability, and costs," 2016.
- [25] K. Pickerel, "What are bifacial solar modules?" 2018. [Online]. Available: <https://www.solarpowerworldonline.com/2018/04/what-are-bifacial-solar-modules/>
- [26] I. E. Comission, "International standard iec 61724: Photovoltaic system performance monitoring - guidelines for measurements, data exchange and analysis," IEC, Tech. Rep., 1998.
- [27] ——, "Iec 61853-1," IEC, Tech. Rep., 2011.
- [28] ——, "Standard 60904-1: Photovoltaic devices, part i: Measurement of photovoltaic current-voltage characteristics," IEC, Tech. Rep., 2006.
- [29] T. S. Liang, M. Pravettoni, C. Deline, J. S. Stein, R. Kopecsek, J. P. Singh, W. Luo, Y. Wang, A. G. Aberle, and Y. S. Khoo, "A review of crystalline silicon bifacial photovoltaic performance characterisation and simulation," *Energy & Environmental Science*, vol. 12, no. 1, pp. 116–148, 2019.
- [30] C. Monokroussos, E. Salis, D. Etienne, X. Zhang, S. Dittman, G. Friesen, K. Morita, J. Stang, T. Herbrecht, V. Fakhfouri, N. Rebeaud, D. Pavanello, and H. Müllejans, "Electrical characterization intercomparison of high-efficiency c-si modules within asian and european laboratories," *Progress in photovoltaics*, 2018.

- [31] Coursera, “What is python used for? a beginner’s guide.” [Online]. Available: <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>
- [32] M. Nostrati, “Python: An appropriate language for real world programming,” *World Applied Programming*, vol. 1, no. 2, pp. 110–117, 2011.
- [33] W. F. Holmgren, C. W. Hansen, and M. A. Mikofski, “pvlib python: a python package for modeling solar energy systems,” *Journal of Open Source Software*, vol. 3, no. 29, p. 884, 2018. [Online]. Available: <https://doi.org/10.21105/joss.00884>
- [34] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [35] U. D. of Energy, “Radiance.” [Online]. Available: <https://floyd.lbl.gov/radiance/framework.html>
- [36] UAEH, “Programación modular.” [Online]. Available: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/113_programacion_modular.html
- [37] Python.org, “tkinter — interface de python para tcl/tk.” [Online]. Available: <https://docs.python.org/es/3/library/tkinter.html>
- [38] ——, “os — interfaces misceláneas del sistema operativo.” [Online]. Available: <https://docs.python.org/es/3.10/library/os.html>
- [39] pandas.org, “pandas documentation.” [Online]. Available: <https://pandas.pydata.org/docs/>
- [40] NumPy, “Numpy documentation.” [Online]. Available: <https://numpy.org/doc/stable/>
- [41] Matplotlib, “Matplotlib documentation.” [Online]. Available: <https://matplotlib.org/stable/index.html#>
- [42] D. Farrel, “tkintertable.” [Online]. Available: <https://github.com/dmnfarrell/tkintertable>
- [43] PVSyst, “Pvsyst download.” [Online]. Available: <https://www.pvsyst.com/download-pvsyst/>
- [44] NREL, “bifacial radiance installation full (windows).” [Online]. Available: <https://www.youtube.com/watch?v=4A9GocfHKyM>
- [45] ——, “Nrel/radiance.” [Online]. Available: <https://github.com/NREL/Radiance/releases>

- [46] H. Sánchez, S. Dittmann, C. Meza, and R. Gottschalg, “The impact of real albedo values on energy estimation for bifacial modules,” *38th European Photovoltaic Solar Energy Conference and Exhibition*, 2021.
- [47] *LG NeON2 BiFacial*, LG Electronics, 2019.
- [48] *LG NeON2 BiFacial*, LG Electronics, 2018.
- [49] *LG NeON2 BiFacial*, LG Electronics, 2016.