



POLITÉCNICA

UNIVERSIDAD
POLITÉCNICA
DE MADRID

escuela técnica superior de
ingeniería
diseño
industrial

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO
INDUSTRIAL

Grado en Ingeniería Eléctrica

TRABAJO DE FIN DE GRADO

**Desarrollo de una herramienta
software para la simulación de
sistemas fotovoltaicos con R**

Autor: Francisco Delgado López

Tutor: Oscar Perpiñán Lamigueiro
Departamento de Ingeniería Eléctrica,
Electrónica, Automática y Física aplicada

Madrid, 9 de septiembre de 2024

Agradecimientos

A lo largo de estos años, muchas personas han sido parte esencial de este viaje.

A mis compañeros de lucha: Candela, Hugo, Juan, Sebas, Santi, Jorge, Víctor, y a tantos otros cuyo nombre quizás omito en este momento, pero que sin duda han dejado una huella imborrable en mi camino.

A aquellos que me han acompañado desde tiempos inmemoriales, siendo mi constante apoyo: Raquel, Samu, Adri (por partida doble), Diego y Jonhy. Gracias por estar siempre ahí.

A Eva, por tu apoyo incondicional. Desde que te conocí, mis días han sido un poco mejores. No encuentro palabras para expresar cuánto significa tenerte a mi lado

A mi familia: mi padre, mi madre y mi hermana, quienes han sido mi pilar y mi refugio. Sin vosotros, este logro no habría sido posible.

Por último, a mi tutor, Óscar, quien con su paciencia y saber hacer me ha podido guiar en este proyecto. Gracias por tu tiempo, dedicación y por todo lo que me has enseñado.

A todos vosotros, de corazón, os quiero y nada habría sido igual sin teneros en mi vida.

¡Gracias!

Resumen

El presente proyecto se enfoca en el desarrollo de un paquete de software estadístico en R, denominado solaR2, diseñado para estimar la productividad de sistemas fotovoltaicos a partir de datos de irradiación solar. solaR2 es una evolución del paquete solaR, con mejoras significativas en su modularidad y eficiencia. A diferencia de solaR, que utilizaba el paquete zoo para la gestión de series temporales, solaR2 se basa en data.table, lo que optimiza la manipulación de grandes volúmenes de datos y acelera el procesamiento. Este avance es crucial para un análisis más detallado y eficiente en el campo de la energía solar fotovoltaica.

solaR2 ofrece herramientas avanzadas para simular el rendimiento de sistemas conectados a la red y sistemas de bombeo de agua con energía solar. Incluye clases, métodos y funciones que permiten calcular la geometría solar y la radiación solar incidente, así como estimar la productividad final de estos sistemas a partir de la irradiación global horizontal diaria e intradiaria.

El diseño modular basado en clases S4 facilita el manejo de series temporales multivariantes y proporciona métodos de visualización avanzados para el análisis de rendimiento en plantas fotovoltaicas a gran escala. La implementación con data.table mejora la eficiencia en la manipulación de datos, permitiendo análisis más rápidos y precisos. Entre sus funcionalidades destacadas están el cálculo de radiación solar en diferentes planos, la estimación de rendimiento de sistemas fotovoltaicos y de bombeo, y la evaluación de sombras.

Además, solaR2 ofrece herramientas avanzadas para la visualización estadística del rendimiento, compatible con otros paquetes de R para manipulación de series temporales y análisis espacial. Esto la convierte en una herramienta útil para investigadores y profesionales en el diseño y optimización de sistemas fotovoltaicos, permitiendo un análisis detallado bajo diversas condiciones. En resumen, solaR2 representa una mejora significativa en el análisis y simulación de sistemas solares, proporcionando una herramienta flexible y reproducible para mejorar la eficiencia energética y la rentabilidad de las instalaciones solares.

Palabras clave: geometría solar, radiación solar, energía solar, fotovoltaica, métodos de visualización, series temporales, datos espacio-temporales, S4.

Abstract

This project focuses on the development of a statistical software package in R, called solaR2, designed to estimate the productivity of photovoltaic systems based on solar irradiation data. solaR2 is an evolution of the solaR package, with significant improvements in modularity and efficiency. Unlike solaR, which used the zoo package for time series management, solaR2 is based on data.table, optimizing the handling of large data volumes and speeding up processing. This advancement is crucial for more detailed and efficient analysis in the field of photovoltaic solar energy.

solaR2 offers advanced tools to simulate the performance of grid-connected systems and solar-powered water pumping systems. It includes classes, methods, and functions that allow for the calculation of solar geometry and incident solar radiation, as well as the estimation of the final productivity of these systems from daily and intraday global horizontal irradiation data.

The modular design, based on S4 classes, facilitates the handling of multivariate time series and provides advanced visualization methods for performance analysis in large-scale photovoltaic plants. The implementation with data.table improves data manipulation efficiency, enabling faster and more accurate analysis. Key features include the calculation of solar radiation on different planes, performance estimation of photovoltaic and pumping systems, and shadow evaluation.

In addition, solaR2 offers advanced tools for statistical visualization of performance, compatible with other R packages for time series manipulation and spatial analysis. This makes it a useful tool for researchers and professionals in the design and optimization of photovoltaic systems, allowing for detailed analysis under various conditions. In summary, solaR2 represents a significant improvement in the analysis and simulation of solar systems, providing a flexible and reproducible tool to enhance energy efficiency and profitability of solar installations.

Keywords: solar geometry, solar radiation, solar energy, photovoltaic, visualization methods, time series, spatiotemporal data, S4.

Índice general

Índice general	ix
Índice de figuras	xI
Índice de tablas	xv
Nomenclatura	xvII
1 Introducción	1
1.1. Objetivos	1
1.2. Análisis previo de soluciones	3
2 Estado del arte	5
2.1. Situación actual de la generación fotovoltaica	5
2.2. Soluciones actuales y sus carencias	6
3 Marco teórico	11
3.1. Radiación solar	12
3.2. Radiación en superficies inclinadas	15
3.3. Cálculo de la energía producida por un generador fotovoltaico	19
3.4. Sombras y ocupación de terreno	29
4 Desarrollo del código	35
4.1. Geometría solar	36
4.2. Datos meteorológicos	41
4.3. Radiación en el plano horizontal	47
4.4. Radiación efectiva en el plano del generador	55
4.5. Producción eléctrica de un SFCR	66
4.6. Producción eléctrica de un SFB	70
4.7. Optimización de distancias	73
4.8. Aspectos técnicos de la elaboración de un paquete en R	76
5 Ejemplo práctico de aplicación	81
5.1. solaR	81
5.2. PVsyst	85
5.3. solaR2	85
5.4. Comparación y conclusiones	87
6 Conclusiones	89
6.1. Aportaciones	89
6.2. Comparación de resultados	91
6.3. Desarrollo futuro	92

Bibliografía	95
A Manual de referencia de solaR2	99
B Código completo	187

Índice de figuras

3.1. El procedimiento de cálculo consiste en obtener la irradiancia efectiva a partir de la irradiación global en un plano horizontal. Primero, se separan las componentes directa y difusa utilizando índices de claridad y fracciones difusas. Luego, se trasladan estos valores al plano inclinado y se ajustan por factores de suciedad y sombras. Con la irradiancia efectiva y la eficiencia del sistema fotovoltaico, se calcula la potencia en corriente continua, que luego se convierte en corriente alterna a través de un inversor. Finalmente, al integrar esta potencia, se obtiene la energía. Figura modificada de la figura 3.3 del libro ESF [Per23].	11
3.2. Perfil de irradiancia difusa y global obtenido a partir del generador empírico de [CR79] para valores de irradiancia tomadas cada 10 minutos. Figura 3.4 del libro ESF [Per23].	16
3.3. Ángulo de visión del cielo. Figura 3.5 del libro ESF [Per23].	17
3.4. Pérdidas angulares de un módulo fotovoltaico para diferentes grados de suciedad en función del ángulo de incidencia. Figura 3.7 del libro ESF [Per23].	18
3.5. Curvas corriente-tensión(línea discontinua) y potencia-tensión(línea continua) de una célula solar ($T_a = 20^\circ C$ y $G = 800W/m^2$). Figura 4.6 del libro ESF [Per23].	20
3.6. Evolución de la eficiencia de células según la tecnología (según el National Renewable Energy Laboratory [Nat24] (EEUU)).	21
3.7. Dimensiones y distancias entre filas de un sistema estático. Figura 6.10 del libro ESF [Per23].	29
3.8. Sombras mutuas en un conjunto de cuatro seguidores. Figura 6.11 del libro ESF [Per23].	30
3.9. Dimensiones de un seguidor a doble eje y longitud de su sombra arrojada. Figura 6.12 del libro ESF [Per23].	31
3.10. Posibles sombras en un conjunto de seis seguidores. Figura 6.13 del libro ESF [Per23].	31
3.11. Ábaco para planta de seguimiento a doble eje. Recoge el ratio entre la energía anual producida por un seguidor afectado por sombras mutuas (E_{acS}) y la producida por un seguidor sin sombreado (E_{ac0}). Las curvas de color negro representan la fracción de energía no afectada por sombras. Las curvas de puntos representan el valor del ROT. Figura 6.14 del libro ESF [Per23].	33
3.12. Dimensiones básicas en sistemas con seguidores de eje horizontal. Figura 6.16 del libro ESF [Per23].	33
4.1. Proceso de cálculo de las funciones de solaR2	35
4.2. Cálculo de la geometría solar mediante la función calcSol , la cual unifica las funciones fSolD y fSolI resultando en un objeto clase Sol el cual contiene toda la información geométrica necesaria para realizar las siguientes estimaciones.	36
4.3. Comparación entre los diferentes métodos de cálculo de fSolD	37
4.4. Representación gráfica de los resultados de la función fSolI	38
4.5. Representación gráfica de los resultados de la función fSolI una vez eliminados los valores nocturnos utilizando el argumento keep.night	39
4.6. Representación gráfica de los resultados de la función fSolI sin realizar la corrección por la ecuación del tiempo.	40

4.7. Los datos meteorológicas se pueden leer mediante las funciones <code>readG0dm</code> , <code>readBD</code> , <code>dt2Meteo</code> , <code>zoo2Meteo</code> y <code>readSIAR</code> las cuales procesan estos datos y los almacenan en un objeto de clase <code>Meteo</code>	41
4.8. Representación gráfica de un objeto <code>Meteo</code> producido con medias mensuales de datos diarios.	42
4.9. Representación gráfica de un objeto <code>Meteo</code> producido con datos diarios procedentes de un archivo.	43
4.10. Representación gráfica de un objeto <code>Meteo</code> producido con datos intradiarios procedentes de un archivo.	44
4.11. Representación gráfica de un objeto <code>Meteo</code> producido con datos almacenados en <code>data.table</code>	45
4.12. Representación gráfica de un objeto <code>Meteo</code> producido por la función <code>readSIAR</code>	47
4.13. Cálculo de la radiación incidente en el plano horizontal mediante la función <code>calcG0</code> , la cual procesa un objeto clase <code>Sol</code> y otro clase <code>Meteo</code> mediante las funciones <code>fCompD</code> y <code>fCompI</code> resultando en un objeto clase <code>G0</code>	47
4.14. Comparación gráfica entre los resultados de la función <code>fCompD</code> cambiando entre tipos de correlación mensual.	48
4.15. Comparación gráfica entre los resultados de la función <code>fCompD</code> cambiando entre tipos de correlación mensual.	49
4.16. Representación gráfica de los resultados de la función <code>fCompD</code> tomando como función de correlación una propuesta por el usuario.	50
4.17. Representación gráfica de los resultados de la función <code>fCompI</code>	51
4.18. Comparación entre los resultados de la función <code>fCompI</code> cambiando entre diferentes tipos de correlación intradiaria.	52
4.19. Representación gráfica de un objeto <code>G0</code> resultado de la función <code>calcG0</code>	55
4.20. Cálculo de la radiación efectiva incidente en el plano del generador mediante la función <code>calcGef</code> , la cual emplea la función <code>fInclin</code> para el computo de las componentes efectivas, la función <code>fTheta</code> que provee a la función anterior los ángulos necesarios para su computo y la función <code>calcShd</code> que reprocesa el objeto de clase <code>Gef</code> resultante, añadiendole el efecto de las sombras producidas entre módulos.	56
4.21. Representación gráfica del resultado de la función <code>fTheta</code> para un sistema fijo.	57
4.22. Representación gráfica del resultado de la función <code>fTheta</code> para un sistema de seguimiento de dos ejes.	57
4.23. Representación gráfica del resultado de la función <code>fTheta</code> para un sistema de seguimiento horizontal.	58
4.24. Representación gráfica del resultado de la función <code>fTheta</code> para un sistema de seguimiento horizontal con backtracking.	58
4.25. Comparación entre resultados de la función <code>fInclin</code> tomando diferentes niveles de suciedad.	59
4.26. Comparación entre resultados de la función <code>fInclin</code> tomando diferentes valores del coeficiente de reflexión del terreno.	60
4.27. Representación gráfica de un objeto <code>Gef</code> resultado de la función <code>calcGef</code>	66
4.28. Estimación de la producción eléctrica de un SFCR mediante la función <code>prodGCPV</code> , la cual emplea la función <code>fProd</code> para el computo de la potencia a la entrada (P_{DC}), a la salida (P_{AC}) y el rendimiento (η_{inv}) del inversor.	66
4.29. Representación gráfica de un objeto <code>ProdGCPV</code> resultado de la función <code>prodGCPV</code>	70
4.30. Estimación de la producción eléctrica de un SFB mediante la función <code>prodPVPS</code> , la cual emplea la función <code>fPump</code> para el computo del rendimiento de las diferentes partes de una bomba centrífuga alimentada por un convertidor de frecuencia.	70
4.31. Representación gráfica de un objeto clase <code>ProdPVPS</code> producto de la función <code>prodPVPS</code>	73

4.32. Ábaco para planta de seguimiento a doble eje producto de la función <code>optimShd</code> , e interpretado gráficamente por la función <code>shadeplot</code>	74
4.33. Ábaco para planta fija producto de la función <code>optimShd</code> , e interpretado gráficamente por la función <code>shadeplot</code>	76
5.1. Representación gráfica de un objeto <code>Meteo</code> formado con la información meteorológica de la azotea de la ETSIDI entre los años 2013 y 2015 (información horaria).	82
5.2. Comparación gráfica entre las productividades de los sistemas.	85
5.3. Representación gráfica de un objeto <code>Meteo</code> formado con la información meteorológica de la azotea de la ETSIDI entre los años 2013 y 2015 (horaria).	86
5.4. Comparación gráfica entre las productividades de los sistemas.	87
6.1. Pestaña <i>blame</i> de GitHub de la función <code>fCompD</code>	90
6.2. Sección <i>contributors</i> de GitHub para los últimos 6 meses.	90
6.3. Resultado de rendimiento de la función <code>prodGCPV</code> obtenidos con la función <code>profvis</code> . Arriba, la pestaña <i>Data</i> ; y abajo, la pestaña <i>Flame Graph</i>	92

Índice de tablas

3.1. Valor d_n correspondiente a los doce días promedio.	14
3.2. Valores del coeficiente de pérdidas angulares y transmitancia relativa en incidencia normal para diferentes tipos de suciedad.	19
5.1. Parámetros técnicos de diferentes tipos de células solares.	82
5.2. Sistemas fotovoltaicos.	83
5.3. Características del inversor.	84
5.4. Energía media mensual estimada por PVSyst en <i>KWh</i>	85

Nomenclatura

A_c	Área de una célula
A_G	Área de un generador fotovoltaico
α	Ángulo de orientación de un sistema fotovoltaico
AM	Masa de aire
AO	Adelanto oficial durante el horario de invierno
B	Radiación directa
$B_0(0)$	Irradiancia extra-atmosférica o extra-terrestre en el plano horizontal
$B_{0d}(0)$	Irradiación diaria extra-atmosférica en el plano horizontal
β	Ángulo de inclinación de un sistema fotovoltaico
D	Radiación difusa
D^C	Radiación difusa circunsolar
δ	Declinación
$\Delta\lambda$	Diferencia entre la longitud local y la longitud del huso horario
D^I	Radiación difusa isotrópica
d_{min}	Distancia mínima entre hileras de un generador para evitar el sombreado
d_n	Día del año
EoT	Ecuación del tiempo
ϵ_0	Corrección debida a la excentricidad de la elipse de la trayectoria terrestre alrededor del Sol
η_{mp}	Eficiencia de una motobomba
F_D	Fracción de difusa
F_{Dd}	Fracción de difusa diaria
F_{Dm}	Fracción de difusa mensual
FT_B	Factor de pérdidas angulares para irradiancia directa
FT_R	Factor de pérdidas angulares para irradiancia de albedo

FT_D	Factor de pérdidas angulares para irradiancia difusa
g	Aceleración de la gravedad
G	Radiación global
GCR	Ground coverage ratio
G_{STC}	Irradiancia incidente en condiciones estandar de medida
H_{dt}	Nivel dinámico de un pozo
H_f	Altura asociada a las pérdidas de fricción en una tubería
H_{OT}	Diferencia de cotas entre la salida de agua y la entrada en el depósito
H_{st}	Nivel estático de un pozo
H_T	Altura total incluyendo las pérdidas de fricción de la tubería
H_{TE}	Altura total equivalente de un sistema de bombeo
H_v	Altura vertical aparente
I_{mpp}	Corriente de una célula en el punto de máxima potencia
I_{sc}	Corriente de cortocircuito de una célula
K_T	Índice de claridad
K_{Td}	Índice de claridad diario
K_{Tm}	Índice de claridad mensual
L_{eo}	Separación entre segidores en sentido Este-Oeste
L_{ns}	Separación entre segidores en sentido Norte-Sur
MPP	Punto de máxima potencia de un dispositivo fotovoltaico
N_p	Número de ramas en paralelo en un generador
N_s	Número de módulos en serie de un generador
ω	Hora solar o tiempo solar verdadero
ω_s	Ángulo del amanecer
P_{AC}	Potencia alterna a la salida de un inversor
P_{DC}	Potencia continua a la salida de un generador fotovoltaico
P_{el}	Potencia eléctrica necesaria en la entrada de una motobomba
P_f	Pérdidas de fricción en la tubería de un sistema de bombeo
P_g^*	Potencia del generador en condiciones estándar
P_H	Potencia hidráulica necesaria en un sistema de bombeo de agua
P_H	Potencia hidráulica

ϕ	Latitud
p_i	Potencia continua a la entrada de un inversor normalizada por la potencia nominal del equipo
P_{inv}	Potencia nominal de un inversor
p_o	Potencia alterna a la salida de un inversor normalizada por la potencia nominal del equipo
ψ_s	Ángulo acimutal solar
Q	Caudal de agua
Q_{max}	Caudal máximo del pozo
Q_t	Caudal de ensayo de un pozo
R	Radiación del albedo
r_D	Relación entre la irradiancia y la radiación difusa en el plano horizontal
ρ	Densidad del agua
ρ	Coeficiente de reflexión del terreno para la irradiancia de albedo
ROT	Ratio de ocupación del terreno
STC	Condiciones estándar de medida de un dispositivo fotovoltaico
Ta	Temperatura ambiente
T_c^*	Temperatura de célula en condiciones estándar de medida
T_c	Temperatura de célula
θ_s	Ángulo de incidencia o ángulo entre el vector solar y el vector director de una superficie
θ_{zs}	Ángulo cenital solar
T_{max}	Temperatura máxima en un día
T_{min}	Temperatura mínima en un día
TO	Hora oficial
$TONC$	Temperatura de operación nominal de célula
V_{mpp}	Tensión de una célula en el punto de máxima potencia
V_{oc}	Tensión de circuito abierto de una célula
Y_f	Productividad de un sistema fotovoltaico

Introducción

1.1. Objetivos

El objetivo principal de este proyecto es el desarrollo de un paquete en R [R C23] con el cual poder realizar estimaciones y representaciones gráficas de la geometría solar, radiación solar en el plano horizontal y del generador, y el funcionamiento de sistemas fotovoltaicos de conexión a red y de bombeo de agua.

Durante el resto del documento, cuando sea preciso, se hará referencia al paquete desarrollado en este proyecto con el nombre `solaR2`.

El usuario puede colocar los datos que considere convenientes (desde una base de datos oficial hasta una base de datos propia) en cada una de las funciones que ofrece el paquete pudiendo así obtener resultados de la geometría solar, de la radiación horizontal, de la efectiva y hasta de la producción de diferentes tipos de sistemas fotovoltaicos.

El paquete también incluye una serie de funciones que permiten hacer representaciones gráficas de estos resultados con el fin de poder apreciar con más detalle las diferencias entre sistemas y contemplar cual es la mejor opción para el emplazamiento elegido.

Este proyecto, toma su origen en el paquete ya existente `solaR` [Per12], el cual, desarrolló el tutor de este proyecto en 2010. Esta versión, la 0.14, ha tenido una serie de actualizaciones, siendo la más reciente la 0.46 (en el 2021). Sin embargo, al ser versiones de un software antiguo se propuso la idea de renovarlo teniendo en cuenta el paquete en el que basa su funcionamiento. El paquete `solaR` ha basado su funcionamiento en el paquete `zoo` [ZG05] el cual proporciona una sólida base para trabajar con series temporales. Sin embargo, como base de `solaR2` se ha optado por el paquete `data.table` [Bar+24]. Este paquete ofrece una extensión de los clásicos `data.frame` de R en los `data.table`, los cuales pueden trabajar rápidamente con enormes cantidades de datos (por ejemplo, 100 GB de RAM).

La clave de ambos proyectos es que, al estar basados en R, cualquier usuario puede acceder a ellos de forma gratuita; tan solo se necesita tener instalado R en tu dispositivo.

Para alojar este proyecto se toman dos vías:

- **Github** [Wan+23]: donde se aloja la versión de desarrollo del paquete.

1. INTRODUCCIÓN

- CRAN [R C24]: acrónimo de *Comprehensive R Archive Network*, es el repositorio donde se alojan las versiones definitivas de los paquetes y desde el cual se descargan a la sesión de R.

El paquete **solaR2** permite realizar las siguientes operaciones:

- Cálculo de toda la geometría que caracteriza a la radiación procedente del Sol.
- Tratamiento de datos meteorológicos (en especial de radiación), procedentes de datos ofrecidos del usuario y de la red de estaciones *SIAR* [Min23].
- Una vez calculado lo anterior, se pueden hacer estimaciones de:
 - Los componentes de radiación horizontal.
 - Los componentes de radiación eficaz en el plano inclinado.
 - La producción de sistemas fotovoltaicos conectados a red y sistemas fotovoltaivos de bombeo.

Este proyecto ha tenido a su vez una serie de objetivos secundarios:

- Uso y manejo de *GNU Emacs* [Sta85] en el que se realizaron todos los archivos que componen este documento (utilizando el modo Org [Dom+03]) y el paquete descrito (empleando ESS [Pro24]).
- Dominio de diferentes paquetes de R:
 - **zoo** [ZG05]: paquete que proporciona un conjunto de clases y métodos en S3 para trabajar con series temporales regulares e irregulares. Usado en el paquete **solaR** como pilar central.
 - **data.table** [Bar+24]: otorga una extensión a los datos de tipo data.frame que permite una alta eficiencia especialmente con conjuntos de datos muy grandes. Se ha utilizado en el paquete **solaR2** en sustitución del paquete **zoo** como tipo de dato principal en el cual se construyen las clases y métodos de este paquete.
 - **microbenchmark** [Mer+23]: proporciona infraestructura para medir y comparar con precisión el tiempo de ejecución de expresiones en R. Usado para comparar los tiempos de ejecución de ambos paquetes.
 - **profvis** [Wic+24]: crea una interfaz gráfica donde explorar los datos de rendimiento de una expresión dada. Aplicada junto con **microbenchmark** para detectar y corregir cuellos de botella en el paquete **solaR2**
 - **lattice** [Sar08]: proporciona diversas funciones con las que representar datos. **solaR2** utiliza este paquete para representar de forma visual los datos obtenidos en las estimaciones.
- Junto con el modo Org, se ha utilizado el prepador de textos L^AT_EX (partiendo de un archivo .org, se puede exportar a un archivo .tex para posteriormente exportar un .pdf).
- Obtener conocimientos teóricos acerca de la radiación solar y de la producción de energía solar mediante sistemas fotovoltaicos y sus diversos tipos. Para ello, se ha usado en mayor medida el libro *Energía Solar Fotovoltaica* [Per23].

1.2. Análisis previo de soluciones

Antes de comenzar el desarrollo del proyecto, se llevó a cabo una revisión de las soluciones de estimaciones de instalaciones fotovoltaicas existentes en el mercado. Algunas de las soluciones encontradas fueron:

1. **PVSyst- Photovoltaic Software:** el software PVSyst, desarrollado por la empresa suiza con el mismo nombre, es quizá el más conocido dentro del ámbito del estudio y la estimación de instalaciones fotovoltaicas. Ofrece una amplia capacidad de personalización de todos los componentes de la instalación.
2. **SISIFO:** es una herramienta web diseñada y desarrollada por el Grupo de Sistemas Fotovoltaicos del Instituto de Energía Solar de la Universidad Politécnica de Madrid. Ha sido y es la herramienta interna utilizada por los ingenieros de dicho grupo.
3. **PVGIS:** aplicación web desarrollada por el European Commission Joint Research Center desde. Su enfoque es asistir en el cálculo y estimación de instalaciones fotovoltaicas, ya sean conectadas a red, de seguimiento o de autoconsumo.
4. **System Advisor Model:** system Advisor Model (SAM), desarrollado por el Laboratorio Nacional de Energías Renovables, perteneciente al Departamento de Energía del gobierno americano, es un software técnico-económico gratuito que ayuda a la toma de decisiones en el amplio campo de las energías renovables. Ofrece un conjunto de soluciones muy completas no solamente relacionadas con la energía fotovoltaica, sino también termosolar, eólica, geotermal o biomasa, entre otras.
5. **solaR:** solaR es un paquete de código para el entorno de R, desarrollado por Oscar Perpiñán. Es el antecesor del paquete del que trata este proyecto.

En el apartado 2.2 se lleva a cabo un desarrollo más detallado de las características de las soluciones mencionadas, así como sus ventajas y limitaciones.

CAPÍTULO 2

Estado del arte

2.1. Situación actual de la generación fotovoltaica

Según el informe anual de 2023 de la UNEF¹ [UNE23], en 2022 la fotovoltaica se posicionó como la tecnología con más crecimiento a nivel internacional, tanto entre las renovables como entre las no renovables. Se instalaron 240 GWp de nueva capacidad fotovoltaica a nivel mundial, suponiendo esto un incremento del 137 % con respecto a 2021.

A pesar de las diversas crisis internacionales, la energía solar fotovoltaica superó los 1185 GWp instalados. Como otros años, las cifras indican que China continuó siendo el primer actor mundial, superando los 106 GWp de potencia instalada en el año. La Unión Europea se situó en el segundo puesto, duplicando la potencia instalada en 2021 y alcanzando un nuevo récord con 41 GWp instalados en 2022.

La producción energía fotovoltaica a nivel mundial representó el 31 % de la capacidad de generación renovable, convirtiéndose así en la segunda fuente de generación, solo por detrás de la energía hidráulica. En 2022 se añadió 3 veces más de energía solar que de energía eólica en todo el mundo.

Por otro lado, la Unión Europea superó a EE.UU. como el segundo mayor actor mundial en desarrollo fotovoltaico, instalando un 47 % más que en 2021 y alcanzando una potencia acumulada de más de 208 GWp. España lideró el mercado europeo con 8,6 GWp instalados en 2022, superando a Alemania.

El año 2022 fue significativo en términos legislativos con el lanzamiento del Plan REPowerEU² [Eur22]. Dentro de este plan, se lanzó la Estrategia de Energía Solar con el objetivo de alcanzar 400 GWp (320 GW) para 2030, incluyendo medidas para desarrollar tejados solares, impulsar la industria fotovoltaica y apoyar la formación de profesionales en el sector.

En 2022, España vivió un auge en el desarrollo fotovoltaico, instalando 5.641 MWp en plantas en suelo, un 30 % más que en 2021, y aumentando el autoconsumo en un 108 %, alcanzando 3.008 MWp. El sector industrial de autoconsumo creció notablemente, representando el 47 % del autoconsumo total.

¹UNEF: Unión Española Fotovoltaica.

²Plan REPowerEU: Proyecto por el cual la Unión Europea quiere poner fin a su dependencia de los combustibles fósiles rusos ahorrando energía, diversificando los suministros y acelerando la transición hacia una energía limpia.

España implementó varias iniciativas legislativas para enfrentar la volatilidad de precios de la energía y la dependencia del gas, destacando el RD-ley 6/2022 [BOE22b] y el RD 10/2022 [BOE22a], que han modificado mecanismos de precios y estableciendo límites al precio del gas.

El Plan SE+³ [dem22] incluye medidas fiscales y administrativas para apoyar las renovables y el autoconsumo. En 2022, se realizaron subastas de energía renovable, asignando 140 MW a solar fotovoltaica en la tercera subasta y 1.800MW en la cuarta, aunque esta última quedó desierta por precios de reserva bajos.

Se adjudicaron 1.200 MW del nudo de transición justa de Andorra a Enel Green Power España, con planes para instalar plantas de hidrógeno verde y agrovoltáica. La actividad en hidrógeno verde y almacenamiento también creció, con fondos adicionales y exenciones de cargos.

El autoconsumo, apoyado por diversas regulaciones y altos precios de la electricidad, registró un crecimiento significativo, alcanzado 2.504 MW de nueva potencia en 2022. Las comunidades energéticas también avanzaron gracias a ayudas específicas, a pesar de la falta de un marco regulatorio definido.

2022 estuvo marcado por los programas financiados por la Unión Europea, especialmente el Mecanismo de Recuperación y Resiliencia [Hac22] que canaliza los fondos NextGenerationEU [Uni20]. El PERTE⁴, aprobado en diciembre de 2021, espera crear más de 280.000 empleos, con ayudas que se ejecutarán hasta 2026. En 2023 se solicitó a Bruselas una adenda para segunda fase del PERTE, obteniendo 2.700 millones de euros adicionales.

La contribución del sector fotovoltaico a la economía española en 2022 fue significativa, aportando 7.014 millones de euros al PIB⁵, un 51 % más que el año anterior, y generando una huella económica total de 15.656 millones de euros. En términos de empleo, el sector involucró a 197.383 trabajadores, de los cuales 40.683 fueron directos, 97.600 indirectos y 59.100 inducidos.

El sector industrial fotovoltaico nacional tiene una fuerte presencia en España, con hasta un 65 % de los componentes manufacturados localmente. Empresas españolas se encuentran entre los principales fabricantes mundiales de inversores y seguidores solares. Además, España es un importante exportador de estructuras fotovoltaicas y cuenta con iniciativas prometedoras para la fabricación de módulos solares.

En definitiva, la fotovoltaica es una tecnología en auge y con perspectivas para ser el pilar de la transición ecológica. Por ello, surge la necesidad de encontrar herramientas que permitan estimar el desempeño que estos sistemas pueden tener a la hora de realizar estudios de viabilidad económica.

2.2. Soluciones actuales y sus carencias

Como se mencionó en el capítulo 1, existen una serie de herramientas que permiten el cálculo y la simulación de instalaciones fotovoltaicas. Todas ellas presentan una serie de ventajas específicas, en contraste de una serie de limitaciones. Estas son:

1. PVsyst - Photovoltaic Software [PVS24].

³Plan + Seguridad Energética: Se trata de un plan con medidas de rápido impacto dirigidas al invierno 2022/2023, junto con medidas que contribuyen a un refuerzo estructural de esa seguridad energética.

⁴PERTE: Proyecto Estratégico para la Recuperación y Transformación Económica.

⁵PIB: Producto Interior Bruto.

Este software se encuentra entre los más conocidos dentro del ámbito del estudio y la estimación de instalaciones fotovoltaicas. Destaca por la personalización detallada de los componentes de la instalación (módulos, inversores, sombreado, etc.), lo que permite una simulación precisa a través de datos meteorológicos y parámetros detallados del sistema. Su uso está extendido en proyectos de gran escala y estudios avanzados de eficiencia.

- Ventajas:

- **Completo y profesional:** PVsyst es altamente detallado, permitiendo análisis avanzados para proyectos pequeños y grandes.
- **Base de datos meteorológicos:** integra datos climáticos de fuentes como Meteonorm [JG20], lo que mejora la precisión de las simulaciones.
- **Simulaciones avanzadas:** permite modelar la energía producida por una planta fotovoltaica y calcular las pérdidas debidas a sombreado, inclinación, orientaciones y resistencias internas de los módulos.
- **Herramientas de dimensionamiento:** ofrece módulos específicos para diseñar la configuración de inversores y módulos solares.

- Limitaciones:

- **Costo:** es un software comercial, con licencias que pueden ser costosas para proyectos pequeños.
- **Curva de aprendizaje:** su interfaz puede resultar compleja para usuarios nuevos, lo que implica una curva de aprendizaje considerable.
- **Enfoque técnico:** está más orientado a ingenieros y técnicos, por lo que carece de accesibilidad para usuarios no especializados.

2. SISIFO [Sis24].

Herramienta web diseñada por el **Grupo de Sistemas Fotovoltaicos del Instituto de Energía Solar de la Universidad Politécnica de Madrid**. Está diseñada para ser accesible y fácil de usar, enfocándose en una audiencia más general, incluyendo ingenieros, pero también técnicos y académicos.

- Ventajas:

- **Facilidad de uso:** tiene una interfaz amigable y fácil de utilizar, lo que lo hace accesible para usuarios con distintos niveles de experiencia.
- **Open-source:** al ser de código abierto, permite a los desarrolladores modificar y adaptar el software a sus necesidades específicas.
- **Simulación integrada:** ofrece la posibilidad de realizar simulaciones basadas en datos meteorológicos, aunque con un nivel de detalle inferior a PVsyst.
- **Soporte comunitario:** al ser de código abierto, cuenta con una comunidad activa de usuarios y desarrolladores que colaboran en mejoras y actualizaciones.

- Limitaciones:

- **Escasa precisión:** al compararse con otras herramientas, su precisión puede ser menor en cuanto a modelado y simulación de pérdidas, ya que simplifica varios aspectos del sistema.
- **Poca funcionalidad en grandes proyectos:** no se adapta a las grandes instalaciones o análisis financieros avanzados con la misma eficacia que en los proyectos más reducidos.

3. PVGIS [PVG24].

Aplicación web desarrollada por el **European Commission Joint Research Center** desde 2001. Está diseñada para proporcionar estimaciones de producción de energía solar en función de la ubicación geográfica y condiciones meteorológicas históricas.

- Ventajas:

- **Gratis y accesible:** esta herramienta es completamente gratuita y accesible a través de una interfaz web, lo que facilita el uso por parte de cualquier persona.
- **Datos meteorológicos precisos:** proporciona acceso a datos meteorológicos satelitales y de estaciones terrestres, lo que permite obtener estimaciones razonables de producción de energía.
- **Estudios rápidos:** es ideal para obtener estimaciones preliminares y estudios de viabilidad de sistemas fotovoltaicos.

- Limitaciones:

- **Falta de personalización:** en comparación con otros programas más avanzados, PVGIS no permite personalizar detalles técnicos de la instalación (por ejemplo, inversores específicos o modelos de paneles) lo que puede reducir la precisión en estudios detallados.
- **Limitación en análisis de pérdidas:** no ofrece herramientas avanzadas para modelar pérdidas complejas como sombreado detallado, resistencias internas o interacciones entre componentes específicos del sistema.
- **Enfoque limitado:** está diseñado principalmente para estimaciones rápidas, por lo que no es adecuado para proyectos a gran escala o análisis financieros detallados.

4. System Advisor Model [SAM24].

Desarrollado por el **Laboratorio Nacional de Energías Renovables**, perteneciente al Departamento de energía del Gobierno de EE.UU. Está orientada a la modelación tanto técnica como económica de sistemas de energía renovable, incluyendo fotovoltaicos.

- Ventajas:

- **Modelo económico avanzado:** integra análisis detallados sobre la viabilidad económica, lo que permite evaluar tanto la producción energética como los costos y beneficios a lo largo de la vida útil del proyecto.
- **Acceso a múltiples tecnologías:** además de fotovoltaicos, permite modelar otras tecnologías de energía renovable, lo que lo hace más flexible para estudios multidisciplinares.
- **Integración de bases de datos:** utiliza datos meteorológicos detallados, lo que mejora la precisión de las simulaciones.

- Limitaciones:

- **Complejidad:** aunque gratuito, SAM es bastante complejo y técnico, esto puede hacer que solo los usuarios con experiencia en el modelado de sistemas energéticos puedan utilizarlo.
- **Interfaz poco intuitiva:** comparado con otras herramientas, requiere un mayor tiempo de familiarización debido a su enfoque integral y detalle en las simulaciones.

Como se mencionó en el capítulo 1 este proyecto toma su base en el paquete **solaR** [Per12], el cual es una herramienta robusta para el cálculo de la radiación solar y el rendimiento de sistemas fotovoltaicos.

Este paquete está diseñado utilizando clases **S4** en **R**, y su núcleo se basa en series temporales multivariantes almacenadas en objetos de la clase **zoo**. Su funcionamiento se basa, al igual que **solar2**, en una serie de funciones constructoras que calculan objetos relacionados con cada paso de la simulación de un sistema fotovoltaico. Podemos dividir su funcionamiento en los siguientes grupos:

1. **Cálculo de la geometría solar:** calcula el movimiento aparente diario (con **fSolD**) e intradiario (con **fSolI**) del Sol desde la Tierra. Para ello se vale de la función **calcSol** la cual devuelve un objeto de clase **Sol** que contiene todos los ángulos necesarios.
2. **Almacenamiento de datos meteorológicos:** se define la clase **Meteo**, la cual, se construye mediante una serie de funciones (**readBD**, **readG0dm**, **zoo2Meteo**, **df2Meteo** ...). Estas funciones toman los datos meteorológicos provenientes de distintas vías (un **data.frame**, un objeto **zoo**, un fichero...) y los adapta para que puedan ser manipulados por el resto de funciones del paquete.
3. **Cálculo de radiación en un plano horizontal:** tomando los objetos anteriores, es capaz de calcular (si no vienen ya dadas) las componentes de la irradiación (con **fCompD**) y de la irradiancia (con **fCompI**). La función **calcG0** devuelve un objeto **G0** que contiene las anteriores componentes y añade medias mensuales de valores diarios y sumas anuales de la irradiación.
4. **Cálculo de radiación en el plano del generador:** toma un objeto **G0** y lo transforma en un objeto **Gef** mediante la función **calcGef**, la cual utilizando las funciones **fTheta** y **fInclin** determinan la irradiación y la radiación efectiva al igual que las medias mensuales de la irradiación diaria y sumas anuales.
5. **Simulación de sistemas fotovoltaicos conectados a red:** con un objeto **Gef** y con los parámetros del sistema, la función **prodGCPV**, tomando los resultados de la función **fProd**, calcula la producción energética de un SFCR. Devuelve un objeto de clase **ProdGCPV** que incluye valores de potencias instantaneas y energías diarias, medias mensuales y sumas anuales.
6. **Simulación de sistemas fotovoltaicos de bombeo:** toma un objeto **Gef** y con los parámetros del sistema y de la bomba, la función **prodPVPS**, tomando los resultados de la función **fPump**, calcula la producción energética de un SFB.
7. **Optimización de distancias:** es capaz de optimizar las distancias de un sfcr mediante la función **optimShd**, la cual devuelve un objeto **Shade** el cual contiene múltiples combinaciones de distancias para que el usuario pueda decidir la mejor.
8. **Métodos de visualización:** para cada uno de los objetos mencionados existen métodos de visualización gráfica para ayudar a comprender los resultados obtenidos.

Pese a ser un herramienta muy capaz, **solar** presenta una serie de carencias relativas:

- **Modularidad:** el paquete **solar** contiene funciones que realizan muchas operaciones, esto deja poco lugar al usuario para que pueda entender cada componente independientemente.
- **Eficiencia y rendimiento:** el paquete **solar** utiliza **zoo** para manejar series temporales, lo cual es adecuado para volúmenes de datos moderados. Sin embargo, **zoo** no está optimizado para operaciones de alta eficiencia en datasets grandes.

- **Escalabilidad:** `solaR` puede experimentar problemas de escalabilidad al trabajar con datasets extensos, ya que `zoo` no es tan eficiente en operaciones que requieren manipulación compleja o parallelización.
- **Manipulación de datos:** `zoo` es adecuado para manejar series temporales básicas, pero carece de las capacidades avanzadas de manipulación de datos que ofrecen otros paquetes.

En el capítulo 5 se realizará un ejemplo práctico que compare los resultados entre `PVsyst`, `solaR` y `solaR2`.

3

CAPÍTULO

Marco teórico

El paquete **solaR2** toma como marco teórico el libro de Oscar Perpiñán, tutor de este trabajo, Energía Solar Fotovoltaica [Per23] para cada una de las operaciones de cálculo que realizan cada una de las funciones. En la figura 3.1, se muestra un diagrama que resume los pasos que se siguen a la hora de calcular la producción de sistemas fotovoltaicos.

Estos pasos son:

1. Calcular la geometría que define la posición relativa del Sol desde la Tierra.

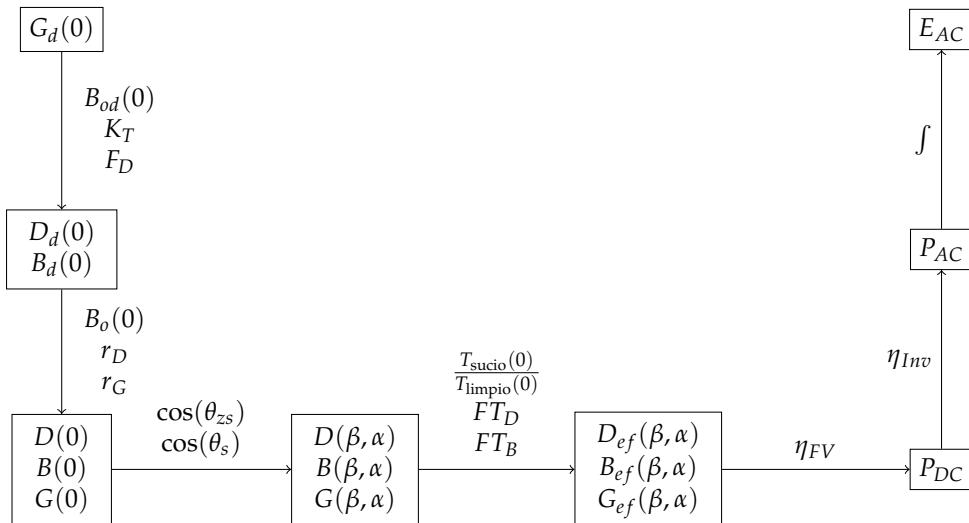


FIGURA 3.1: *El procedimiento de cálculo consiste en obtener la irradiancia efectiva a partir de la irradiación global en un plano horizontal. Primero, se separan las componentes directa y difusa utilizando índices de claridad y fracciones difusas. Luego, se trasladan estos valores al plano inclinado y se ajustan por factores de suciedad y sombras. Con la irradiancia efectiva y la eficiencia del sistema fotovoltaico, se calcula la potencia en corriente continua, que luego se convierte en corriente alterna a través de un inversor. Finalmente, al integrar esta potencia, se obtiene la energía. Figura modificada de la figura 3.3 del libro ESF [Per23].*

2. Obtener la irradiación global diaria en el plano horizontal.
3. A partir de la irradiación global, obtener las componentes de difusa y directa.
4. Trasladar estos valores de irradiación a valores de irradiancia.
5. Integrando estos valores, obtener las estimaciones irradiación diaria difusa, directa y global.
6. El generador fotovoltaico produce una potencia en corriente continua dependiente del rendimiento del mismo.
7. Un inversor, el cual tiene una eficiencia asociada, convierte esta potencia en corriente alterna.
8. Integrar esta potencia para obtener la energía que produce el generador en un tiempo determinado.

3.1. Radiación solar

3.1.1. Geometría Sol y Tierra

Como es sabido, el movimiento terrestre se compone de una traslación alrededor del Sol y un giro sobre su eje. En este movimiento, la Tierra se desplaza alrededor del Sol siguiendo una elipse de baja excentricidad en la que el Sol ocupa uno de los focos. La duración de este movimiento define el concepto de año. La corrección debida a la excentricidad de la elipse¹ se calcula con:

$$\epsilon_0 = 1 + 0,033 \cdot \cos\left(\frac{2\pi d_n}{365}\right) \quad (3.1)$$

donde d_n es el número de día del año (siendo $d_n = 1$ el 1 de enero).

El movimiento que describe la Tierra al girar alrededor del Sol, está contenido en un plano conocido como *plano de la eclíptica*, el cual, entre este y el eje polar (línea imaginaria que uno los dos polos de la Tierra), se forma un ángulo conocido como declinación², el cual se puede aproximar de forma sencilla de la siguiente manera³:

$$\delta = 23,45^\circ \cdot \sin\left(\frac{2\pi \cdot (d_n + 284)}{365}\right) \quad (3.2)$$

3.1.2. Movimiento aparente del sol

Las variaciones en la declinación hacen que los días fluctúen su duración. Esta duración se puede calcular mediante el ángulo del amanecer⁴ (ω_s), el cual determina la diferencia angular entre el mediodía solar y el momento en el que amanece. Por lo tanto, un día durará $2 \cdot |\omega_s|$ (ya que es el ángulo entre el amanecer y el mediodía, y el mediodía y el anochecer). Se puede calcular el ángulo del amanecer de la siguiente manera:

$$\omega_s = \begin{cases} -\arccos(-\tan \delta \tan \phi) & \text{si } |\tan \delta \tan \phi| < 1 \\ -\pi & \text{si } -\tan \delta \tan \phi < -1 \\ 0 & \text{si } -\tan \delta \tan \phi > 1 \end{cases} \quad (3.3)$$

¹Correspondiente a la función **eccentricity**.

²Correspondiente a la función **declination**

³Por razones de economización del espacio, se va a optar por utilizar las ecuaciones de Cooper [Coo69] por su sencillez. Sin embargo, la función **fSolD** (como se verá en el capítulo 4) puede seleccionar entre 4 tipos de ecuaciones expuestas por diferentes autores (Strous [Str11], Spencer [Spe71] y Michalsky [Mic88]).

⁴Correspondiente a la función **sunrise**.

donde ϕ corresponde a la latitud (positiva al norte y negativa al sur).

Sin embargo, cada localización tiene una hora oficial, TO , la cual no se corresponde con el ángulo que forma el Sol a lo largo del día, por lo que resulta necesario calcular la hora solar verdadera (ω)⁵:

$$\omega = 15 \cdot (TO - AO - 12) + \Delta\lambda + \frac{EoT}{4} \quad (3.4)$$

donde:

- TO es la hora oficial.
- AO es el adelanto oficial durante el horario de invierno.
- $\Delta\lambda$ es la diferencia entre la longitud local y la longitud del huso horario.
- EoT es la ecuación del tiempo⁶, se calcula de la siguiente manera:

$$EoT = 229,18 \cdot (-0,0334 \cdot \sin(\frac{2\pi}{365,24} \cdot dn) + 0,04184 \cdot \sin(2 \cdot \frac{2\pi}{365,24} \cdot dn + 3,5884)) \quad (3.5)$$

3.1.3. Radiación fuera de la atmósfera terrestre

La radiación extra-atmosférica es la radiación directa del Sol que alcanza la superficie de la atmósfera. El tamaño de la Tierra frente a la distancia que la separa del Sol es muy pequeña, por lo que es razonable asumir que su valor es constante en toda la superficie exterior de la atmósfera. Se define la constante solar B_0 como el valor de irradiancia solar incidente en un plano normal al vector Sol-Tierra. Se acepta como representativo el valor promedio de $B_0 = 1367W/m^2$.

Para calcular la irradiancia incidente en una superficie tangente a la atmósfera en una latitud determinada ($B_0(0)$)⁷, se toma la siguiente ecuación:

$$B_0(0) = B_0 \cdot \epsilon_0 \cdot \cos(\theta_{zs}) \quad (3.6)$$

donde, $\cos(\theta_{zs})$ ⁸ es el coseno del ángulo cenital solar⁹, que se calcula de la siguiente manera:

$$\cos(\theta_{zs}) = \cos(\delta)\cos(\omega)\cos(\phi) + \sin(\delta)\sin(\phi) \quad (3.7)$$

Es importante añadir que, con el ángulo cenital, se puede obtener otro ángulo relevante en la geometría solar. Se trata del ángulo azimutal, ψ_s ¹⁰, el cual es el ángulo formado por el meridiano solar y el meridiano del lugar (Sur en el hemisferio Norte y viceversa).

$$\cos(\psi_s) = signo(\phi) \cdot \frac{\cos(\delta)\cos(\omega)\cos(\phi) - \cos(\phi)\sin(\delta)}{\sin(\theta_{zs})} \quad (3.8)$$

⁵Correspondiente a la función `sunHour`.

⁶Correspondiente a la función `eot`.

⁷Este cálculo, junto con el de la hora solar, ω , el coseno del ángulo cenital, θ_{zs} y el ángulo azimutal, ψ_s , se pueden calcular mediante la función `fSolI`, haciendo uso de sus respectivas funciones.

⁸Correspondiente a la función `zenith`.

⁹El ángulo cenital, θ_{zs} , es el complementario de la altura solar, γ_s .

¹⁰Correspondiente a la función `azimuth`.

Para calcular la irradiación diaria extra-atmósferica¹¹, $B_{0d}(0)$ ¹², se puede integrar la ecuación 3.6:

$$B_{0d}(0) = -\frac{24}{\pi} B_0 \epsilon_0 (\omega_s \sin \phi \sin \delta + \cos \phi \cos \delta \sin \omega_s) \quad (3.9)$$

Es posible demostrar que el promedio mensual de esta irradiación diaria coincide numéricamente con el valor de irradiación diaria correspondiente a los denominados “días promedios”, días en los que la declinación correspondiente coincide con el promedio mensual (tabla 3.1).

TABLA 3.1: *Valor d_n correspondiente a los doce días promedio.*

Mes	Ene	Feb	Mar	Abr	May	Jun	Jul	Ago	Sep	Oct	Nov	Dic
d_n	17	45	74	105	135	161	199	230	261	292	322	347

Todas estas ecuaciones están recogidas en la función `calcSol`, la cual computa todos los ángulos solares, diarios e intradiarios, necesarios para la simulación de sistemas fotovoltaicos.

3.1.4. Influencia de la atmósfera

La radiación solar, al atravesar la atmósfera, es afectada por reflexión, atenuación y difusión, lo que altera sus características. La reflexión en las nubes reduce la radiación que llega a la Tierra, mientras que la absorción por vapor de agua, ozono y CO_2 cambia el espectro de la radiación. Además, la dispersión por partículas afecta la distribución espacial. Existen tres tipos de difusión según el tamaño de las partículas en interacción:

- **Difusión de Rayleigh:** la longitud de onda es mayor que el tamaño de la partícula, ocurre en las capas altas y causa el color azul del cielo.
- **Difusión de Mie:** la longitud de onda es similar al tamaño de la partícula, ocurre en las capas bajas.
- **Difusión no selectiva:** la longitud de onda es menor que el tamaño de la partícula.

Resulta útil definir la masa de aire (*AM, air mass*) como la relación entre el camino recorrido por los rayos directos del Sol a través de la atmósfera hasta la superficie receptora y el que recorrerían en caso de incidencia vertical. Se puede aproximar de la siguiente manera:

$$AM = 1 / \cos(\theta_{zs}) \quad (3.10)$$

Para el cálculo de la irradiancia solar que finalmente incide en una superficie arbitraria localizada en corteza terrestre, será útil distinguir tres contribuciones diferentes. Estas contribuciones, comúnmente denominadas *componentes*, son:

- **Radiación Directa, B :** representa la fracción de irradiación procedente en línea recta del Sol.

¹¹Correspondiente a la función `b00d`.

¹²Este cálculo, junto con la declinación, δ , la corrección por excentricidad, ϵ_0 , la ecuación el tiempo, EoT , y el ángulo del amanecer, ω_s , se pueden calcular mediante la función `fSolD`, haciendo uso de sus respectivas funciones.

- **Radiación Difusa**, D : fracción de radiación que procede de todo el cielo, excepto del Sol. Son todos aquellos rayos que dispersa la atmósfera.
- **Radiación del albedo**, R : parte de la radiación procedente de la reflexión con el suelo.

La suma de las tres componentes constituye la denominada radiación global:

$$G = B + D + R \quad (3.11)$$

3.1.5. Cálculo de componentes de radiación solar

Para caracterizar la radiación solar en un lugar, Liu y Jordan [LJ60] propusieron el índice de claridad, K_T . Este índice es la relación entre la radiación global y la radiación extra-atmosférica, ambas en el plano horizontal. El índice de claridad diario es la relación entre los valores diarios de irradiación:

$$K_{Td} = \frac{G_d(0)}{B_{0d}(0)} \quad (3.12)$$

mientras que el índice de claridad mensual es la relación entre las medias mensuales de la irradiación diaria:

$$K_{Tm} = \frac{G_{d,m}(0)}{B_{0d,m}(0)} \quad (3.13)$$

Una vez se tiene el índice de claridad, se puede calcular la fracción de radiación difusa en el plano horizontal. En el caso de medias mensuales [Pag61]:

$$F_{Dm} = 1 - 1,13 \cdot K_{Tm} \quad (3.14)$$

donde:

- Fracción de radiación difusa: $F_D = \frac{D(0)}{G(0)}$

Al tener la fracción de radiación difusa, se pueden obtener los valores de la radiación directa y difusa en el plano horizontal¹³:

$$D_d(0) = F_D \cdot G_d(0) \quad (3.15)$$

$$B_d(0) = G_d(0) - D_d(0) \quad (3.16)$$

Estas expresiones se recogen en la función `calcG0`, la cual calcula las componentes de la irradiancia intradiaria y la irradiación diaria (además de medias mensuales de estas y sumas anuales). Se vale de las funciones `fCompD`, para la irradiación, y `fCompI`, para la irradiancia.

3.2. Radiación en superficies inclinadas

Dados los valores de irradiación diaria difusa, directa y global en el plano horizontal se puede realizar la transformación al plano inclinado. Para ello, es necesario estimar el perfil de irradiancia correspondiente a cada valor de irradiación. Dado que la variación solar durante una hora es baja, podemos suponer que el valor medio de la irradiancia durante esa hora coincide numéricamente con la irradiación horaria. Por otra parte, el análisis de valores medios en largas series temporales ha mostrado que la relación entre la irradiancia y la irradiación extra-atmosférica [CR79] (3.17):

$$r_D = \frac{D(0)}{D_d(0)} = \frac{B_0(0)}{B_{0d}(0)} \quad (3.17)$$

¹³Correspondiente a la familia de funciones `corrFdKt`.

Este factor r_D es calculable directamente sabiendo que la relación entre irradiancia e irradiación extra-atmosférica es deducible teóricamente a partir de las ecuaciones 3.6 y 3.9:

$$\frac{B_0(0)}{B_{0d}(0)} = \frac{\pi}{T} \cdot \frac{\cos(\omega) - \cos(\omega_s)}{\omega_s \cdot \cos(\omega_s) - \sin(\omega_s)} = r_D \quad (3.18)$$

el mismo análisis mostró una relación entre la irradiancia e irradiación global asimilable a una función dependiente de la hora solar (3.19):

$$r_G = \frac{G(0)}{G_d(0)} = r_D \cdot (a + b \cdot \cos(w)) \quad (3.19)$$

donde:

- $a = 0,409 - 0,5016 \cdot \sin(\omega_s + \frac{\pi}{3})$
- $b = 0,6609 + 0,4767 \cdot \sin(\omega_s + \frac{\pi}{3})$

Es importante resaltar que estos perfiles proceden de medias sobre largos períodos, de ahí que, como es observable en la figura 3.2, las fluctuaciones propias del movimiento de nubes a lo largo del día queden atenuadas y se obtenga una curva sin alteraciones.

3.2.1. Transformación al plano del generador

Una vez obtenidos los valores de irradiancia en el plano horizontal, se traspone al plano del generador:

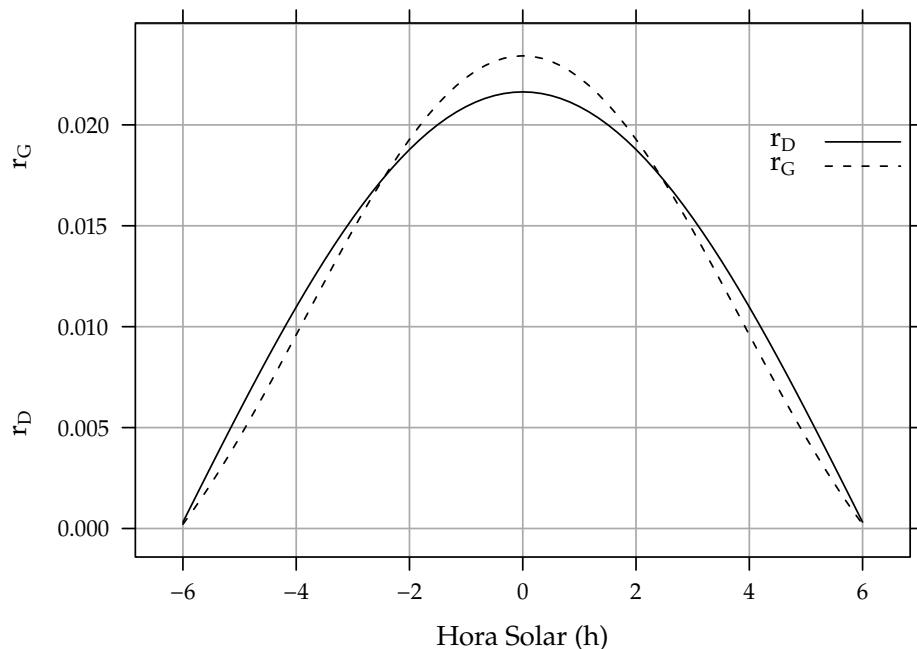


FIGURA 3.2: Perfil de irradiancia difusa y global obtenido a partir del generador empírico de [CR79] para valores de irradiancia tomadas cada 10 minutos. Figura 3.4 del libro ESF [Per23].

- **Irradiancia Directa** $B(\beta, \alpha)$: ecuación basada en geometría solar (ángulo zenital) y del generador (ángulo de incidencia).

$$B(\beta, \alpha) = B(0) \cdot \frac{\max(0, \cos(\theta_s))}{\cos(\theta_{zs})} \quad (3.20)$$

donde:

- Ángulo de inclinación: β .
 - Ángulo de orientación: α .

- **Irradiancia Difusa** $D(\beta, \alpha)$: utilizando el modelo de cielo anisotrópico [Per23], se distinguen dos componentes de la irradiancia difusa, denominados *circunsolar* e *isotrópica*.

$$D(\beta, \alpha) = D^I(\beta, \alpha) + D^C(\beta, \alpha) \quad (3.21)$$

$$D^I(\beta, \alpha) = D(0)(1 - k_1) \cdot \frac{1 + \cos(\beta)}{2} \quad (3.22)$$

$$D^C(\beta, \alpha) = D(0) \cdot k_1 \cdot \frac{\max(0, \cos(\theta_s))}{\cos(\theta_{z_s})} \quad (3.23)$$

donde:

- $$\bullet \quad k_1 = \frac{B(n)}{B_0 \cdot \epsilon_0} = \frac{B(0)}{B_0(0)}$$

- **Irradiancia de albedo** $R(\beta, \alpha)$: se considera isotrópica debido a su baja contribución a la radiación global. Se calcula a partir de la irradiancia global en el plano horizontal usando un coeficiente de reflexión, ρ , que depende del terreno. En la ecuación 3.24, se utiliza el factor $\frac{1-\cos(\beta)}{2}$, complementario al factor de visión de la difusa isotrópica (figura 3.3)

$$R(\beta, \alpha) = \rho \cdot G(0) \cdot \frac{1 - \cos(\beta)}{2} \quad (3.24)$$

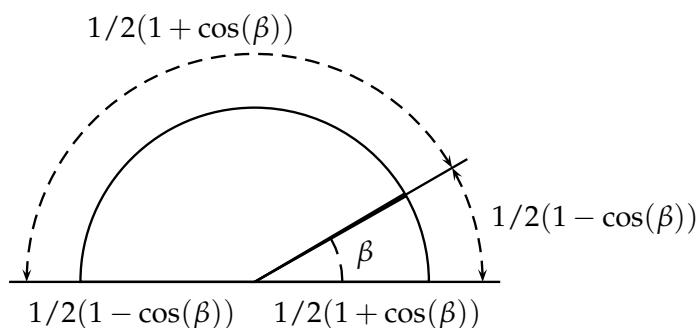


FIGURA 3.3: Ángulo de visión del cielo. Figura 3.5 del libro ESF [Per23].

3.2.2. Ángulo de incidencia y suciedad

En un módulo fotovoltaico, la radiación incidente generalmente no es perpendicular a la superficie del módulo, lo que provoca pérdidas por reflexión o pérdidas angulares, cuantificadas por el ángulo de incidencia θ_s . La suciedad acumulada en la superficie del módulo también reduce la transmitancia del vidrio (representada por $T_{limpio}(0)$), disminuyendo la irradiancia efectiva, es decir, la radiación que realmente puede ser aprovechada por el módulo. La irradiancia efectiva para radiación directa se expresa en la ecuación 3.25:

$$B_{ef}(\beta, \alpha) = B(\beta, \alpha) \cdot \left[\frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FTB(\theta_s)) \quad (3.25)$$

donde $FTB(\theta_s)$ es el factor de pérdidas angulares, que se calcula con la ecuación¹⁴ 3.26:

$$FTB(\theta_s) = \frac{\exp(-\frac{\cos(\theta_s)}{a_r}) - \exp(-\frac{1}{a_r})}{1 - \exp(-\frac{1}{a_r})} \quad (3.26)$$

Este factor depende del ángulo de incidencia, θ_s , y del coeficiente de pérdidas angulares, a_r . Cuando la radiación es perpendicular a la superficie ($\theta_s = 0$), FTB es cero. En la figura 3.4 se puede observar que las pérdidas angulares son más significativas cuando θ_s supera los 60°, y se acentúan con niveles mayores de suciedad.

Para calcular las componentes de radiación difusa isotrópica y de albedo, se utilizan las ecuaciones¹⁵ 3.27 y 3.28:

$$FTD(\beta) \approx \exp\left[-\frac{1}{a_r} \cdot \left(c_1 \cdot \left(\sin\beta + \frac{\pi - \beta - \sin\beta}{1 + \cos\beta}\right) + c_2 \cdot \left(\sin\beta + \frac{\pi - \beta - \sin\beta}{1 + \cos\beta}\right)^2\right)\right] \quad (3.27)$$

$$FTR(\beta) \approx \exp\left[-\frac{1}{a_r} \cdot \left(c_1 \cdot \left(\sin\beta + \frac{\beta - \sin\beta}{1 - \cos\beta}\right) + c_2 \cdot \left(\sin\beta + \frac{\beta - \sin\beta}{1 - \cos\beta}\right)^2\right)\right] \quad (3.28)$$

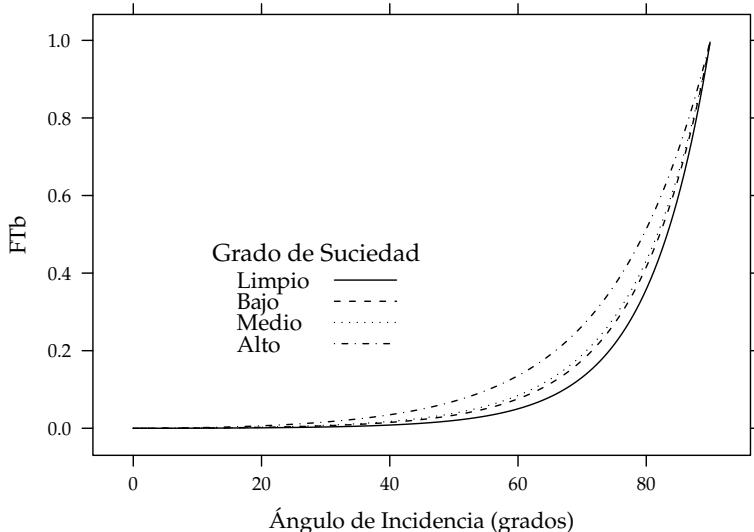


FIGURA 3.4: Pérdidas angulares de un módulo fotovoltaico para diferentes grados de suciedad en función del ángulo de incidencia. Figura 3.7 del libro ESF [Per23].

¹⁴Implementada en la función `fInclin`.

¹⁵Implementadas en la función `fInclin`.

donde:

- Ángulo de inclinación del generador (en radianes): β .
- Coeficiente de pérdidas angulares: a_r .
- Coeficientes de ajuste: c_1 y c_2 (en la tabla 3.2 se recogen algunos valores característicos de un módulo de silicio monocristalino convencional para diferentes grados de suciedad).

Para estas componentes el cálculo de irradiancia efectiva es similar al de la irradiancia directa (ecuaciones 3.29 y 3.31). Para la componente difusa circunsolar, se empleará el factor de pérdidas angulares de la irradiancia efectiva (ecuación 3.30):

$$D_{ef}^I(\beta, \alpha) = D^I(\beta, \alpha) \cdot \left[\frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FT_D(\beta)) \quad (3.29)$$

$$D_{ef}^C(\beta, \alpha) = D^C(\beta, \alpha) \cdot \left[\frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FT_B(\theta_s)) \quad (3.30)$$

$$R_{ef}(\beta, \alpha) = R(\beta, \alpha) \cdot \left[\frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FT_R(\beta)) \quad (3.31)$$

Siguiendo el esquema de la figura 3.1, a partir de estas irradiancias efectivas se puede calcular la irradiación global efectiva diaria, mensual y anual¹⁶. Comparando la irradiación global incidente con la irradiación efectiva, se puede evaluar el impacto de la suciedad y el desajuste del ángulo en períodos prolongados.

3.3. Cálculo de la energía producida por un generador fotovoltaico

3.3.1. Funcionamiento de una célula solar

Para calcular la energía producida por un generador fotovoltaico, se deben tener en cuenta la influencia de factores tales como la radiación o la temperatura en una célula solar¹⁷ y en los valores de tensión y corriente que se alcanzan en dichas condiciones.

Para definir una célula solar, se tomar 4 variables:

TABLA 3.2: Valores del coeficiente de pérdidas angulares y transmitancia relativa en incidencia normal para diferentes tipos de suciedad.

Grado de Suciedad	$\frac{T_{sucio}(0)}{T_{limpio}(0)}$	a_r	c_2
Limpio	1	0.17	-0.069
Bajo	0.98	0.20	-0.054
Medio	0.97	0.21	-0.049
Alto	0.92	0.27	-0.023

¹⁶Correspondiente a la función `calcGef`.

¹⁷Los cálculos de este apartado quedan recogidos en la función `fProd`.

- La corriente de cortocircuito: I_{sc} .
- La tensión de circuito abierto: V_{oc} .
- La corriente en el punto de máxima potencia: I_{mpp} .
- La tensión en el punto de máxima potencia: V_{mpp} .

Punto de máxima potencia

El punto de máxima potencia es aquel situado en la curva de funcionamiento del generador donde, como su propio nombre indica, los valores de tensión y corriente son tales que la potencia que entrega es máxima (figura 3.5).

Factor de forma y eficiencia

El área encerrada por el rectángulo definido por el producto $I_{mpp} \cdot V_{mpp}$ es, como se puede apreciar en la figura 3.5, inferior a la representada por el producto $I_{sc} \cdot V_{oc}$. La relación entre estas dos superficies se cuantifica con el factor de forma:

$$FF = \frac{I_{mpp} \cdot V_{mpp}}{I_{sc} \cdot V_{oc}} \quad (3.32)$$

Conocidos los valores de I_{sc} y V_{oc} , es posible calcular la potencia en el punto de máxima potencia, dado que $P_{mpp} = FF \cdot I_{sc} \cdot V_{oc}$.

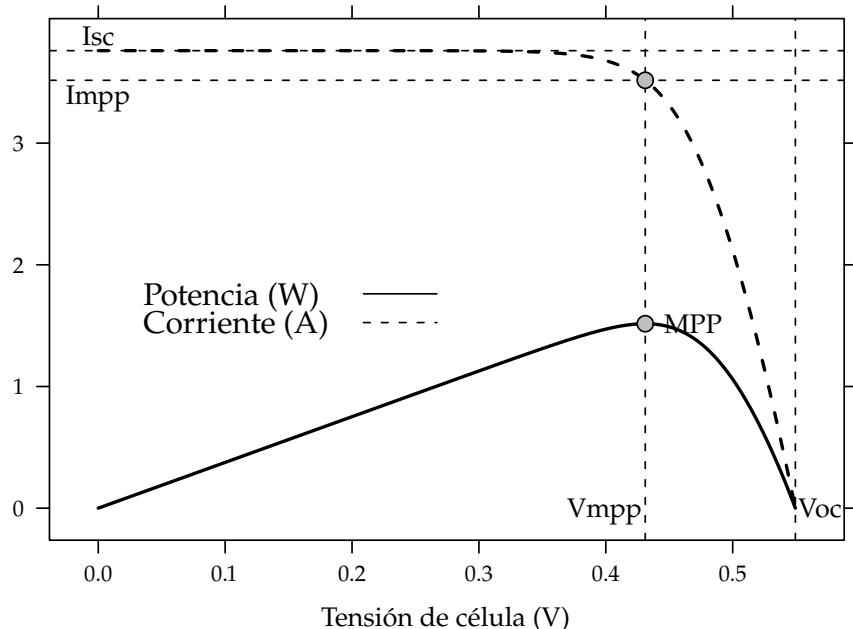


FIGURA 3.5: Curvas corriente-tensión (línea discontinua) y potencia-tensión (línea continua) de una célula solar ($T_a = 20^\circ C$ y $G = 800 W/m^2$). Figura 4.6 del libro ESF [Per23].

Por otra parte, la calidad de una célula se puede cuantificar con la eficiencia de conversión (ecuación 3.33).

$$\eta = \frac{I_{mpp} \cdot V_{mpp}}{P_L} \quad (3.33)$$

donde $P_L = A_c \cdot G_{ef}$ representa la potencia luminosa que incide en la célula. Como es evidente de la ecuación 3.33, este valor de eficiencia se corresponde al caso en el que el acoplamiento entre la carga y la célula permite a esta trabajar en el punto de máxima potencia. En la figura 3.6 se muestra la evolución temporal del valor de eficiencia de célula de laboratorio para diferentes tecnologías.

Influencia de la temperatura y la radiación

La temperatura y la radiación son factores cruciales en el funcionamiento de una célula solar. El aumento de la temperatura ambiente reduce la tensión de circuito abierto según la relación dV_{oc}/dT_c , que para células de silicio cristalino es de $-2,3 \text{ mV}^{\circ\text{C}}$. Además, disminuye la eficiencia de la célula solar con $\frac{d\eta}{dT_c} = -0,4\%/\text{ }^{\circ}\text{C}$.

En cuanto a la iluminación, la photocorriente y la tensión de circuito abierto son proporcionales a la irradiancia incidente.

Tomando en cuenta estas influencias, se definen una condiciones de funcionamiento, denominadas condiciones estándar de medida (STC), válidas para caracterizar una célula en el entorno de un laboratorio. Estas condiciones vienen determinadas por:

- Irradiancia: $G_{stc} = 1000 \text{ W/m}^2$ con incidencia normal.
- Temperatura de célula: $T_c^* = 25^{\circ}\text{C}$.
- Masa de aire: $AM = 1,5^{18}$.

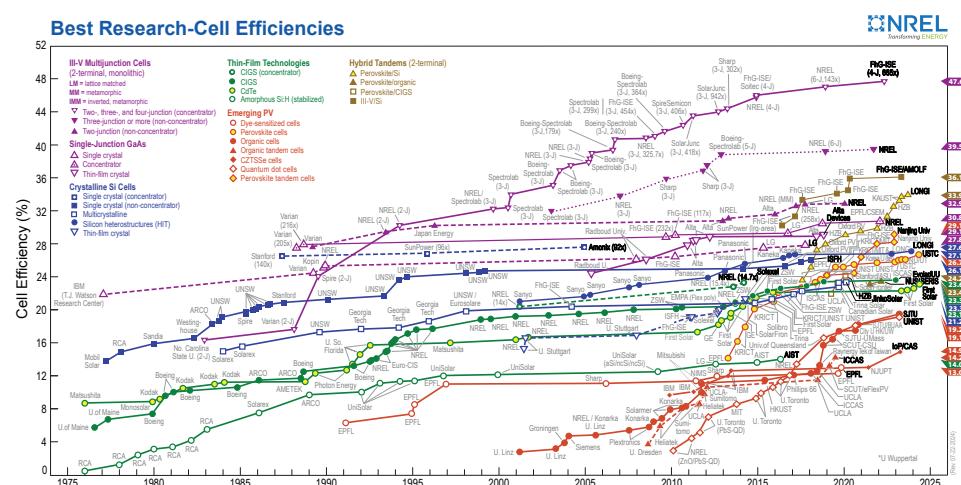


FIGURA 3.6: Evolución de la eficiencia de células según la tecnología (según el National Renewable Energy Laboratory [Nat24] (EEUU)).

¹⁸ Relación entre el camino recorrido por los rayos directos del Sol a través de la atmósfera hasta la superficie receptora y el que recorrerían en caso de incidencia vertical ($AM = 1/\cos\theta_{zs}$).

Frecuentemente los fabricantes informan de los valores de las tensiones V_{oc}^* y V_{mpp}^* y las corrientes I_{sc}^* y I_{mpp}^* ¹⁹. A partir de estos valores es posible referir a estas condiciones:

- La potencia: $P_{mpp}^* = I_{mpp}^* \cdot V_{mpp}^*$.
- El factor de forma: $FF^* = \frac{P_{mpp}^*}{I_{sc}^* \cdot V_{oc}^*}$.
- La eficiencia: $\eta^* = \frac{I_{mpp}^* \cdot V_{mpp}^*}{A_c \cdot G_{stc}}$.

3.3.2. Funcionamiento de un módulo fotovoltaico

Comportamiento térmico de un módulo

La mayoría de las ecuaciones que definen el comportamiento de un módulo fotovoltaico²⁰ se establecen en lo que se conoce como condiciones estándar de funcionamiento. En estas condiciones, la temperatura de la célula es de $25^\circ C$. Sin embargo, la temperatura de operación de la célula es diferente y depende directamente de la radiación que recibe el módulo en cada momento.

El módulo recibe una cantidad de radiación dada, absorbiendo la fracción de esta que no se refleja al exterior. De dicha fracción, parte de ella es transformada en energía eléctrica, mientras que el resto se entrega en forma de calor al entorno.

Para simplificar, se puede asumir que el incremento de la temperatura de la célula respecto de la temperatura ambiente depende linealmente de la irradiancia incidente en esta. El coeficiente de proporcionalidad depende de muchos factores, tales como el modo de instalación del módulo, la velocidad del viento, la humedad ambiente y las características constructivas del laminado.

Estos factores quedan recogidos en un valor único representado por la temperatura de operación nominal de célula (NOCT o TONC), definida como aquella que alcanza una *célula* cuando su *módulo* trabaja en las siguientes condiciones:

- Irradiancia: $G = 800 W/m^2$.
- Masa de aire: $AM = 1,5$.
- Irradiancia normal.
- Temperatura ambiente: $T_a = 20^\circ C$.
- Velocidad de viento: $v_v = 1 m/s$.

La ecuación 3.34 expresa una aproximación aceptable del comportamiento térmico de una célula integrada en un módulo en base a las consideraciones previas:

$$T_c = T_a + G_{ef} \cdot \frac{NOCT - 20}{800} \quad (3.34)$$

Para la simulación del funcionamiento de un módulo fotovoltaico en condiciones de operación real, es necesario contar con secuencias de valores de temperatura ambiente. Si no se dispone de información detallada, se puede asumir un valor constante de $T_a = 25^\circ C$ para simulaciones

¹⁹Es de uso común añadir un asterisco como superíndice para denotar aquellos parámetros medidos en estas condiciones.

²⁰Los cálculos de este apartado, quedan recogidos en la función **fProd**.

anuales. Sin embargo, si se conocen los valores máximos y mínimos diarios de la temperatura ambiente, se puede generar una secuencia intradiaria usando una combinación de funciones coseno:

- 1. Calcular los valores de T_m y T_r :** Se definen los valores de la temperatura media diaria, T_m simplemente como la media entre la T_{max} y la T_{min} :

$$T_m = \frac{T_{max} + T_{min}}{2} \quad (3.35)$$

y la mitad de la amplitud de temperaturas, T_r :

$$T_r = \frac{T_{max} - T_{min}}{2} \quad (3.36)$$

- 2. Calcular la matriz de T_a :** Una vez obtenidos los valores de T_m y T_r , el siguiente paso es calcular la matriz de valores de temperatura ambiental T_a para cada hora. Para ello, se deben calcular 3 valores auxiliares:

$$a_1 = \frac{12\pi \cdot (\omega_s - \omega)}{21\pi + 12 \cdot \omega_s} \quad (3.37)$$

$$a_2 = \frac{\pi \cdot (3\pi - 12\omega)}{3\pi - 12\omega} \quad (3.38)$$

$$a_3 = \frac{\pi(24\pi + 12 \cdot (\omega_s - \omega))}{21 \cdot \pi + 12 \cdot \omega_s} \quad (3.39)$$

Con estos tres valores se pueden calcular la terna de temperaturas T_1 , T_2 , T_3 :

$$T_1 = T_m - T_r \cdot \cos(a_1) \quad (3.40)$$

$$T_2 = T_m + T_r \cdot \cos(a_2) \quad (3.41)$$

$$T_3 = T_m - T_r \cdot \cos(a_3) \quad (3.42)$$

- 3. Selección del valor clave:** Se selecciona el valor adecuado de T_a en base al siguiente criterio:

$$T_a = \begin{cases} T_1 & \text{si } \omega \leq \omega_s \\ T_2 & \text{si } \omega_s < \omega \leq \frac{\pi}{4} \\ T_3 & \text{si } \omega > \frac{\pi}{4} \end{cases} \quad (3.43)$$

Cálculo de V_{oc} y I_{sc}

Conocidos los valores horarios de temperatura de la célula, se puede calcular V_{oc} utilizando la ecuación 3.44. Y, por último, mediante la ecuación 3.45 se puede calcular I_{sc} .

$$V_{oc}(T_c) = V_{oc}^* + (T_c - T_c^*) \cdot \frac{dV_{oc}}{dT_c} \cdot N_{cs} \quad (3.44)$$

$$I_{sc} = G_{ef} \cdot \frac{I_{sc}^*}{G^*} \quad (3.45)$$

Factor de forma variable

Una vez obtenidos los valores de V_{oc} y I_{sc} , el siguiente paso ha de ser calcular los valores de tensión y corriente en el punto de máxima potencia, pues es donde el generador estará entregando su máxima potencia, como su propio nombre indica. Por tanto, es un punto de interés para el cálculo.

Existen dos metodologías de cálculo de dicho punto, uno de ellos significantemente más sencillo que el otro. Este consiste en suponer que el Factor de Forma, definido en la expresión 3.32, es constante.

Si suponemos que FF es constante, se podrían extraer los valores de tensión y corriente en el punto de máxima potencia, ya que si

$$FF = FF^* \quad (3.46)$$

entonces

$$\frac{I_{mpp} \cdot V_{vmpp}}{I_{sc} \cdot V_{oc}} = \frac{I_{mpp}^* \cdot V_{vmpp}^*}{I_{sc}^* \cdot V_{oc}^*} \quad (3.47)$$

pudiendo así obtener los valores de I_{mpp} y V_{vmmp} .

Sin embargo, este suposición da resultados alejados a una estimación acertada. Por ello, se tendrá en cuenta la variación del factor de forma:

- **Cálculo de la tensión térmica, V_t , a temperatura de la célula:** se calculará el valor de V_t a 25°C con la expresión:

$$V_{tn} = \frac{V_t \cdot (273 + 25)}{300} \quad (3.48)$$

- **Cálculo de R_s^* :** el segundo paso consiste en calcular el valor de resistencia en serie con los valores STC:

$$R_s^* = \frac{\frac{V_{oc}^*}{N_{cs}} - \frac{V_{mpp}^*}{N_{cs}} + m \cdot V_{tn} \cdot \ln(1 - \frac{I_{mpp}^*}{I_{sc}^*})}{\frac{I_{mpp}^*}{N_{cp}}} \quad (3.49)$$

- **Cálculo de r_s :** utilizando el valores de R_s^* , calculado en el paso anterior junto con los valores de V_{oc} y I_{sc} , podemos calcular r_s que se utilizará más adelante en el proceso.

$$r_s = R_s^* \cdot \left(\frac{N_{cs}}{N_{cp}} \cdot \frac{I_{sc}}{V_{oc}} \right) \quad (3.50)$$

- **Cálculo de k_{oc} :** a continuación, utilizando los valores de temperatura ambiente obtenidos con anterioridad junto con la tensión de circuito abierto, se calcula k_{oc} mediante la expresión:

$$k_{oc} = \frac{V_{oc}/N_{cs}}{m \cdot V_t \cdot \frac{T_c+273}{300}} \quad (3.51)$$

Con los cálculos previos, este método propone localizar el punto de máxima potencia de forma aproximada mediante las ecuaciones:

$$i_{mpp} = 1 - \frac{D_M}{k_{oc}} \quad (3.52)$$

$$v_{mpp} = 1 - \frac{\ln(k_{oc}/D_M)}{k_{oc}} - r_s \cdot i_{mpp} \quad (3.53)$$

donde:

$$D_M = D_{M0} + 2 \cdot r_s \cdot D_{M0}^2 \quad (3.54)$$

$$D_{M0} = \frac{k_{oc} - 1}{k_{oc} - ln k_{oc}} \quad (3.55)$$

Por último, multiplicando los valores de i_{mpp} y v_{mpp} por I_{sc} y V_{oc} respectivamente, se obtienen los valores de I_{mpp} y V_{mpp} , que serán los que se utilicen para calcular la potencia entregada por el generador en el punto de máxima potencia.

Teniendo estos valores se puede obtener:

$$P_{mpp} = I_{mpp} \cdot V_{mpp} \quad (3.56)$$

3.3.3. Cálculo de potencias y energías de un sistema fotovoltaico conectado a la red

La potencia obtenida en el paso anterior es la de un solo módulo. Para conocer la potencia que va a ser capaz de entregar un SFCR, se debe tener en cuenta su configuración de módulos en serie y en paralelo²¹.

$$P_g^* = N_s \cdot N_p \cdot P_m^* \quad (3.57)$$

Con este paso se obtiene la potencia horaria entregada por el generador fotovoltaico. El siguiente paso será pasar esa potencia a través del inversor y calcular la potencia a la salida de este.

Primero, se establecen las expresiones de las potencias normalizadas, siendo P_{inv} la potencia nominal del inversor:

$$p_i = \frac{P_{DC}}{P_{inv}} \quad (3.58)$$

$$p_o = \frac{P_{AC}}{P_{inv}} \quad (3.59)$$

Por otro lado, el rendimiento de un inversor fotovoltaico se puede modelizar de la siguiente manera:

$$\eta_{inv} = \frac{p_o}{p_o + k_0 + k_1 p_o + k_2 p_o^2} \quad (3.60)$$

De las dos ecuaciones anteriores se puede deducir:

$$p_i = p_o + k_0 + k_1 p_o + k_2 p_o^2 \quad (3.61)$$

Desarrollando esta ecuación, se puede obtener una ecuación de segundo grado con p_o como incógnita:

$$k_2 p_o^2 + (k_1 + 1) p_o + (k_0 - p_i) = 0 \quad (3.62)$$

Por último, volviendo a las primeras expresiones, se puede obtener la potencia en corriente alterna:

$$P_{AC} = p_o \cdot P_{inv} \quad (3.63)$$

²¹Los cálculos de este apartado, quedan recogidos en las funciones `fProd` y `prodCGPV`.

Con esta potencia, integrando en función del tiempo, se puede obtener la energía que genera el sistema

$$E_{AC} = \int_T P_{AC} dt \quad (3.64)$$

y la productividad:

$$Y_f = \frac{E_{ac}}{P_g^*} \quad (3.65)$$

3.3.4. Cálculo de potencias y energías de un sistema fotovoltaico de bombeo

Potencia hidráulica

La potencia hidráulica, P_H , necesaria para bombear agua es función de:

- La altura vertical aparente, H_v
- El caudal de agua, Q

$$P_H = g \cdot \rho \cdot Q \cdot H_V \quad (3.66)$$

Expresando P_H en watios, H_v en metros y Q en m^3/h la ecuación resulta en:

$$P_H = 2,725 \cdot Q \cdot H_v \quad (3.67)$$

Asumiendo que el agua bombeado sale por el conducto a baja velocidad, la potencia de salida de la bomba necesita satisfacer P_H , más las pérdidas de fricción en la tubería, P_f . Este valor se asimila a una altura equivalente H_f , asociado a un caudal determinado:

$$H_T = H_v + H_f \quad (3.68)$$

La potencia eléctrica a la entrada de la motobomba, P_{el} , es:

$$P_{el} = \frac{P_H + P_f}{\eta_{mp}} \quad (3.69)$$

donde η_{mp} es la eficiencia de la motobomba. La potencia eléctrica requerida por la motobomba es entregada por un generador FV y acondicionador de potencia:

$$P_{el} = P_g^* \cdot \frac{G}{G_{stc}} \frac{\eta_g}{\eta_g^*} \cdot \eta_{inv} \quad (3.70)$$

siendo G la irradiancia en el plano del generador, η_{inv} la eficiencia del equipo de acondicionamiento de potencia y $\frac{\eta_g}{\eta_g^*}$ modela el comportamiento del generador con la temperatura.

Caudal diario

El caudal diario bombeado por este conjunto es:

$$Q_d = \int_d \frac{G}{G^*} \cdot P_g^* \cdot \eta_g \cdot \frac{\eta_{ig}}{\eta_{ig}^*} \cdot \eta_{inv} \cdot \eta_{mp} dt \quad (3.71)$$

Altura

Se puede definir una altura total equivalente, H_{TE} , con las siguientes suposiciones:

- Las pérdidas de fricción en tubería son despreciables ($H_f < 0,05 \cdot H_T$).
- El nivel del agua dentro del pozo se mantiene constante.

$$Q_d = \frac{P_g^*}{2,725 \cdot G^* \cdot H_{TE}} \int \left(\frac{G}{\eta_{ig}^* \eta_m^* \eta_{inv} \eta_{mp}} \right) dt \quad (3.72)$$

Ahora el cálculo en la integral solo depende de la radiación, temperatura, y equipos.

Para calcular esta altura total equivalente, se debe suponer que:

- El pozo está caracterizado con tres parámetros:
 - Nivel estático, H_{st} .
 - Nivel dinámico, H_{dt} .
 - Caudal de ensayo, Q_t .
- Que se ha realizado el ensayo de bombeo para caracterizar los pozos con bomba portátil empleando el caudal máximo del pozo, Q_{max} ($Q_t = Q_{max}$).

Con estas suposiciones se puede llegar a la expresión:

$$H_{TE} = H_{ot} + H_{st} + \left(\frac{H_{dt} - H_{st}}{Q_T} \right) Q_{AP} + H_f(Q_{AP}) \quad (3.73)$$

donde:

- H_{OT} , es la altura desde la salida de agua hasta el suelo.
- Nivel estático, H_{st} .
- Nivel dinámico, H_{dt} .
- Caudal aparente, $Q_{AP} = \alpha \cdot Q_d$ ($\alpha = 1/24 = 0,0416 h^{-1}$).
- $H_f(Q_{AP})$, pérdidas en la tubería al caudal aparente.

Potencia del generador

Como primera aproximación, se consideran constantes a lo largo del tiempo las eficiencias de los componentes del sistema con la elección de ciertos valores adecuados ($\frac{\eta_g}{\eta_g^*} = 0,85$, $\eta_{mp} = 0,35$, $\eta_{inv} = 0,9$). Así, es posible calcular de forma aproximada la potencia nominal del generador necesaria para bombear un caudal diario Q_d a una altura total equivalente H_{TE} a partir de la ecuación:

$$P_g^* = \frac{10 \cdot H_{TE} \cdot Q_d}{\frac{G_d}{G_{stc}}} \quad (3.74)$$

Simulación de sistemas fotovoltaicos de bombeo

Debido a la complicación del cálculo del dimensionamiento de los sistemas fotovoltaicos de bombeo, se puede recurrir a métodos de simulación asistidos por ordenador²². El algoritmo a seguir es:

1. Curva característica de la bomba que relaciona la altura, H , y el caudal, Q , a la frecuencia nominal de la bomba:

$$H = a \cdot f^2 + b \cdot f \cdot Q + c \cdot Q^2 \quad (3.75)$$

■ Donde a , b , y c son coeficientes característicos de la bomba y f es la frecuencia.

2. Relaciones de semejanza para bombas centrífugas:

$$\frac{f_1}{f_2} = \frac{Q_1}{Q_2} = \left(\frac{H_1}{H_2}\right)^{1/2} = \left(\frac{P_1}{P_2}\right)^{1/3} \quad (3.76)$$

3. Cálculo de caudal y altura a frecuencia nominal (50 Hz):

$$Q_{50} = \frac{50 \cdot Q}{f} \quad (3.77)$$

$$H_{50} = H \cdot \left(\frac{50}{f}\right)^2 \quad (3.78)$$

4. Ecuación de potencia hidráulica:

$$P_{h,50} = 2,725 \cdot Q_{50} \cdot H_{50} \quad (3.79)$$

5. Potencia mecánica en el eje de la bomba a 50 Hz:

$$P_{b,50} = \frac{P_{h,50}}{\eta_b} \quad (3.80)$$

6. Potencia mecánica a frecuencia f :

$$P_b = P_{b,50} \cdot \left(\frac{f}{50}\right)^3 \quad (3.81)$$

7. Potencia eléctrica demandada por el motor:

$$P_{bc} = P_b \cdot \frac{50}{f} \quad (3.82)$$

$$P_{e,50} = \frac{P_{bc}}{\eta_m} \quad (3.83)$$

$$P_e = P_{e,50} \cdot \frac{f}{50} \quad (3.84)$$

8. Perfil de irradiancia diaria (según IEC 61725):

$$G = G_{max} \cdot \cos\left(\frac{t}{t_0} \cdot \frac{\pi}{2}\right) \cdot \left[1 + s \cdot \left(1 - \cos\left(\frac{t}{t_0} \cdot \frac{\pi}{2}\right)\right)\right] \quad (3.85)$$

donde G es la irradiancia (W/m^2) en la hora t , G_{max} es el valor máximo de irradiancia (W/m^2) durante el día en cuestión y s es el factor de forma definido por:

$$s = \frac{d \cdot \frac{\pi}{2} - 1}{1 - \frac{\pi}{4}} \quad (3.86)$$

siendo d el factor de conjunto de datos calculado con:

$$d = \frac{G_d}{G_{max} \cdot h} \quad (3.87)$$

²²Correspondientes a la función **fPump**.

3.4. Sombras y ocupación de terreno

Al diseñar una central fotovoltaica se debe decidir la ubicación de las diferentes partes del generador resolviendo un compromiso entre la mejor ocupación del terreno disponible y la minimización del impacto de sombras mutuas arrojadas entre los módulos²³.

Este factor de sombras implica un nivel de ocupación de terreno que depende del modo de seguimiento del generador. La ocupación del terreno se puede medir con dos métricas:

- **Relación de ocupación del terreno** (*Ground Coverage Ratio, GCR*): es la relación entre el área del generador, A_G , y el área del terreno ocupado, A_T (por tanto, siempre será $GCR < 1$).

$$GCR = \frac{A_G}{A_T} \quad (3.88)$$

- **Ratio de ocupación del terreno** (*ROT*, o *Ground Requirement Ratio, GRR*): es el inverso del GCR, la relación entre el área de terreno ocupado, A_T , y el área del generador, A_G .

$$ROT = \frac{A_T}{A_G} \quad (3.89)$$

3.4.1. Sistemas estáticos

Las filas que componen el generador arrojan sombras unas sobre otras en determinados momentos del días y año. Como recomendación general, es de uso común respetar un mínimo de 4 horas de sol en torno al mediodía del solsticio de invierno libres de sombra. La longitud de la sombra de un obstáculo se mide con:

$$d = \frac{h}{\tan \gamma_s} \quad (3.90)$$

siendo h la altura de la fila adyacente, $h = L \cdot \sin(\beta)$, y L la longitud del generador, según se indica en la figura 3.7.

En el mediodía del solsticio de invierno, la altura solar es $\gamma_s = 90^\circ - 23,45^\circ - |\phi| \simeq 67^\circ - |\phi|$. Por tanto, la distancia mínima que permite 4 horas libres de sombra alrededor del mediodía es:

$$d_{min} = \frac{h}{\tan(61^\circ - |\phi|)} \quad (3.91)$$

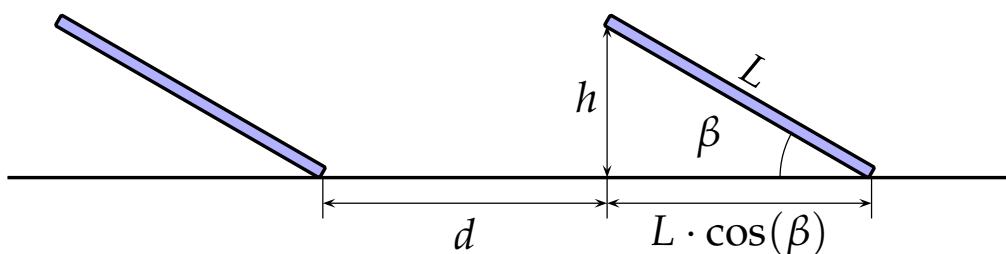


FIGURA 3.7: Dimensiones y distancias entre filas de un sistema estático. Figura 6.10 del libro ESF [Per23].

²³Correspondiente a las funciones `calcShd` (cálculo de sombras) y `optimShd` (optimización de distancias).

3.4.2. Sistemas de seguimiento a doble eje

El diseño de un sistema de seguimiento solar a doble eje busca optimizar la ubicación de los seguidores para minimizar las pérdidas de radiación por sombras, utilizando eficientemente el terreno. Para esto, se simula el sistema en diferentes configuraciones y se elige la más eficiente en términos de productividad y ROT, que se calcula con la fórmula:

$$ROT = \frac{L_{ns} \cdot L_{eo}}{L \cdot W} \quad (3.92)$$

donde (figuras 3.8 y 3.9):

- L_{ns} es la separación entre seguidores en la dirección Norte-Sur.
- L_{eo} es la separación en la dirección Este-Oeste.
- L es la longitud del seguidor.
- W es la anchura del seguidor.

El sistema se modela como un grupo de seis seguidores en una matriz de dos filas en dirección Norte-Sur (figura 3.10), representando tres situaciones de sombra: lateral (Este-Oeste), frontal (Norte-Sur) y diagonal, caracterizados por los factores de sombra FS_{xx} (definidos como la relación entre el área sombreada y el área total del generador). Las ecuaciones para estos factores son, en las que se emplean los valores normalizados de las distancias, $l_{eo} = \frac{L_{eo}}{W}$ y $l_{ns} = \frac{L_{ns}}{W}$:

$$\begin{aligned} |l_{eo} \cdot \cos(\psi_s)| < 1 \\ |l_{eo} \cdot \sin(\psi_s)| < s \end{aligned} \Rightarrow FS_{eo} = \frac{(1 - |l_{eo} \cos(\psi_s)|) \cdot (s - |l_{eo} \sin(\psi_s)|)}{s} \quad (3.93)$$

$$\begin{aligned} |l_{ns} \cdot \cos(\psi_s)| < s \\ |l_{ns} \cdot \sin(\psi_s)| < 1 \end{aligned} \Rightarrow FS_{ns} = \frac{(s - |l_{ns} \cos(\psi_s)|) \cdot (1 - |l_{ns} \sin(\psi_s)|)}{s} \quad (3.94)$$

$$\begin{aligned} s > |l_{ns} \cdot \cos(\psi_s)| + |l_{eo} \sin(\psi_s)| \\ 1 > |l_{eo} \cdot \cos(\psi_s)| - |l_{ns} \cdot \sin(\psi_s)| \end{aligned} \Rightarrow$$

$$FS_d = \frac{[s - (|l_{eo} \cdot \sin(\psi_s)| + |l_{ns} \cos(\psi_s)|)] \cdot [1 - (|l_{eo} \cdot \cos(\psi_s)| - |l_{ns} \sin(\psi_s)|)]}{s} \quad (3.95)$$

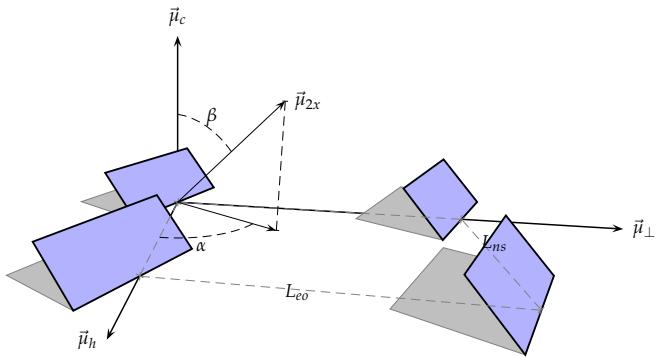


FIGURA 3.8: Sombras mutuas en un conjunto de cuatro seguidores. Figura 6.11 del libro ESF [Per23].

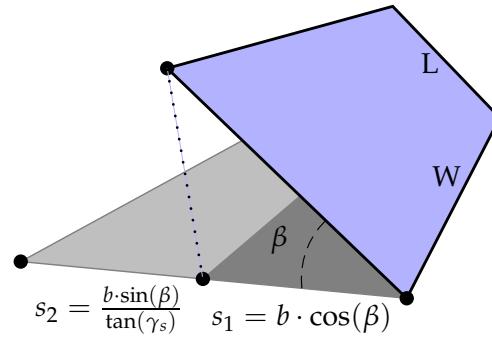


FIGURA 3.9: Dimensiones de un seguidor a doble eje y longitud de su sombra arrojada. Figura 6.12 del libro ESF [Per23].

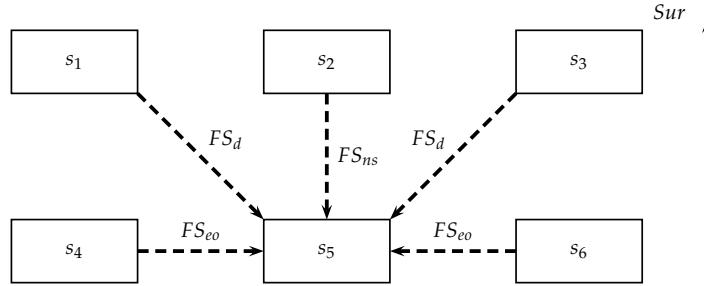


FIGURA 3.10: Posibles sombras en un conjunto de seis seguidores. Figura 6.13 del libro ESF [Per23].

siendo ψ_s el acimut solar y γ_s la altura solar y donde la longitud de sombra (normalizada con la anchura del seguidor) se calcula con:

$$s = s_1 + s_2 \quad (3.96)$$

$$s_1 = b \cdot \cos(\beta) \quad (3.97)$$

$$s_2 = \frac{b \cdot \sin(\beta)}{|\tan(\gamma_s)|} \quad (3.98)$$

El factor $\frac{\sin(\gamma_s)}{\sin(\gamma_s + \beta)}$ representa la proyección de sombra existente en el suelo sobre el plano del generador, y por tanto, el porcentaje de área sombreada que debe ser eliminado de la radiación directa. Desarrollando este factor, se obtiene una formulación alternativa que puede facilitar el cálculo de los tres factores:

$$FS_{eo} = \frac{(1 - l_{eo} \cos(\psi_s)) \cdot (s - l_{eo} \sin(\psi_s))}{s} \quad (3.99)$$

$$FS_{ns} = \frac{(s - l_{ns} \cos(\psi_s)) \cdot (1 - l_{ns} \sin(\psi_s))}{s} \quad (3.100)$$

$$FS_d = \frac{[s - (l_{eo} \cdot \sin(\psi_s) + l_{ns} \cos(\psi_s))] \cdot [1 - (l_{eo} \cdot \cos(\psi_s) - l_{ns} \sin(\psi_s))]}{s} \quad (3.101)$$

Realizando la simulación de este sistema, incluyendo el cálculo de sombras y repitiendo la simulación para varias combinaciones (Lns, Leo) pueden elaborarse gráficos de nivel como el de la figura 3.11. En estas, se recoge el ratio entre la energía anual producida por un seguidor *promedio* incluyendo el efecto de por sombras mutuas²⁴ y la energía anual producida por un seguidor sin sombreado.

3.4.3. Sistemas de seguimiento de eje horizontal

Se considera que los seguidores son de longitud infinita en sentido Norte-Sur (se desprecia el efecto de borde). Así, los parámetros que determinan el diseño de este tipo de sistema son (figura 3.12):

1. La inclinación del generador fotovoltaico, β , (coincidente con el ángulo ψ_{ns}).
2. La dimensión en sentido Este-Oeste del campo generador, L.
3. La separación entre los diferentes seguidores en la dirección Este-Oeste, L_{eo} . Por tanto, $ROT = \frac{L_{eo}}{L}$.

Para caracterizar numéricamente el sombreado, se empleará el factor FS_{eo} . Mediante consideraciones geométricas, utilizando la distancia normalizada $l_{eo} = \frac{L_{eo}}{L}$, es posible escribir:

$$\begin{aligned} FS_{eo} &= \frac{s - l_{eo}}{s} \\ &= 1 - l_{eo} \cdot \cos(\beta) \\ &= 1 - l_{eo} \cdot \frac{\sin(\omega)}{\sqrt{\sin^2(\omega) + (\cos(\omega) \cos(\phi) + \tan(\delta) \sin(\phi))^2}} \end{aligned} \quad (3.102)$$

Limitación de ángulo y retroseguimiento

En seguidores de eje horizontal se puede evitar la incidencia de sombras en cualquier en cualquier instante mediante algoritmos de *backtracking* o retroseguimiento [Pan+91]. Esta técnica provoca el desvío del seguidor de su posición óptima en los instantes en los que se produce la sombra entre seguidores, evitando el impacto de sombras pero con la consiguiente reducción en energía producida pro alejamiento del apuntamiento óptimo.

Para evitar la aparición de sombras, el ángulo de inclinación de los seguidores debe ser tal que la longitud de la sombra sea igual a la distancia entre seguidores. Siendo β el ángulo de inclinación con retroseguimiento, y β_0 el ángulo de inclinación original. De la ecuación 3.102 se deduce que sólo será necesario aplicar esta técnica cuando $l_{eo} \cdot \cos(\beta_0) \leq 1$. El triángulo definido por el rayo solar, el seguidor y la sombra debe cumplir la siguiente condición, basada en el teorema de los senos:

$$\frac{l_{eo}}{\cos(\beta_0 - \beta)} = \frac{1}{\cos \beta_0} \quad (3.103)$$

Por tanto, el ángulo de inclinación que garantiza la ausencia de sombras a costa de apartarse de la condición de seguimiento es:

$$\beta = \beta_0 - \arccos(l_{eo} \cdot \cos \beta_0) \quad (3.104)$$

ecuación que debe aplicarse solo cuando $l_{eo} \cdot \cos(\beta_0) \leq 1$. En caso contrario, $\beta = \beta_0$.

²⁴En el cálculo de la producción del seguidor afectado por sombras mutuas, se considera que la reducción en potencia está exclusivamente relacionada con el área sombreada, por tanto, no se tienen en cuenta las conexiones eléctricas entre módulos.

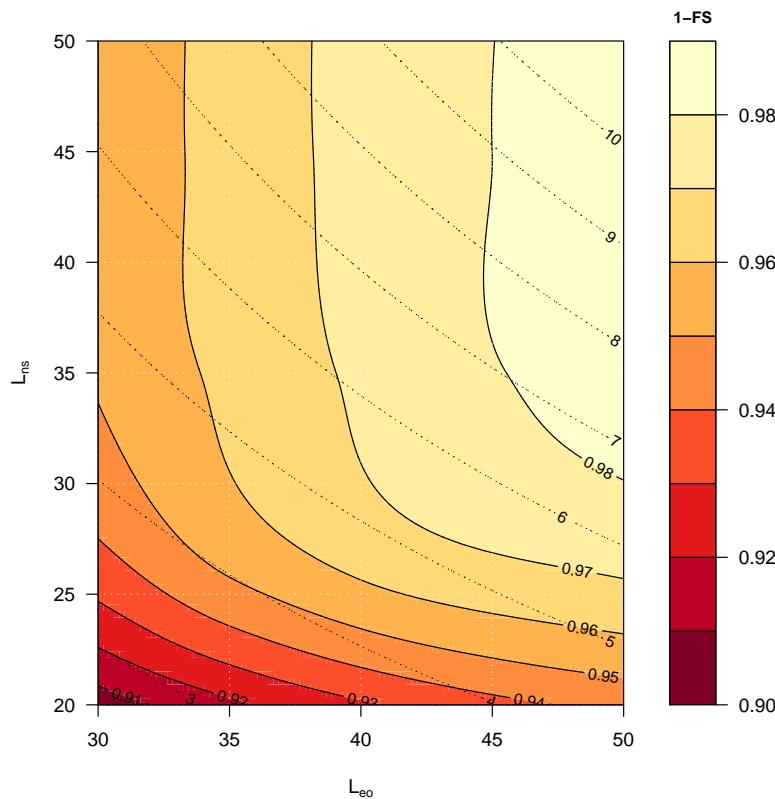


FIGURA 3.11: *Ábaco para planta de seguimiento a doble eje. Recoge el ratio entre la energía anual producida por un seguidor afectado por sombras mutuas (E_{acS}) y la producida por un seguidor sin sombreado (E_{ac0}). Las curvas de color negro representan la fracción de energía no afectada por sombras. Las curvas de puntos representan el valor del ROT. Figura 6.14 del libro ESF [Per23].*

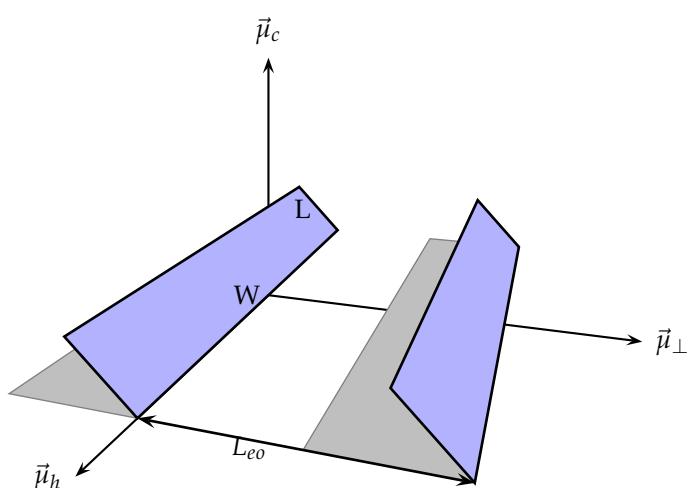


FIGURA 3.12: *Dimensiones básicas en sistemas con seguidores de eje horizontal. Figura 6.16 del libro ESF [Per23].*

Desarrollo del código

En la figura 4.1, se muestra el proceso de cálculo que sigue el paquete a la hora de obtener la estimación de la producción del sistema fotovoltaico. A la hora de estimar la producción, el

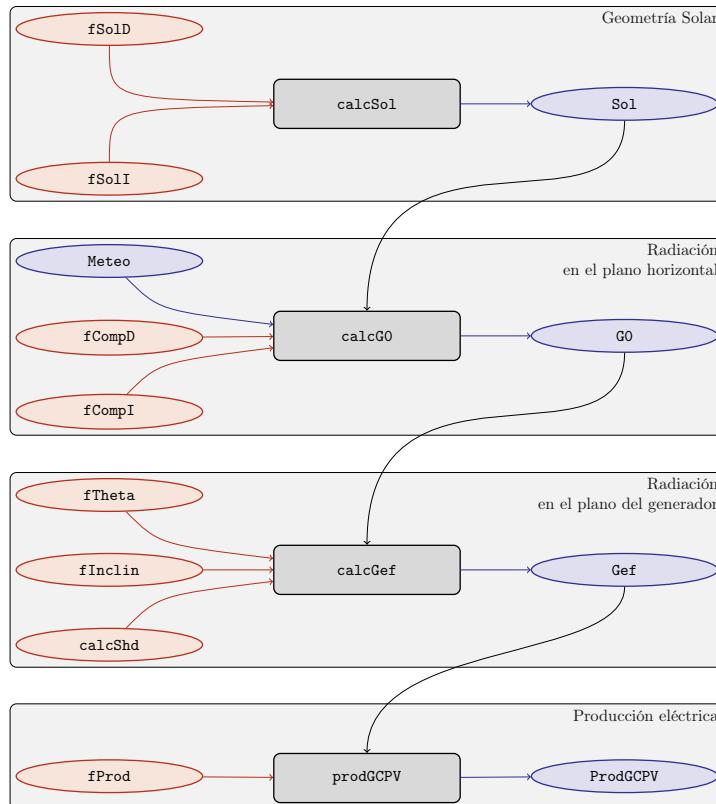


FIGURA 4.1: Proceso de cálculo de las funciones de **solar2**

programa sigue los siguientes procesos¹:

¹Todas las funciones recogidas en este capítulo están descritas en el manual de uso del paquete **solar2**, el cual, está disponible en el apéndice A de este documento.

4.1. Geometría solar

Para calcular la geometría que definen las posiciones de la Tierra y el Sol, **solar2** se vale de una función constructora, **calcSol**, la cual calcula mediante las funciones **fSolD** y **fSolI** todos los ángulos y componentes que caracterizan la geometría solar.

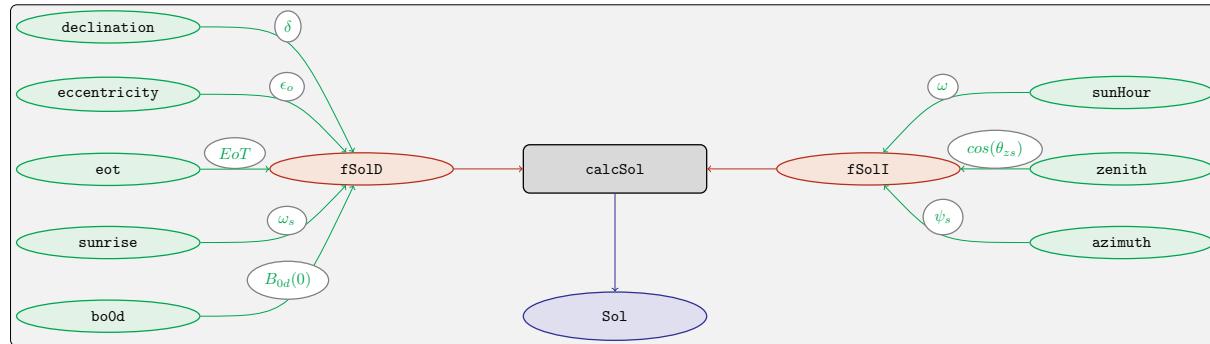


FIGURA 4.2: *Cálculo de la geometría solar mediante la función **calcSol**, la cual unifica las funciones **fSolD** y **fSolI** resultando en un objeto clase **Sol** el cual contiene toda la información geométrica necesaria para realizar las siguientes estimaciones.*

Como se puede ver en la figura 4.2, **calcSol** funciona gracias a las siguientes funciones:

- **fSolD**: la cual, a partir de la latitud (ϕ), calcula la geometría a nivel diario, es decir, los ángulos y componentes que se pueden calcular en cada día independiente. Estas son:
 - Declinación (δ): calculada a partir de la función **declination**.
 - Excentricidad (e_o): obtenida mediante la función **eccentricity**.
 - Ecuación del tiempo (EoT): obtenida mediante la función **eot**.
 - Ángulo del amanecer (ω_s): calculada a partir de la función **sunrise**.
 - Irradiancia diaria extra-atmosférica ($B_{0d}(0)$): obtenida a partir de la función **bo0d**.

```

1 lat <- 37.2
2 BTd <- fBTd(mode = 'prom')
3 solD <- fSolD(lat = lat, BTd = BTd)
4 show(solD)
  
```

Key: <Dates>							
	Dates	lat	decl	eo	EoT	ws	Bo0d
1:	2024-01-17	37.2	-0.36271754	1.0340422	-0.0455346238	-1.278593	4738.993
2:	2024-02-14	37.2	-0.22850166	1.0259717	-0.0614793356	-1.393341	6137.388
3:	2024-03-15	37.2	-0.03191616	1.0107943	-0.0368674274	-1.546560	8086.323
4:	2024-04-15	37.2	0.17531794	0.9926547	0.0017482721	-1.705659	9921.357
5:	2024-05-15	37.2	0.33246485	0.9775162	0.0143055938	-1.835976	11115.619
6:	2024-06-10	37.2	0.40257826	0.9691480	-0.0007378952	-1.899934	11573.907
7:	2024-07-18	37.2	0.36439367	0.9675489	-0.0263454380	-1.864521	11257.133
8:	2024-08-18	37.2	0.22407398	0.9758022	-0.0111761118	-1.744657	10183.208
9:	2024-09-18	37.2	0.02730595	0.9907919	0.0342189964	-1.591529	8508.642
10:	2024-10-19	37.2	-0.17900474	1.0088406	0.0689613044	-1.433019	6554.218
11:	2024-11-18	37.2	-0.33862399	1.0245012	0.0575423573	-1.300179	4951.750
12:	2024-12-13	37.2	-0.40478283	1.0328516	0.0158622941	-1.239567	4284.472

Tiene los siguientes argumentos:

- **lat**: para introducir la latitud en grados.
- **BTd**: para introducir la base temporal **diaria** sobre la que se harán los cálculos.
- **method**: para elegir el método de cálculo. Se puede elegir entre: **michalsky**, **cooper**, **spencer** y **strous**

```

1 BTd <- fBTd(mode = 'prom')
2 sold_michalsky <- fSolD(lat, BTd, method = 'michalsky')
3 sold_michalsky[, group := 'michalsky']
4 sold_cooper <- fSolD(lat, BTd, method = 'cooper')
5 sold_cooper[, group := 'cooper']
6 sold_spencer <- fSolD(lat, BTd, method = 'spencer')
7 sold_spencer[, group := 'spencer']
8 sold_strouss <- fSolD(lat, BTd, method = 'strous')
9 sold_strouss[, group := 'strous']
10 sold_all <- rbind(sold_michalsky, sold_cooper,
11                     sold_spencer, sold_strouss)
12 xyplot(eo + ws + Bo0d ~ Dates , sold_all,
13         groups = group, type = 'l', auto.key = TRUE,
14         par.settings = solaR.theme, scales = list(y = 'free'),
15         ylab = '', layout = c(1, 3))

```

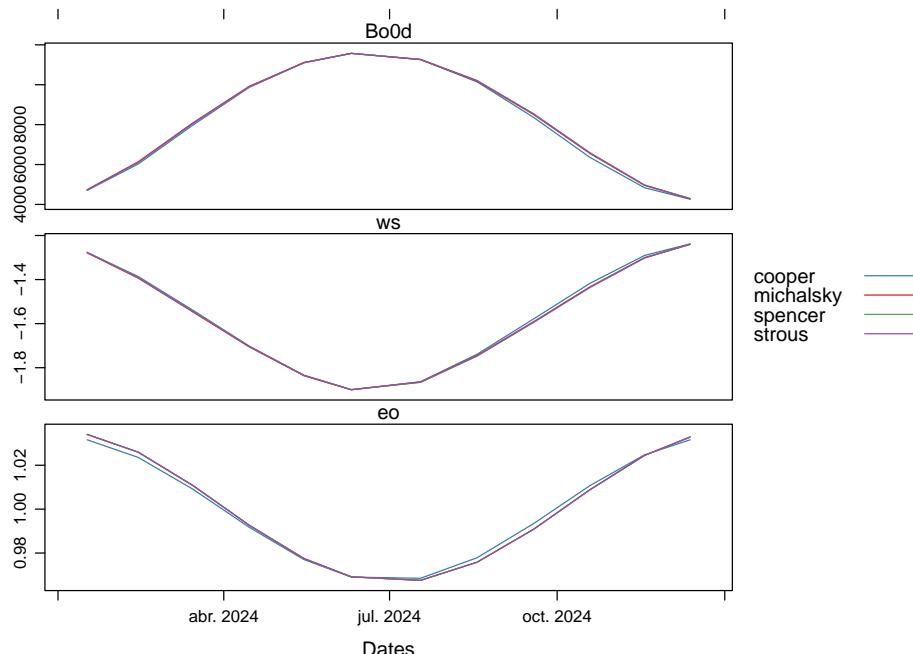


FIGURA 4.3: Comparación entre los diferentes métodos de cálculo de **fSolD**.

- **fSolI**: toma los resultados obtenidos en **fSolD** y calcula la geometría a nivel intradiario, es decir, aquella que se puede calcular en unidades de tiempo menores a los días. Estas son:

4. DESARROLLO DEL CÓDIGO

- La hora solar o tiempo solar verdadero (ω): calculada a partir de la función **sunHour**.
- Los momentos del día en los que es de noche (*night*): calculada a partir del resultado anterior y de el ángulo del amanecer (calculada en **fSolD**)².
- El coseno del ángulo cenital solar ($\cos(\theta_{zs})$): obtenida a partir de la función **zenith**.
- La altura solar (γ_s): obtenida a partir del resultado anterior³.
- El ángulo acimutal solar (θ_{zs}): calculada mediante la función **azimuth**.
- La irradiancia extra-atmosférica ($B_0(0)$): calculada mediante el coseno del ángulo cenital, la constante solar (B_0) y la excentridad (calculada en **fSolD**) [ecuación 3.6].

Su argumento principal es **sample** con el cual se puede determinar el intervalo intradiario de cálculos.

```
1 sold <- solD[1] #Computo solo un día a fin mejorar la visualización
2 solI <- fSolI(sold = sold, sample = 'min')
3 xyplot(solI)
```

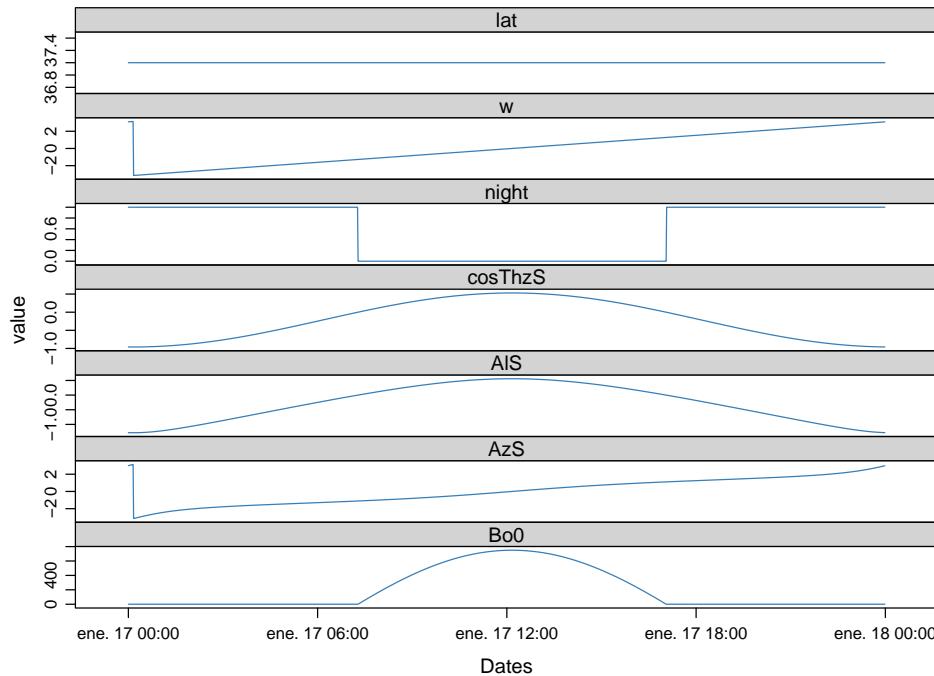


FIGURA 4.4: Representación gráfica de los resultados de la función **fSolI**.

Además, como los datos nocturnos aportan poco a los cálculos que atañen a este proyecto, **fSolI** presenta la posibilidad de eliminar estos datos con el argumento **keep.night**.

```
1 solI_nighth <- fSolI(sold = sold, sample = 'min', keep.night = FALSE)
2 xyplot(solI_nighth)
```

²Cuando la hora solar verdadera excede los ángulos en los que amanece y anochece ($|\omega| \geq |\omega_s|$), el Sol queda por debajo de la línea del horizonte, por lo que es de noche.

³ $\gamma_s = \arcsin(\cos(\theta_s))$.

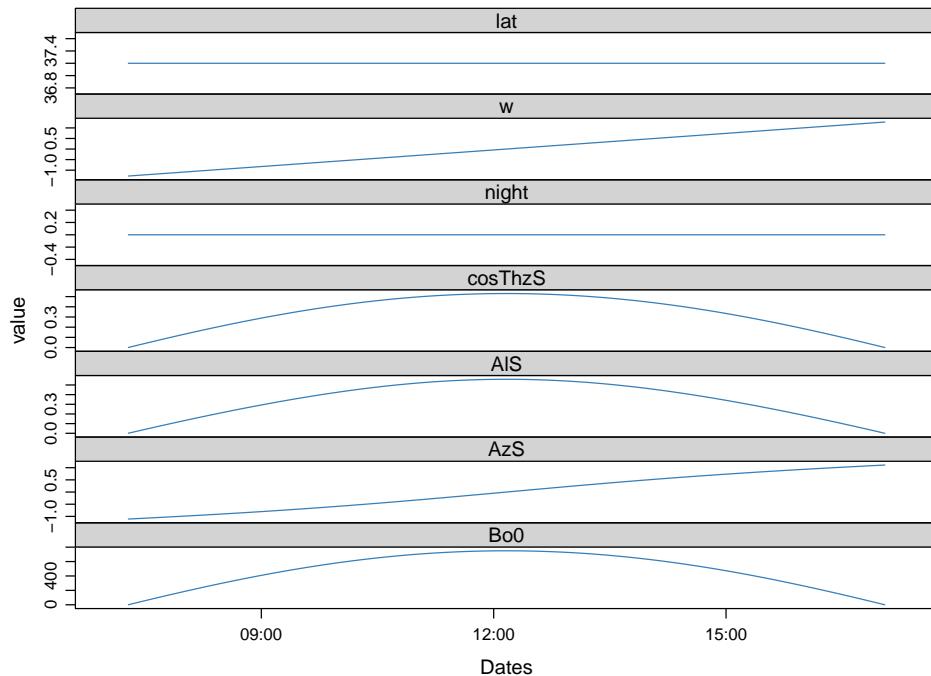


FIGURA 4.5: Representación gráfica de los resultados de la función `fSolI` una vez eliminados los valores nocturnos utilizando el argumento `keep.night`.

Aparte, en vez de identificar el intervalo intradiario (con el argumento `sample`), se puede dar directamente la base temporal intradiaria.

```

1 BTd <- sold$Dates
2 BTi <- fBTi(BTd, sample = 'min')
3 solI_BTi <- fSolI(sold, BTi = BTi, keep.night = FALSE)
4 show(solI_BTi)
```

	Dates	lat	w	night	cosThzS	Als	AzS	Bo0
	<POSct>	<num>	<num>	<lgcl>	<num>	<num>	<num>	<num>
1:	2024-01-17 07:17:00	37.2	-1.277809	FALSE	5.589721e-04	5.589722e-04	-1.108643	0.79012709
2:	2024-01-17 07:18:00	37.2	-1.273447	FALSE	3.667107e-03	3.667115e-03	-1.106002	5.18358628
3:	2024-01-17 07:19:00	37.2	-1.269085	FALSE	6.771089e-03	6.771141e-03	-1.103356	9.57117640
4:	2024-01-17 07:20:00	37.2	-1.264722	FALSE	9.870860e-03	9.871021e-03	-1.100706	13.95281396
5:	2024-01-17 07:21:00	37.2	-1.260360	FALSE	1.296636e-02	1.296673e-02	-1.098050	18.32841558

583:	2024-01-17 16:59:00	37.2	1.261058	FALSE	1.247170e-02	1.247202e-02	1.098475	17.62919226
584:	2024-01-17 17:00:00	37.2	1.265420	FALSE	9.375501e-03	9.375638e-03	1.101130	13.25260566
585:	2024-01-17 17:01:00	37.2	1.269782	FALSE	6.275042e-03	6.275083e-03	1.103780	8.86999635
586:	2024-01-17 17:02:00	37.2	1.274144	FALSE	3.170382e-03	3.170387e-03	1.106425	4.48144774
587:	2024-01-17 17:03:00	37.2	1.278507	FALSE	6.157845e-05	6.157845e-05	1.109064	0.08704333

También, se puede indicar que no realice las correcciones de la ecuación del tiempo.

```

1 solI_EoT <- fSolI(sold = sold, BTi = BTi, EoT = FALSE)
2 xyplot(solI_EoT)
```

4. DESARROLLO DEL CÓDIGO

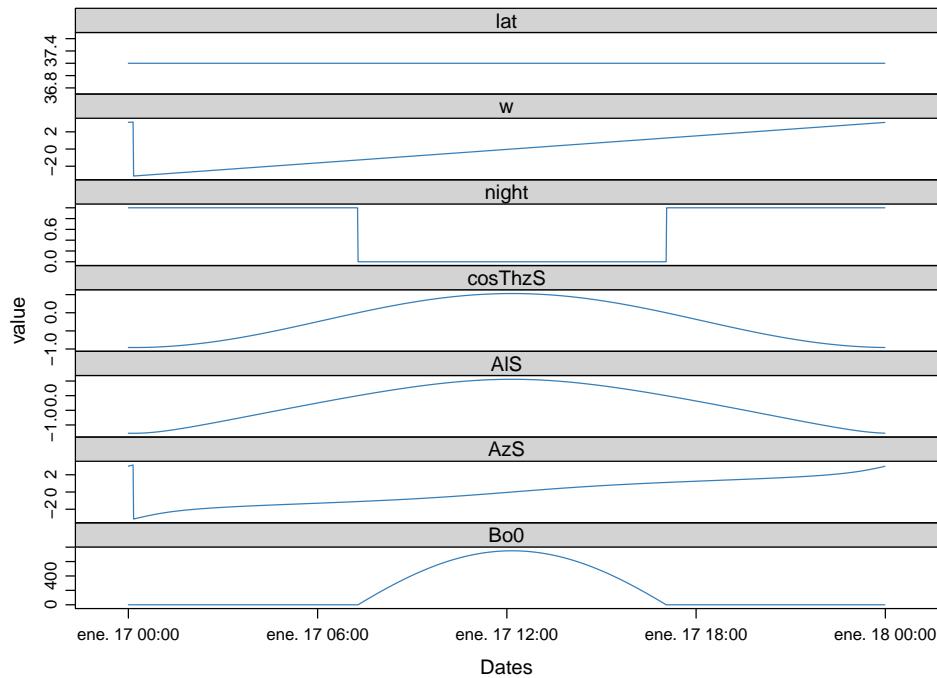


FIGURA 4.6: Representación gráfica de los resultados de la función `fSolI` sin realizar la corrección por la ecuación del tiempo.

Finalmente, estas dos funciones, como se muestra en la figura 4.2, convergen en la función `calcSol`, dando como resultado un objeto de clase `Sol`. Este objeto muestra un resumen de ambos elementos junto con la latitud de los cálculos. Los argumentos de `calcSol` son las argumentos de las funciones anteriores

```
1 sol <- calcSol(lat = lat, BTd = BTd, sample = 'hour')
2 show(sol)
```

```
Object of class Sol

Latitude: 37.2 degrees

Daily values:
  Dates      decl        eo       EoT        ws
Min. :2024-01-17 Min. :-0.3627 Min. :1.034 Min. :-0.04553 Min. :-1.279
1st Qu.:2024-01-17 1st Qu.:-0.3627 1st Qu.:1.034 1st Qu.:-0.04553 1st Qu.:-1.279
Median :2024-01-17 Median :-0.3627 Median :1.034 Median :-0.04553 Median :-1.279
Mean   :2024-01-17 Mean   :-0.3627 Mean   :1.034 Mean   :-0.04553 Mean   :-1.279
3rd Qu.:2024-01-17 3rd Qu.:-0.3627 3rd Qu.:1.034 3rd Qu.:-0.04553 3rd Qu.:-1.279
Max.   :2024-01-17 Max.  :-0.3627 Max.  :1.034 Max.  :-0.04553 Max.  :-1.279
  Bo0d
Min. :4739
1st Qu.:4739
Median :4739
Mean   :4739
3rd Qu.:4739
Max.   :4739

Intradaily values:
  Dates         w      night      cosThzS       AIS
Min. :2024-01-17 00:00:00 Min. :-2.92240 Mode :logical Min. :-0.9586 Min. :-1.2819
1st Qu.:2024-01-17 05:45:00 1st Qu.:-1.41740 FALSE:10    1st Qu.:-0.7289 1st Qu.:-0.8172
```

```

Median :2024-01-17 11:30:00 Median : 0.08759 TRUE :14 Median :-0.2143 Median :-0.2160
Mean   :2024-01-17 11:30:00 Mean   : 0.08766 Mean   :-0.2144 Mean   :-0.2724
3rd Qu.:2024-01-17 17:15:00 3rd Qu.: 1.59259 3rd Qu.: 0.3003 3rd Qu.: 0.3051
Max.   :2024-01-17 23:00:00 Max.   : 3.09905 Max.   : 0.5295 Max.   : 0.5580
      AzS          Bo0
Min.  :-2.49463 Min.   : 0.0
1st Qu.:-1.18986 1st Qu.: 0.0
Median : 0.09526 Median : 0.0
Mean   : 0.08773 Mean   :197.0
3rd Qu.: 1.28896 3rd Qu.:424.5
Max.   : 3.00158 Max.   :748.4

```

4.2. Datos meteorológicos

Para el procesamiento de datos meteorológicos, **solar2** provee una serie de funciones que son capaces de leer todo tipo de datos. Estos datos se procesan y se almacenan en un objeto de tipo **Meteo** tal y como se ve en la figura 4.7. Estas funciones son:

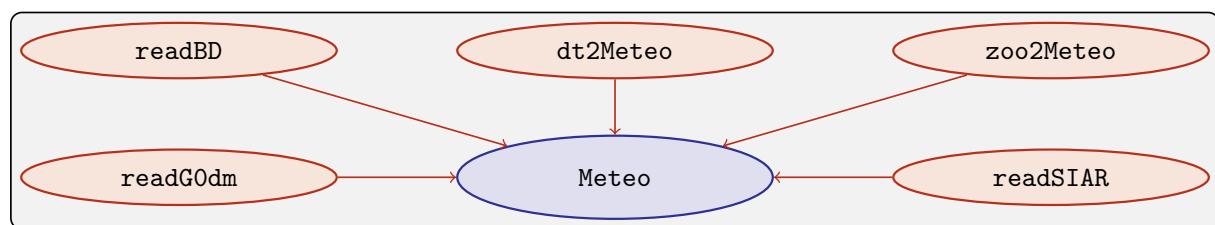


FIGURA 4.7: Los datos meteorológicas se pueden leer mediante las funciones **readG0dm**, **readBD**, **dt2Meteo**, **zoo2Meteo** y **readSIAR** las cuales procesan estos datos y los almacenan en un objeto de clase **Meteo**.

- **readG0dm**: Esta función construye un objeto **Meteo** a partir de 12 valores de medias mensuales de irradiación. Como argumentos tiene:

- **G0dm**: vector con medias mensuales de irradiación global horizontal.
- **Ta**: vector con la temperatura ambiente.
- **lat**: latitud en grados.
- **year**: año de los datos. Por defecto, presenta el año actual.
- **source**: información de la fuente de los datos.

```

1 G0dm <- c(2.766,3.491,4.494,5.912,6.989,7.742,
2           7.919,7.027,5.369,3.562,2.814,2.179) * 1000;
3 Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2,
4         28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
5 BD <- readG0dm(G0dm = G0dm, Ta = Ta, lat = 37.2)
6 xyplot(BD)

```

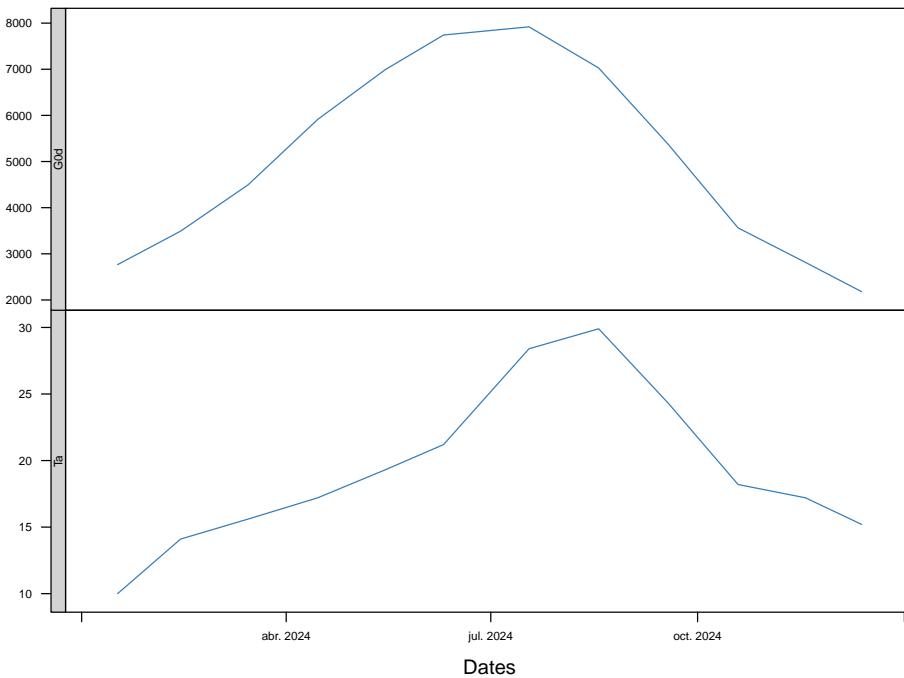


FIGURA 4.8: Representación gráfica de un objeto **Meteo** producido con medias mensuales de datos diarios.

- **readBD:** Esta familia de funciones puede leer ficheros de datos y transformarlos en un objeto de clase **Meteo**. Se dividen en:

- **readBDd:** Procesa datos meteorológicos de tipo diarios. Como argumentos tiene:
 - **file:** nombre del archivo que contiene los datos.
 - **lat:** latitud en grados.
 - **format:** formato de los datos de fechas.
 - **header, fill, dec, sep:** argumentos para **read**.
 - **dates.col:** nombre de la columna que contiene los datos de fechas. Por defecto, es 'Dates'.
 - **ta.col:** nombre de la columna que contiene los datos de temperatura. Por defecto, es 'Ta'.
 - **g0.col:** nombre de la columna que contiene los datos de irradiación. Por defecto, es 'G0'.
 - **keep.cols:** si su valor es **TRUE**, mantiene las columnas que no sean importantes para el resto de operaciones.

```
1 ## Se utiliza un archivo alojado en el
2 ## github del tutor de este proyecto
3 myURL <- "https://raw.githubusercontent.com/oscarperpinan/R/master/data/
4     aranjuez.csv"
5 download.file(myURL, 'data/aranjuez.csv', quiet = TRUE)
6 BDd <- readBDd(file = 'data/aranjuez.csv', lat = lat,
7                 format = '%Y-%m-%d', header = TRUE,
8                 fill = TRUE, dec = '.', sep = ',', dates.col = '',
9                 ta.col = 'TempAvg', g0.col = 'Radiation', keep.cols = TRUE)
10 xyplot(BDd)
```

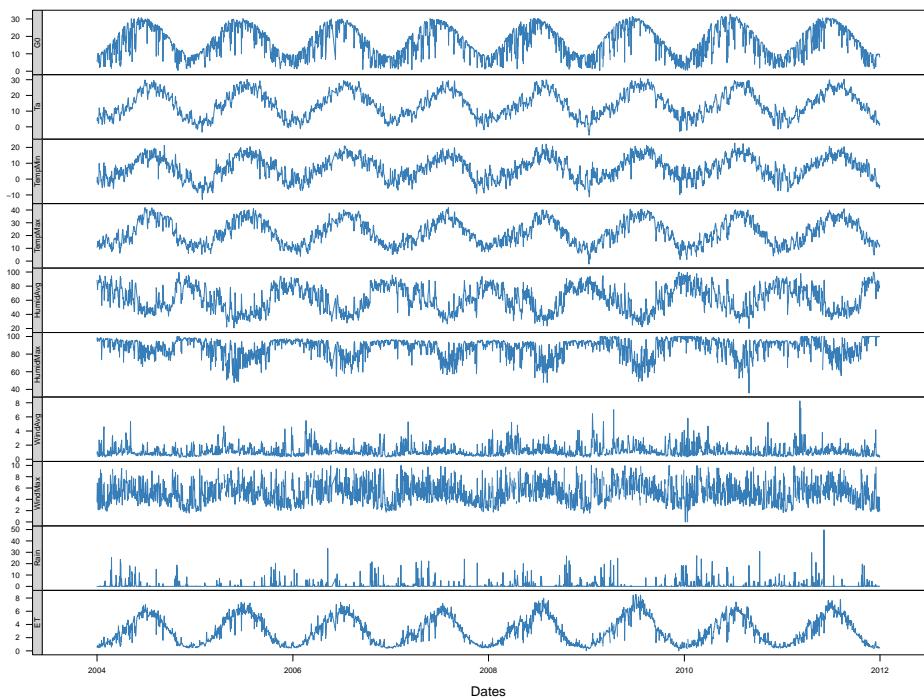


FIGURA 4.9: Representación gráfica de un objeto **Meteo** producido con datos diarios procedentes de un archivo.

- **readBDi**: procesa datos meteorológicos de tipo intradiarios. Como argumentos tiene:
 - **file**: nombre del archivo que contiene los datos.
 - **lat**: latitud en grados.
 - **format**: formato de los datos de fechas.
 - **header**, **fill**, **dec**, **sep**: argumentos para **fread**.
 - **dates.col**: nombre de la columna que contiene los datos de fechas y/o tiempos. Por defecto, es 'Dates'.
 - **times.col**: nombre de la columna que contiene los datos de tiempos en caso de que **dates.col** no los incluyera.
 - **ta.col**: nombre de la columna que contiene los datos de temperatura. Por defecto, es 'Ta'.
 - **g0.col**: nombre de la columna que contiene los datos de irradiancia. Por defecto, es 'G0'.
 - **keep.cols**: si su valor es **TRUE**, mantiene las columnas que no sean importantes para el resto de operaciones.

```

1 myURL <- "https://raw.githubusercontent.com/oscarperpinan/R/master/data/
2   NREL-Hawaii.csv"
3 download.file(myURL, 'data/NREL-Hawaii.csv', quiet = TRUE)
4 BDi <- readBDi(file = 'data/NREL-Hawaii.csv', lat = 19,
5   format = "%m/%d/%Y %H:%M", header = TRUE,
6   fill = TRUE, dec = '.', sep = ',',
7   dates.col = 'DATE', times.col = 'HST',
8   ta.col = 'Air Temperature [deg C]',
9   g0.col = 'Global Horizontal [W/m^2]',
```

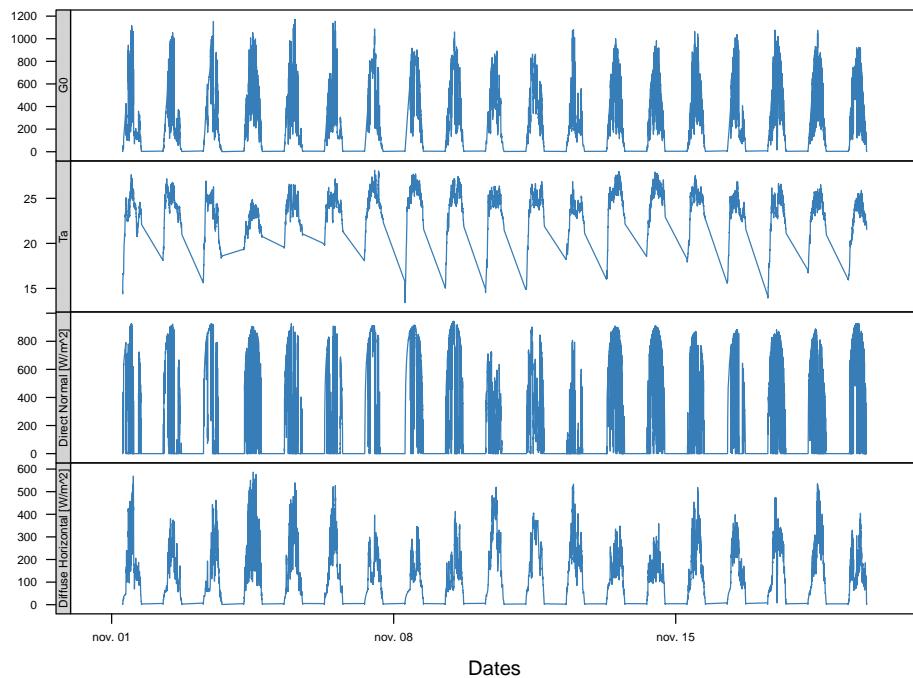
10 | **xypplot**(BDi)

FIGURA 4.10: Representación gráfica de un objeto **Meteo** producido con datos intradiarios procedentes de un archivo.

- **dt2Meteo**: transforma un **data.table** o **data.frame** en un objeto de clase **Meteo**. Como argumentos tiene:

- **file**: **data.table** que contiene los datos.
- **lat**: latitud en grados.
- **source**: información sobre la fuente de los datos.
- **type**: tipo de datos. A elegir entre **bdI** (intradiarios), **bd** (diarios) y **prom** (medias mensuales). Si no viene dado, lo calcula por su cuenta.

```

1 data(helios)
2 names(helios) <- c('Dates', 'G0d', 'TempMax', 'TempMin')
3 helios_meteo <- dt2Meteo(file = helios, lat = 40, type = 'bd')
4 xypplot(helios_meteo)

```

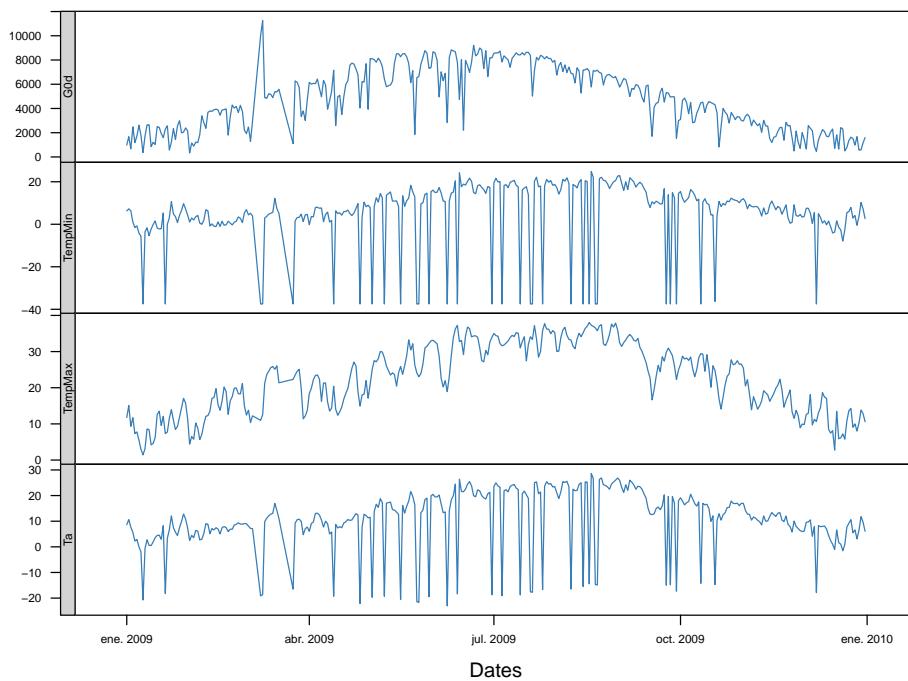


FIGURA 4.11: Representación gráfica de un objeto **Meteo** producido con datos almacenados en **data.table**.

- **zoo2Meteo**: transforma un objeto de clase **zoo**⁴ en un objeto de clase **Meteo**. Como argumentos tiene:
 - **file**: **data.table** que contiene los datos.
 - **lat**: latitud en grados.
 - **source**: información sobre la fuente de los datos.

```

1 library(zoo)
2 bd_zoo <- read.csv.zoo('data/aranjuez.csv')
3 BD_zoo <- zoo2Meteo(file = bd_zoo, lat = 40)
4 show(BD_zoo)

```

```

Object of class Meteo

Source of meteorological information: bd-zoo-bd_zoo
Latitude of source: 40 degrees

Meteorological Data:
  TempAvg     TempMax     TempMin     HumidAvg     HumidMax     WindAvg
  Min.   : -5.309   Min.   : -2.362   Min.   : -12.980   Min.   : 19.89   Min.   : 35.88   Min.   : 0.251
  1st Qu.:  7.692   1st Qu.: 14.530   1st Qu.:  1.515   1st Qu.: 47.04   1st Qu.: 81.60   1st Qu.: 0.667
  Median : 13.810   Median : 21.670   Median :  7.170   Median : 62.58   Median : 90.90   Median : 0.920
  Mean   : 14.405   Mean   : 22.531   Mean   :  6.888   Mean   : 62.16   Mean   : 87.22   Mean   : 1.174
  3rd Qu.: 21.615   3rd Qu.: 30.875   3rd Qu.: 12.590   3rd Qu.: 77.38   3rd Qu.: 94.90   3rd Qu.: 1.431
  Max.   : 30.680   Max.   : 41.910   Max.   : 22.710   Max.   :100.00   Max.   :100.00   Max.   : 8.260
  NA's    : 4          NA's    : 4          NA's    : 4          NA's    : 13        NA's    : 8

```

⁴Pese a que este proyecto trate de “desligarse” del paquete **zoo**, sigue siendo un paquete muy extendido. Por ello, es interesante tener una función así para que los usuarios tengan una mayor flexibilidad.

4. DESARROLLO DEL CÓDIGO

WindMax	Rain	Radiation	ET
Min. : 0.000	Min. : 0.000	Min. : 0.277	Min. : 0.000
1st Qu.: 3.783	1st Qu.: 0.000	1st Qu.: 9.370	1st Qu.: 1.168
Median : 5.027	Median : 0.000	Median : 16.660	Median : 2.758
Mean : 5.208	Mean : 1.094	Mean : 16.742	Mean : 3.091
3rd Qu.: 6.537	3rd Qu.: 0.200	3rd Qu.: 24.650	3rd Qu.: 4.926
Max. : 10.000	Max. : 49.730	Max. : 32.740	Max. : 8.564
NA's : 128	NA's : 4	NA's : 13	NA's : 18

- **readSIAR**: esta función es capaz de extraer información de la red SIAR⁵ y transformarlo en un objeto de clase **Meteo**. Como argumentos tiene:

- **Lon**: longitud en grados.
- **Lat**: latitud en grados.
- **inicio**: primer día de los registros.
- **final**: último día de los registros.
- **tipo**: la API SIAR⁶ permite tener 4 tipos de registros: **Mensuales**, **Semanales**, **Diarios** y **Horarios**.
- **n_est**: con este argumento, la función es capaz de localizar el número seleccionado de estaciones más próximas a la ubicación dada, y obtener los datos individuales de cada una de ellas. Una vez obtenidos estos datos realiza una interpolación de distancia inversa ponderada (IDW⁷) y entrega un solo resultado. Es importante añadir que la API SIAR tiene una limitación a la solicitud de registros que se le hace cada minuto, por lo que esta función cuenta con un comprobante para impedir que el usuario exceda este límite.

```

1 library(httr2)
2 library(jsonlite)
3 bd_SIAR <- readSIAR(Lat = 40.40596822621351, Lon = -3.70038308516172,
4                         ## Ubicación de la Escuela Técnica Superior
5                         ## de Ingeniería y Diseño Industrial (ETSIDI)
6                         inicio = '2023-09-01', final = '2024-08-01',
7                         tipo = 'Mensuales', n_est = 3)
8 xyplot(bd_SIAR)
```

⁵La red SiAR (Sistema de Información Agroclimática para el Regadio) es una infraestructura que captura, registra y divulga los datos climáticos necesarios para el cálculo de la demanda hídrica en las zonas de riego [Min23].

⁶La API (Interfaz de Programación de Aplicaciones) que se usa para la función **readSIAR** está proporcionada por la propia red SIAR [Min23].

⁷La interpolación IDW es un método de interpolación que estima el valor de un punto desconocido basado en los valores conocidos de puntos cercanos. Los puntos más cercanos tienen más peso en la estimación que los más lejanos, utilizando una relación inversa con la distancia.

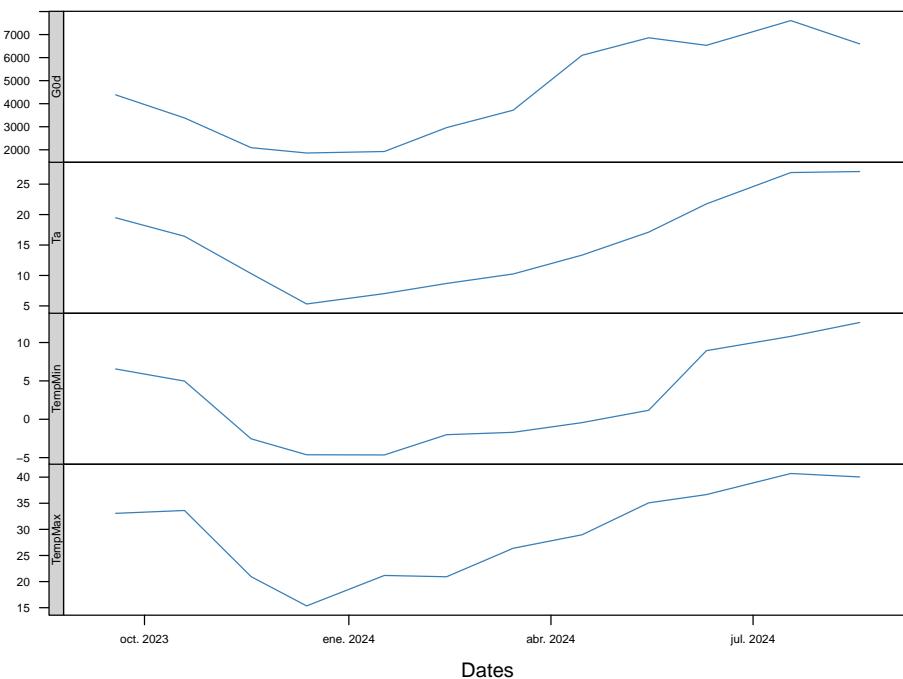


FIGURA 4.12: Representación gráfica de un objeto **Meteo** producido por la función **readSIAR**.

4.3. Radiación en el plano horizontal

Una vez se ha calculado la geometría solar (sección 4.1) y se han procesado los datos meteorológicos (sección 4.2), es necesario calcular la radiación en el plano horizontal. Para ello, **solar2** cuenta con la función **calcG0** la cual mediante las funciones **fCompD** y **fCompI** procesan los objetos de clase **Sol** y clase **Meteo** para dar un objeto de tipo **G0**.

Como se puede ver en la figura 4.13, **calcG0** funciona gracias a las siguientes funciones:

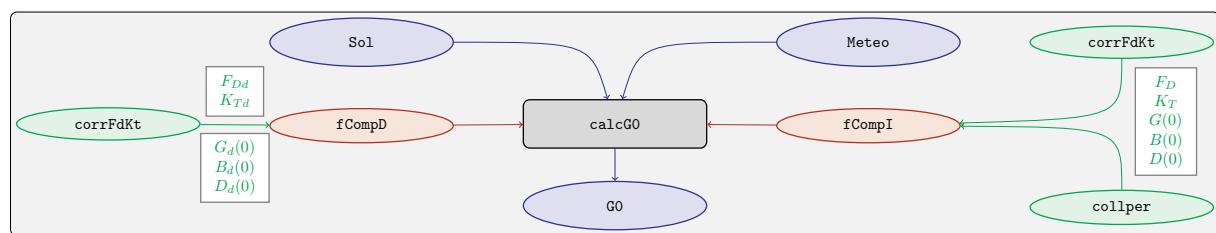


FIGURA 4.13: Cálculo de la radiación incidente en el plano horizontal mediante la función **calcG0**, la cual procesa un objeto clase **Sol** y otro clase **Meteo** mediante las funciones **fCompD** y **fCompI** resultando en un objeto clase **G0** :

- **fCompD**: la cual calcula todas las componentes de la radiación diaria en una superficie horizontal mediante regresiones entre los parámetros del índice de claridad y la fracción difusa. Tiene los siguientes argumentos:

- **sol**: un objeto de clase **Sol**.

4. DESARROLLO DEL CÓDIGO

- **G0d**: un objeto clase **Meteo** o un **data.table** con datos de irradiación diaria en una superficie horizontal.
- **corr**: a elegir el tipo de correlación entre la fracción de difusa y el índice de claridad. Dependiendo del tipo de datos:
 - Mensuales:

```

1 lat <- 37.2
2 BTd <- fBTd(mode = 'prom')
3 solD <- fSolD(lat, BTd)
4 G0d <- c(2.766,3.491,4.494,5.912,6.989,7.742,
5           7.919,7.027,5.369,3.562,2.814,2.179) * 1000
6 compD_page <- fCompD(sol = sold, G0d = G0d, corr = "Page")
7 compD_page[, group := 'page']
8 compD_lj <- fCompD(sol = sold, G0d = G0d, corr = "LJ")
9 compD_lj[, group := 'lj']
10 compD_dia <- rbind(compD_page, compD_lj)
11 xyplot(Fd + D0d + B0d ~ Dates, compD_dia,
12         groups = group, type = 'l', auto.key = TRUE,
13         par.settings = solaR.theme, scales = list(y = 'free'),
14         ylab = ' ', layout = c(1, 3))
15

```

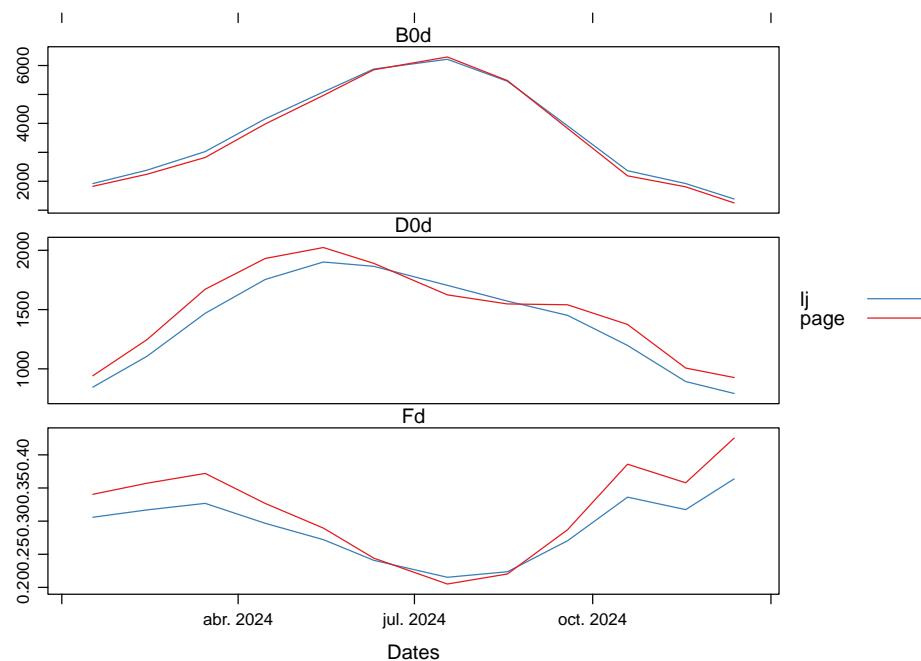


FIGURA 4.14: Comparación gráfica entre los resultados de la función *fCompD* cambiando entre tipos de correlación mensual.

- Diarios:

```

1 sol <- calcSol(lat, BTd = indexD(helios_meteo))
2 compD_cpr <- fCompD(sol = sol, G0d = helios_meteo,
3                       corr = "CPR")
4 compD_cpr[, group := 'cpr']

```

```

5 compD_ekdd <- fCompD(sol = sol, G0d = helios_meteo,
6                         corr = 'EKDD')
7 compD_ekdd[, group := 'ekdd']
8 compD_climeddd <- fCompD(sol = sol, G0d = helios_meteo,
9                           corr = 'CLIMEDD')
10 compD_climeddd[, group := 'climeddd']
11 compD_mes <- rbind(compD_cpr, compD_ekdd, compD_climeddd)
12 xyplot(Fd + D0d + B0d ~ Dates, compD_mes,
13         groups = group, type = 'l', auto.key = TRUE,
14         par.settings = solaR.theme, scales = list(y = 'free'),
15         ylab = '', layout = c(1, 3))

```

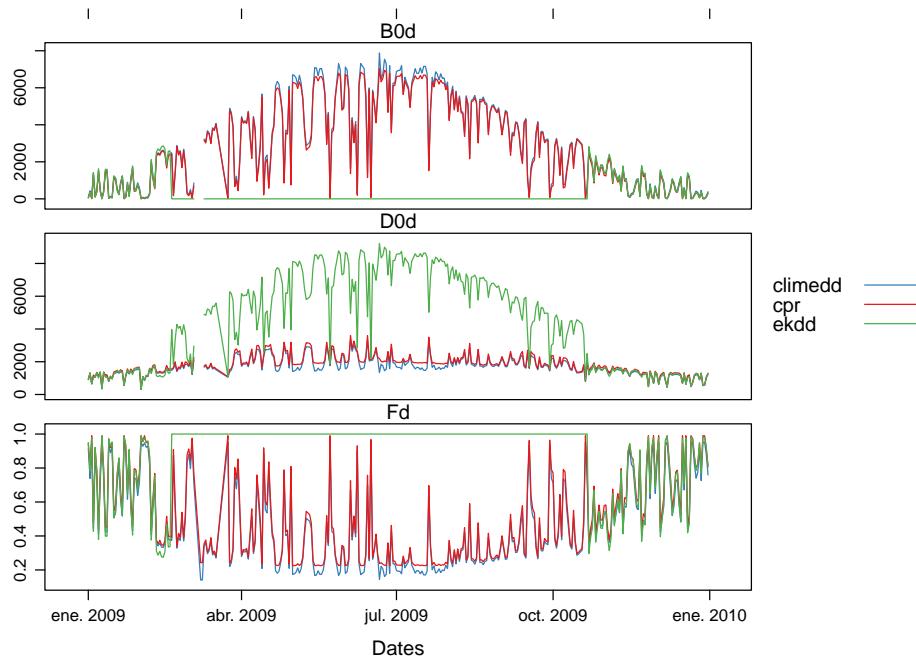


FIGURA 4.15: Comparación gráfica entre los resultados de la función `fCompD` cambiando entre tipos de correlación mensual.

También, se puede aportar una función de correlación propia con el argumento `f`.

```

1 f_corrd <- function(sol, G0d){
2   ## Función CLIMEDD
3   Kt <- Ktd(sol, G0d)
4   Fd=(Kt<=0.13)*(0.952)+#
5   (Kt>0.13 & Kt<=0.8)*(0.868+1.335*Kt-5.782*Kt^2+3.721*Kt^3)+#
6   (Kt>0.8)*0.141
7   return(data.table(Fd, Kt))
8 }
9 compD_user <- fCompD(sol = sol, G0d = helios_meteo,
10                      corr = 'user', f = f_corrd)
11 xyplot(compD_user)

```

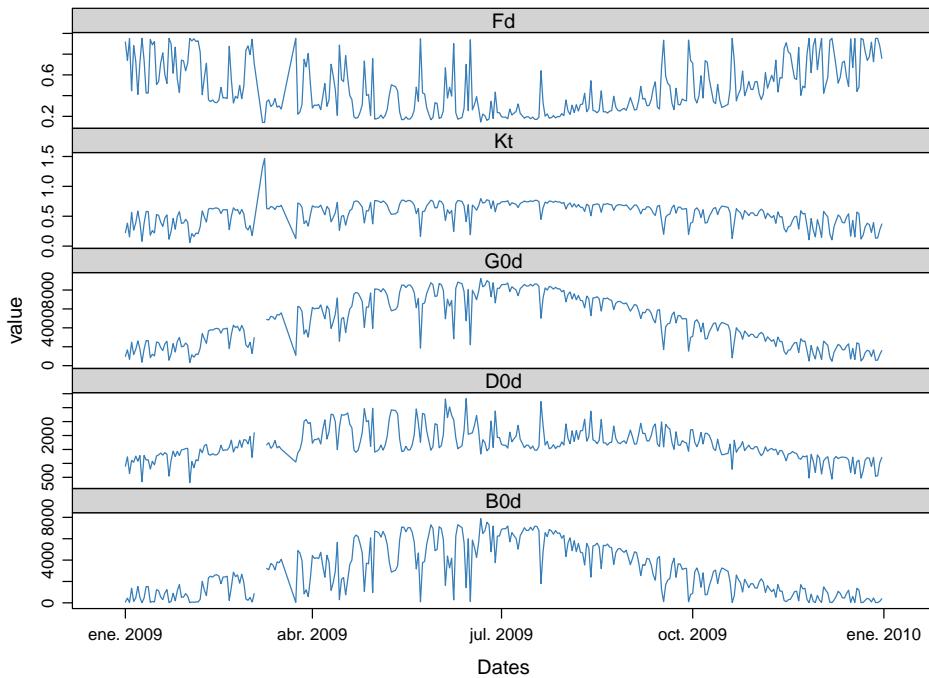


FIGURA 4.16: Representación gráfica de los resultados de la función `fCompD` tomando como función de correlación una propuesta por el usuario.

Por último, si `G0d` ya contiene todos los componentes, se puede especificar que no haga ninguna correlación.

```
1 compD_none <- fCompD(sol = sol, G0d = compD_user, corr = 'none')
2 compD_none
```

	Key: <Dates>					
	Dates	Fd	Kt	G0d	D0d	B0d
	<POS>	<num>	<num>	<num>	<num>	<num>
1:	2009-01-01	0.9173035	0.2260525	980.14	899.0859	81.05410
2:	2009-01-02	0.7386109	0.3840912	1671.80	1234.8097	436.99027
3:	2009-01-03	0.9501390	0.1535268	671.02	637.5622	33.45776
4:	2009-01-04	0.4467640	0.5655380	2482.80	1109.2257	1373.57434
5:	2009-01-05	0.8829779	0.2671047	1178.19	1040.3157	137.87428

351:	2009-12-27	0.7261768	0.3925374	1676.56	1217.4789	459.08109
352:	2009-12-28	0.9520296	0.1314429	562.64	535.6499	26.99008
353:	2009-12-29	0.9519370	0.1370913	588.30	560.0245	28.27547
354:	2009-12-30	0.8802137	0.2699681	1161.81	1022.6410	139.16897
355:	2009-12-31	0.7594870	0.3695667	1595.46	1211.7311	383.72890

- `fCompI`: calcula, en base a los valores de irradiación diaria, todas las componentes de irradiancia. Se vale de dos procedimientos en base al tipo de argumentos que toma:
 - `compD`: si recibe un `data.table` resultado de `fCompD`, calcula las relaciones entre las componentes de irradiancia e irradiación de las componentes de difusa y global, obteniendo con ellas un perfil de irradiancias [3.2] (las irradiancias global y difusa salen de estas relaciones, mientras que la directa surge por diferencia entre las dos).

```
1 ## Se recorta helios_meteo para mejorar la visualización
2 bdh <- helios_meteo[, as.Date('2009-01-03')]
```

```

3 | sol <- calcSol(lat = 37.2, BTd = indexD(bdh),
4 |                         sample = 'min', keep.night = FALSE)
5 | compD <- fCompD(sol = sol, G0d = bdh)
6 | compI <- fCompI(sol = sol, compD = compD)
7 | xyplot(compI)

```

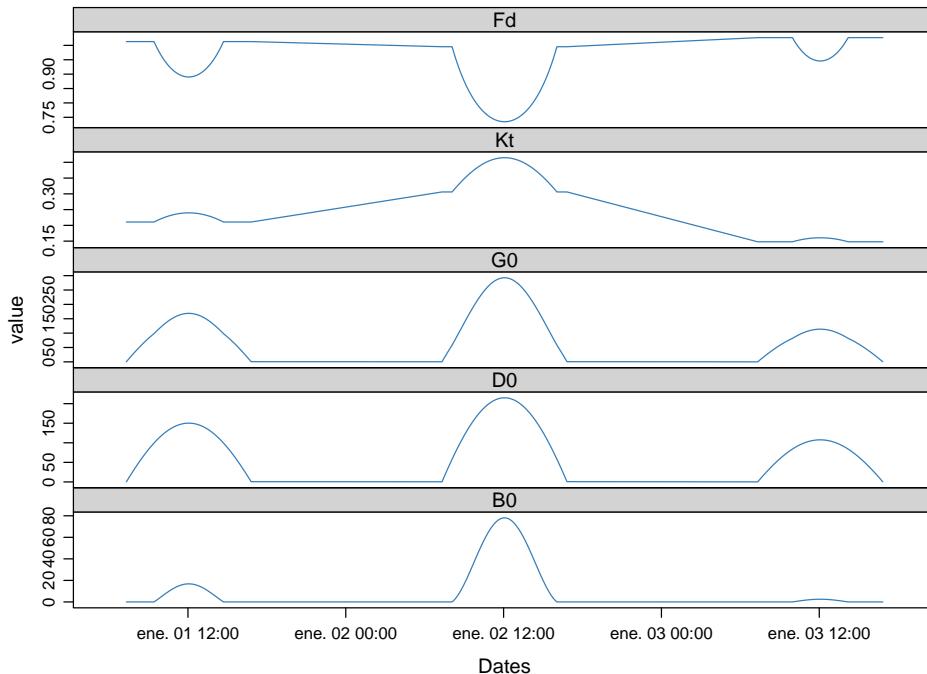


FIGURA 4.17: Representación gráfica de los resultados de la función *fCompI*.

- GOI: este argumento recibe datos de irradiancia, para después, poder aplicar las correcciones indicadas en el argumento **corr**.

```

1 GOI <- compI$G0
2 compI_ekdh <- fCompI(sol = sol, GOI = GOI, corr = 'EKDh')
3 compI_ekdh[, group := 'ekdh']
4 compI_brl <- fCompI(sol = sol, GOI = GOI, corr = 'BRL')
5 compI_brl[, group := 'brl']
6 compI_climedh <- fCompI(sol = sol, GOI = GOI, corr = 'CLIMEDh')
7 compI_climedh[, group := 'climedh']
8 compI_all <- rbind(compI_ekdh, compI_brl, compI_climedh)
9 xyplot(Fd + D0 + B0 ~ Dates, compI_all,
10        groups= group, type = 'l', auto.key= TRUE,
11        par.settings= solaR.theme, scales =list(y = 'free'),
12        ylab = '', layout= c(1,3))

```

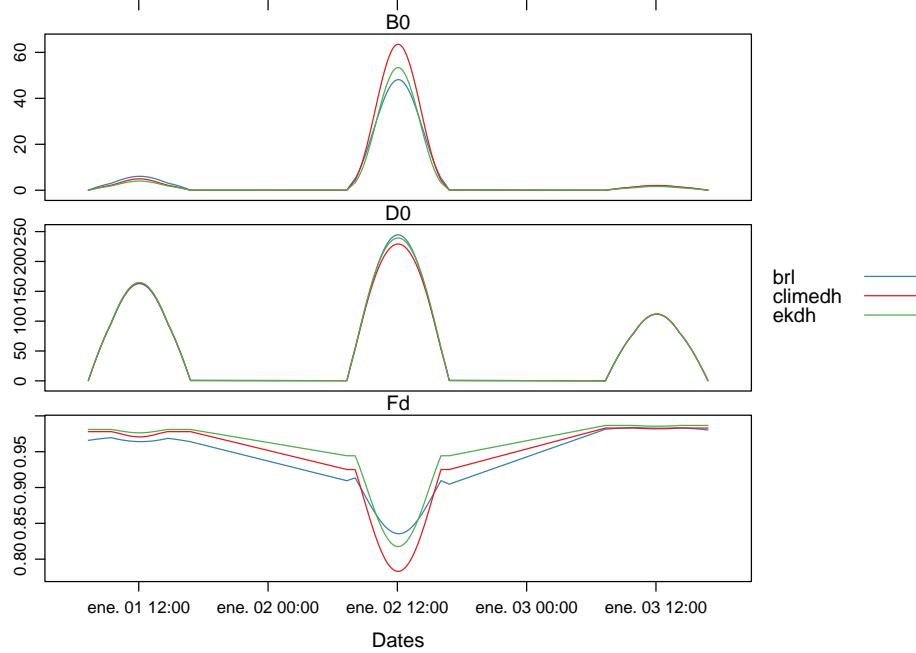


FIGURA 4.18: Comparación entre los resultados de la función `fCompI` cambiando entre diferentes tipos de correlación intradiaria.

Como con `fCompD`, se puede añadir una función correctora propia.

```

1 f_corri <- function(sol, GOi){
2   ## Función CLIMEDh
3   Kt <- Kti(sol, GOi)
4   Fd=(Kt<=0.21)*(0.995-0.081*Kt)+ 
5     (Kt>0.21 & Kt<=0.76)*(0.724+2.738*Kt-8.32*Kt^2+4.967*Kt^3)+ 
6     (Kt>0.76)*0.180
7   return(data.table(Fd, Kt))
8 }
9 compI_user <- fCompI(sol = sol, GOI = GOI, corr = 'user', f = f_corri)
10 show(compI_user)
```

	Key: <Dates>	Dates	Fd	Kt	G0	D0	B0
		<POS>	<num>	<num>	<num>	<num>	<num>
1:	2009-01-01 07:19:00	0.9780054	0.2106054	0.3399878	0.3325099	0.007477898	
2:	2009-01-01 07:20:00	0.9780054	0.2106054	1.2414206	1.2141161	0.027304559	
3:	2009-01-01 07:21:00	0.9780054	0.2106054	2.1414924	2.0943911	0.047101285	
4:	2009-01-01 07:22:00	0.9780054	0.2106054	3.0401860	2.9733183	0.066867698	
5:	2009-01-01 07:23:00	0.9780054	0.2106054	3.9374844	3.8508809	0.086603423	

1709:	2009-01-03 16:46:00	0.9830147	0.1479667	2.7493226	2.7026245	0.046698067	
1710:	2009-01-03 16:47:00	0.9830147	0.1479667	2.1172830	2.0813204	0.035962686	
1711:	2009-01-03 16:48:00	0.9830147	0.1479667	1.4842707	1.4590599	0.025210782	
1712:	2009-01-03 16:49:00	0.9830147	0.1479667	0.8502977	0.8358551	0.014442561	
1713:	2009-01-03 16:50:00	0.9830147	0.1479667	0.2153760	0.2117178	0.003658226	

Y además, se puede no añadir correlación.

```
1 GOI <- compI_user
```

```

2 | compI_none <- fCompI(sol = sol, GOI = GOI, corr = 'none')
3 | show(compI_none)

```

Key: <Dates>						
	Dates	Fd	Kt	G0	D0	B0
	<POSc>	<num>	<num>	<num>	<num>	<num>
1:	2009-01-01 07:19:00	0.9780054	0.2106054	0.3399878	0.3325099	0.007477898
2:	2009-01-01 07:20:00	0.9780054	0.2106054	1.2414206	1.2141161	0.027304559
3:	2009-01-01 07:21:00	0.9780054	0.2106054	2.1414924	2.0943911	0.047101285
4:	2009-01-01 07:22:00	0.9780054	0.2106054	3.0401860	2.9733183	0.066867698
5:	2009-01-01 07:23:00	0.9780054	0.2106054	3.9374844	3.8508809	0.086603423

1709:	2009-01-03 16:46:00	0.9830147	0.1479667	2.7493226	2.7026245	0.046698067
1710:	2009-01-03 16:47:00	0.9830147	0.1479667	2.1172830	2.0813204	0.035962686
1711:	2009-01-03 16:48:00	0.9830147	0.1479667	1.4842707	1.4590599	0.025210782
1712:	2009-01-03 16:49:00	0.9830147	0.1479667	0.8502977	0.8358551	0.014442561
1713:	2009-01-03 16:50:00	0.9830147	0.1479667	0.2153760	0.2117178	0.003658226

Por último, esta función incluye un argumento extra, **filterG0** que cuando su valor es **TRUE**, elimina todos aquellos valores de irradiancia que son mayores que la irradiancia extra-atmosférica (ya que es incoherente que la irradiancia terrestre sea mayor que la extra-terrestre).

Estas dos funciones, como se muestra en la figura 4.13, convergen en la función constructora **calcG0**, dando como resultado un objeto de clase **G0**. Este objeto muestra la media mensual de la irradiación diaria y la irradiación anual. Aparte, incluye los resultados de **fCompD** y **fCompI** y los objetos **Sol** y **Meteo** de los que parte.

Como argumento más importante está **modeRad**, el cual selecciona el tipo de datos que introduce el usuario en el argumento **dataRad**. Estos son:

- Medias mensuales.

```

1 | G0dm <- c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742, 7.919,
2 |           7.027, 5.369, 3.562, 2.814, 2.179) * 1000
3 | Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2,
4 |           28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
5 | prom <- data.table(G0dm, Ta)
6 | g0_prom <- calcG0(lat, modeRad = 'prom', dataRad = prom)
7 | show(g0_prom)

```

Object of class G0
Source of meteorological information: prom-
Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees
Monthly averages:
Dates G0d D0d B0d
<char> <num> <num> <num>
1: Jan. 2024 2.766 0.941698 1.824302
2: Feb. 2024 3.491 1.247146 2.243854
3: Mar. 2024 4.494 1.671763 2.822237
4: Apr. 2024 5.912 1.931146 3.980854
5: May. 2024 6.989 2.023364 4.965636
6: Jun. 2024 7.742 1.889994 5.852006
7: Jul. 2024 7.919 1.624064 6.294936
8: Aug. 2024 7.027 1.547591 5.479409
9: Sep. 2024 5.369 1.540708 3.828292

4. DESARROLLO DEL CÓDIGO

```
10: Oct. 2024 3.562 1.374513 2.187487
11: Nov. 2024 2.814 1.006959 1.807041
12: Dec. 2024 2.179 0.926737 1.252263

Yearly values:
  Dates      G0d      D0d      B0d
  <int>    <num>    <num>    <num>
1: 2024 1839.365 540.6331 1298.732
```

- Generación de secuencias diarias mediante matrices de transición de Markov.

```
1 g0_aguiar <- calcG0(lat, modeRad = 'aguiar', dataRad = prom)
2 show(g0_aguiar)
```

```
Object of class GO

Source of meteorological information: bd-aguiar

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates      G0d      D0d      B0d
  <char>    <num>    <num>    <num>
1: Jan. 2024 2.766 1.087710 1.678290
2: Feb. 2024 3.491 1.591853 1.899147
3: Mar. 2024 4.494 2.163886 2.330114
4: Apr. 2024 5.912 2.357474 3.554526
5: May. 2024 6.989 2.478828 4.510172
6: Jun. 2024 7.742 2.548190 5.193810
7: Jul. 2024 7.919 2.196037 5.722963
8: Aug. 2024 7.027 2.131409 4.895591
9: Sep. 2024 5.369 2.009201 3.359799
10: Oct. 2024 3.562 1.650990 1.911010
11: Nov. 2024 2.814 1.307506 1.506494
12: Dec. 2024 2.179 1.071548 1.107452

Yearly values:
Key: <Dates>
  Dates      G0d      D0d      B0d
  <int>    <num>    <num>    <num>
1: 2024 1839.365 689.0275 1150.338
```

- Diarios.

```
1 bd <- as.data.tableD(g0_aguiar)
2 g0_bd <- calcG0(lat, modeRad = 'bd', dataRad = bd)
3 show(g0_bd)
```

```
Object of class GO

Source of meteorological information: bd-data.table

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates      G0d      D0d      B0d
  <char>    <num>    <num>    <num>
1: Jan. 2024 2.766 1.087710 1.678290
2: Feb. 2024 3.491 1.591853 1.899147
3: Mar. 2024 4.494 2.163886 2.330114
4: Apr. 2024 5.912 2.357474 3.554526
5: May. 2024 6.989 2.478828 4.510172
6: Jun. 2024 7.742 2.548190 5.193810
```

```

7: Jul. 2024 7.919 2.196037 5.722963
8: Aug. 2024 7.027 2.131409 4.895591
9: Sep. 2024 5.369 2.009201 3.359799
10: Oct. 2024 3.562 1.650990 1.911010
11: Nov. 2024 2.814 1.307506 1.506494
12: Dec. 2024 2.179 1.071548 1.107452

Yearly values:
Key: <Dates>
      Dates      G0d      D0d      B0d
      <int>    <num>    <num>    <num>
1: 2024 1839.365 689.0275 1150.338

```

- Intradiarios

```

1 bdI <- as.data.tableI(g0_aguiar)
2 g0_bdI <- calcG0(lat, modeRad = 'bdI', dataRad = bdI)
3 xyplot(g0_bdI)

```

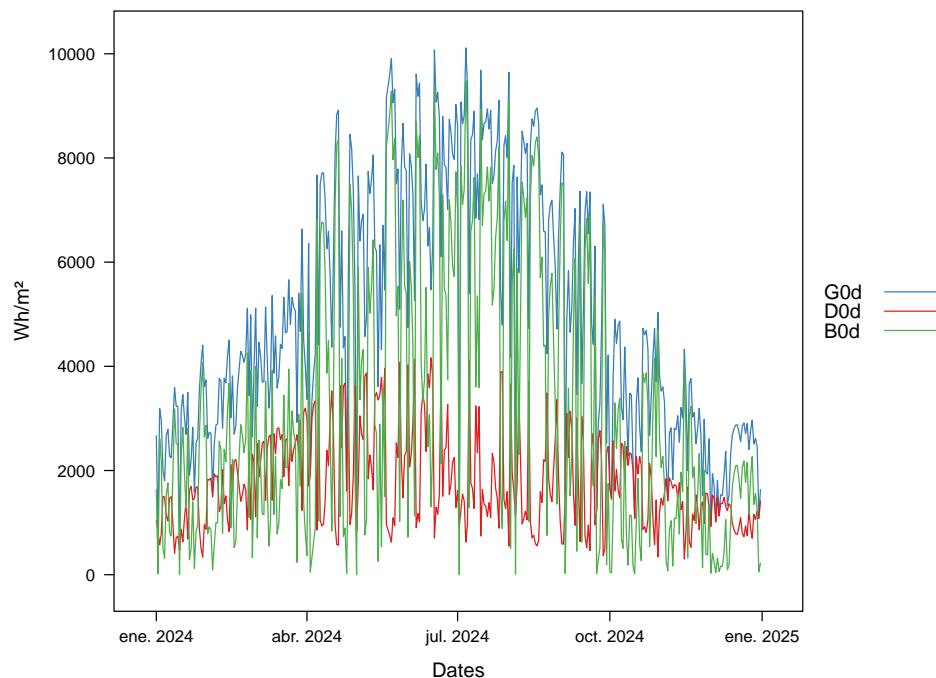


FIGURA 4.19: Representación gráfica de un objeto **G0** resultado de la función **calcG0**.

4.4. Radiación efectiva en el plano del generador

Teniendo la radiación incidente en plano horizontal (sección 4.3), se puede calcular la radiación efectiva incidente en el plano del generador. Para ello, **solaR2** cuenta con la función **calcGef** la cual mediante las funciones **fInclin** y **calcShd** procesa un objeto de clase **G0** para obtener un objeto **Gef**.

Como se puede ver en la figura 4.20, **calcGef** funciona gracias a las siguientes funciones:

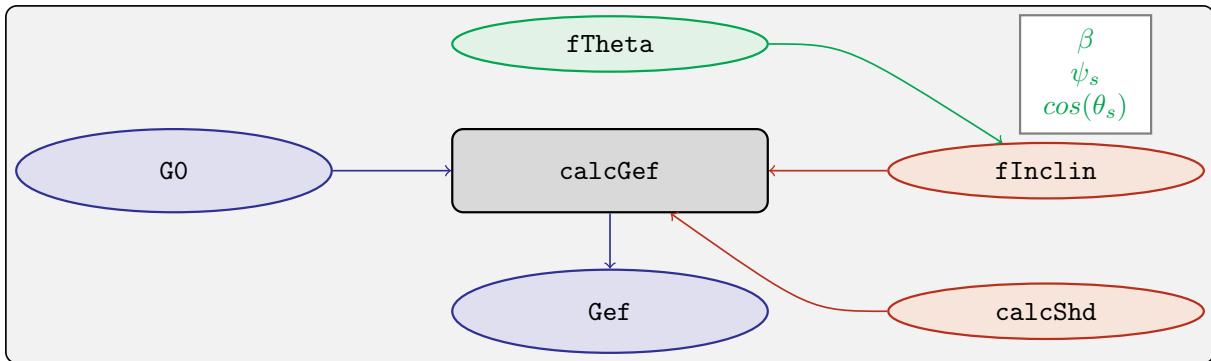


FIGURA 4.20: *Cálculo de la radiación efectiva incidente en el plano del generador mediante la función `calcGef`, la cual emplea la función `fInclin` para el computo de las componentes efectivas, la función `fTheta` que provee a la función anterior los ángulos necesarios para su computo y la función `calcShd` que reprocesa el objeto de clase `Gef` resultante, añadiéndole el efecto de las sombras producidas entre módulos.*

- **fTheta:** la cual, partiendo del ángulo de inclinación (β) y la orientación (α), calcula el ángulo de inclinación en cada instante (β), el ángulo azimutal (ψ_s) y el coseno del ángulo de incidencia de la radiación solar en la superficie ($\cos(\theta_s)$). Como principal argumento tiene `modeTrk`, el cual determina el sistema de seguimiento que tiene el sistema:

- **fixed:** para sistemas estáticos.

```

1 BTd <- fBTd(mode = 'serie')[1:10]
2 sol <- calcSol(lat, BTd = BTd, keep.night = FALSE)
3 beta <- lat - 10
4 alpha <- 0
5 angGen_fixed <- fTheta(sol = sol, beta = beta, alpha = alpha,
6                           modeTrk = 'fixed')
7 xypplot(angGen_fixed)

```

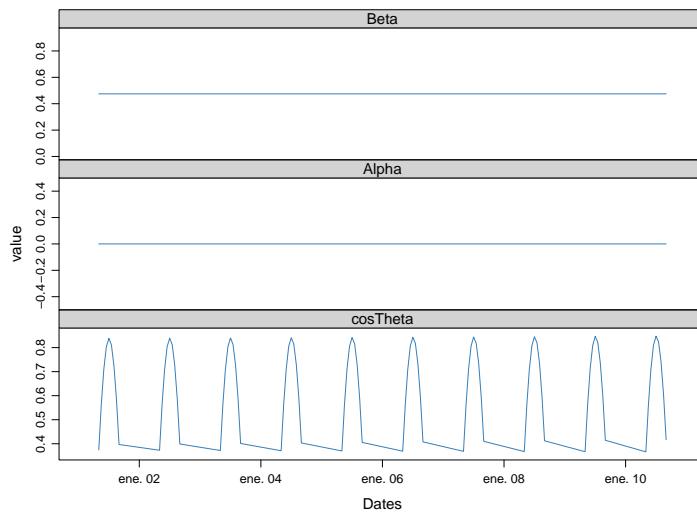


FIGURA 4.21: Representación gráfica del resultado de la función `fTheta` para un sistema fijo.

- **two:** para sistemas de seguimiento de doble eje.

```

1 angGen_two <- fTheta(sol = sol, beta = beta, alpha = alpha,
2                         modeTrk = 'two')
3 xyplot(angGen_two)

```

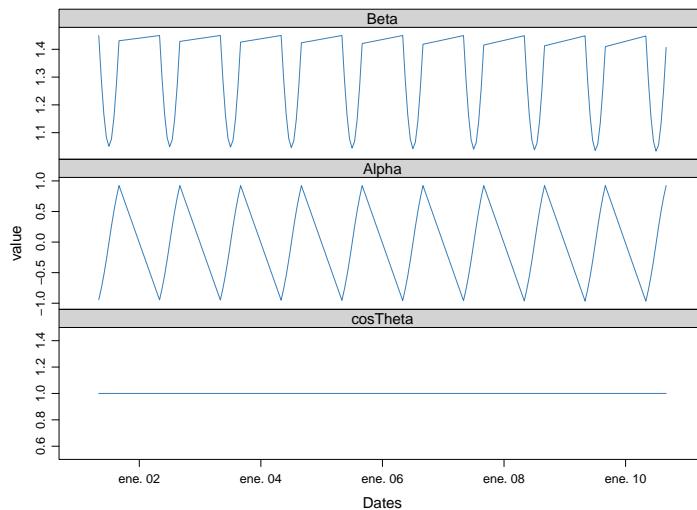


FIGURA 4.22: Representación gráfica del resultado de la función `fTheta` para un sistema de seguimiento de dos ejes.

- **horiz:** para sistemas de seguimiento horizontal Norte-Sur.

```

1 angGen_horiz <- fTheta(sol = sol, beta = beta, alpha = alpha,
2                           modeTrk = 'horiz')
3 xyplot(angGen_horiz)

```

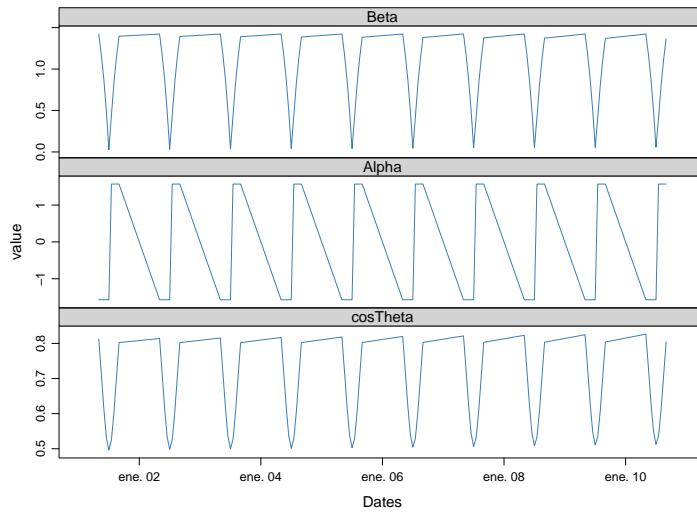


FIGURA 4.23: Representación gráfica del resultado de la función **fTheta** para un sistema de seguimiento horizontal.

También, tiene un argumento **BT** que indica cuando se usa la técnica de backtracking para un sistema horizontal Norte-Sur. Para funcionar, necesita de los argumentos **struct**, el cual presenta una lista con la altura de los módulos, y **dist**, el cual presenta un **data.frame** (o **data.table**) con la distancia que separa los módulos en la dirección Este-Oeste.

```

1 struct <- list(L = 1)
2 distances <- data.table(Lew = 2)
3 angGen_BT <- fTheta(sol = sol, beta = beta, alpha = alpha,
4                      modeTrk = 'horiz', BT = TRUE,
5                      struct = struct, dist = distances)
6 xyplot(angGen_BT)

```

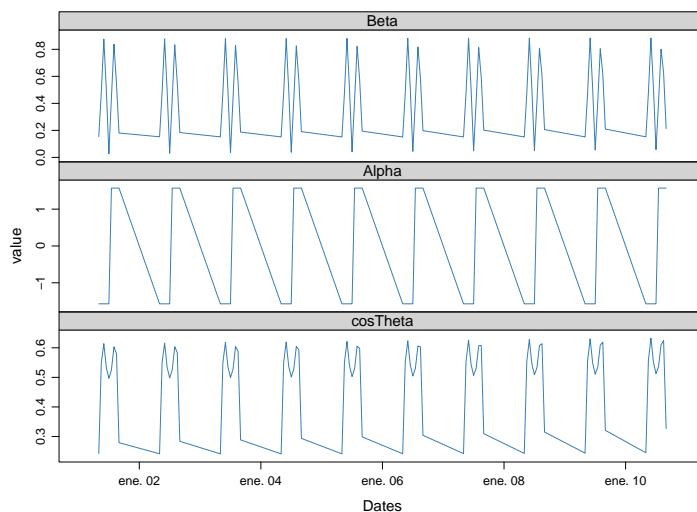


FIGURA 4.24: Representación gráfica del resultado de la función **fTheta** para un sistema de seguimiento horizontal con backtracking.

- **fInclin:** la cual, partiendo del resultado de **fTheta** y de un objeto de clase **G0**, calcula la irradiancia solar incidente en una superficie inclinada junto con los efectos del ángulo de incidencia y la suciedad para obtener la irradiancia efectiva. Como argumentos principales están:

- **iS:** permite seleccionar entre 4 valores del 1 al 4 correspondientes al grado de suciedad del módulo. Siendo 1 limpio y 4 alto y basándose en los valores de la tabla 3.2 calcula la irradiancia efectiva. Por defecto, tiene valor 2 (grado de suciedad bajo).

```

1 compI <- calcG0(lat, modeRad = 'bd', dataRad = bd[1:31], keep.night = FALSE
2   )
3 sol <- calcSol(lat, BTi = indexI(compI))
4 angGen <- fTheta(sol = sol, beta = beta, alpha = alpha)
5 inclin_limpio <- fInclin(compI = compI, angGen = angGen, iS = 1)
6 inclin_limpio[, group := 'is = 1']
7 inclin_sucio <- fInclin(compI = compI, angGen = angGen, iS = 4)
8 inclin_sucio[, group := 'is = 4']
9 inclin_is <- rbind(inclin_limpio, inclin_sucio)
10 xyplot(Gef + Def + Bef + Ref ~ Dates, inclin_is,
11         groups = group, type = 'l', auto.key = TRUE,
12         par.settings = solaR.theme, scales = list(y = 'free'),
13         ylab = '', layout = c(2, 2))

```

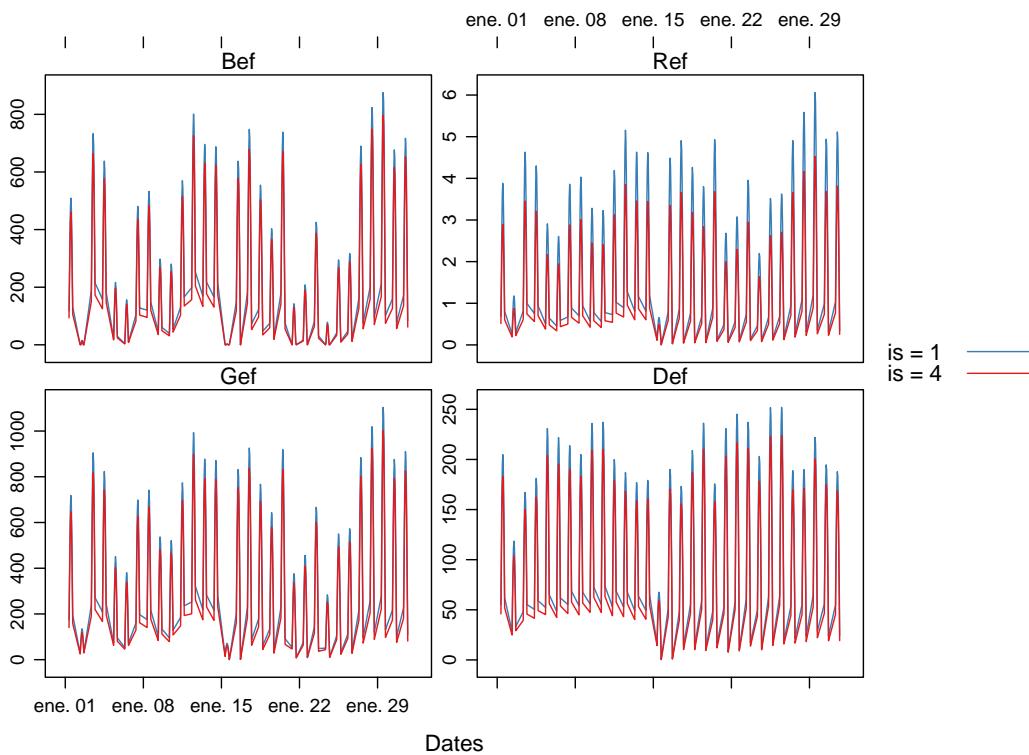


FIGURA 4.25: Comparación entre resultados de la función **fInclin** tomando diferentes niveles de suciedad.

- **alb:** correspondiente al coeficiente de reflexión del terreno para la irradiancia de albedo. Por defecto, tiene un valor de 0,2 (valor aceptable para un terreno normal).

```

1 inclin_alb0 <- fInclin(compl = compl, angGen = angGen, alb = 0)
2 inclin_alb0[, group := 'alb = 0']
3 inclin_alb1 <- fInclin(compl = compl, angGen = angGen, alb = 1)
4 inclin_alb1[, group := 'alb = 1']
5 inclin_alb <- rbind(inclin_alb0, inclin_alb1)
6 xyplot(Gef + Def + Bef + Ref ~ Dates, inclin_alb,
7     groups = group, type = 'l', auto.key = TRUE,
8     par.settings = solaR.theme, scales = list(y = 'free'),
9     ylab = '', layout = c(2, 2))

```

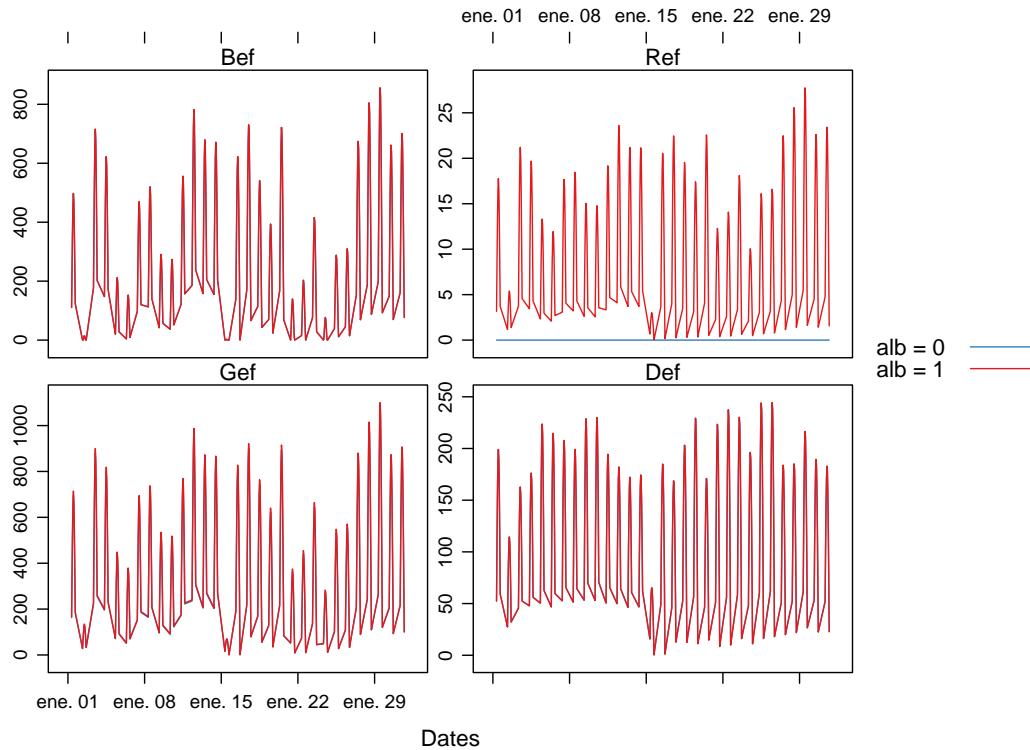


FIGURA 4.26: Comparación entre resultados de la función **fInclin** tomando diferentes valores del coeficiente de reflexión del terreno.

Además, cuenta con dos argumentos adicionales, **horizBright**, el cual, cuando su valor es **TRUE** (el que tiene por defecto), realiza una corrección de la radiación difusa [RBD90], y **HCPV** es el acrónimo de **High Concentration PV system**⁸ (sistema fotovoltaico de alta concentración) que cuando su valor es **TRUE** (por defecto está puesto en **FALSE**), anula los valores de radiación difusa y de albedo.

```

1 inclin_horizBright <- fInclin(compl = compl, angGen = angGen,
2                                 horizBright = FALSE)
3 summary(inclin_horizBright)

```

⁸La tecnología de concentración fotovoltaica funciona gracias a unos dispositivos ópticos que permiten concentrar la radiación solar sobre una célula fotovoltaica de tamaño reducido pero con una eficiencia muy superior a las células tradicionales. Con ello, se consigue emplear una menor cantidad de semiconductores, reduciendo los costes.

Dates	Bo	Bn	G
Min. :2024-01-01 08:00:00.00	Min. : 0.0	Min. : 0.0	Min. : 0.4059
1st Qu.:2024-01-09 09:45:00.00	1st Qu.: 642.2	1st Qu.:212.5	1st Qu.: 229.7549
Median :2024-01-17 09:30:00.00	Median :1005.1	Median :484.0	Median : 427.1040
Mean :2024-01-16 21:56:08.92	Mean : 898.5	Mean :455.2	Mean : 449.1999
3rd Qu.:2024-01-24 13:15:00.00	3rd Qu.:1152.8	3rd Qu.:685.0	3rd Qu.: 670.5626
Max. :2024-01-31 17:00:00.00	Max. :1248.5	Max. :996.1	Max. :1110.4693
D	Di	Dc	B R
Min. : 0.4013	Min. : 0.4013	Min. : 0.00	Min. : 0.0 Min. :0.004576
1st Qu.: 77.1846	1st Qu.: 40.3333	1st Qu.: 31.04	1st Qu.:117.3 1st Qu.:1.396238
Median :145.0080	Median : 57.7167	Median : 64.29	Median :264.7 Median :3.119382
Mean :137.6786	Mean : 72.4888	Mean : 65.19	Mean :308.3 Mean : 3.203413
3rd Qu.:188.6954	3rd Qu.:100.4762	3rd Qu.:101.48	3rd Qu.:495.7 3rd Qu.:4.743507
Max. :259.1854	Max. :195.6988	Max. :178.88	Max. :878.1 Max. :8.083268
FTb	FTd	FTr	Dief Dcef
Min. :0.005218	Min. :0.06474	Min. :0.2995	Min. : 0.3678 Min. : 0.00
1st Qu.:0.010325	1st Qu.:0.06474	1st Qu.:0.2995	1st Qu.: 36.9678 1st Qu.: 26.72
Median :0.022076	Median :0.06474	Median :0.2995	Median : 52.9007 Median : 60.26
Mean :0.062300	Mean : 0.06474	Mean :0.2995	Mean : 66.4403 Mean : 61.89
3rd Qu.:0.096613	3rd Qu.:0.06474	3rd Qu.:0.2995	3rd Qu.: 92.0924 3rd Qu.: 98.10
Max. :0.344085	Max. :0.06474	Max. :0.2995	Max. :179.3695 Max. :174.34
Gef	Def	Bef	Ref
Min. : 0.371	Min. : 0.3678	Min. : 0.0	Min. :0.003142
1st Qu.: 198.916	1st Qu.: 69.9858	1st Qu.:101.2	1st Qu.:0.958529
Median : 401.396	Median :135.3589	Median :242.6	Median :2.141483
Mean : 423.118	Mean : 128.3346	Mean :292.6	Mean :2.199171
3rd Qu.: 640.731	3rd Qu.:177.6177	3rd Qu.:479.4	3rd Qu.:3.256459
Max. :1077.330	Max. :243.0163	Max. :855.8	Max. :5.549235

```

1 inclin_HCPV <- fInclin(compI = compI, angGen = angGen,
2                           HCPV = TRUE)
3 summary(inclin_HCPV)

```

Dates	Bo	Bn	G
Min. :2024-01-01 08:00:00.00	Min. : 0.0	Min. : 0.0	Min. : 0.4059
1st Qu.:2024-01-09 09:45:00.00	1st Qu.: 642.2	1st Qu.:212.5	1st Qu.: 230.4207
Median :2024-01-17 09:30:00.00	Median :1005.1	Median :484.0	Median : 427.5819
Mean :2024-01-16 21:56:08.92	Mean : 898.5	Mean :455.2	Mean : 449.7985
3rd Qu.:2024-01-24 13:15:00.00	3rd Qu.:1152.8	3rd Qu.:685.0	3rd Qu.: 671.2063
Max. :2024-01-31 17:00:00.00	Max. :1248.5	Max. :996.1	Max. :1110.9892
D	Di	Dc	B R
Min. : 0.4013	Min. : 0.4013	Min. : 0.00	Min. : 0.0 Min. :0.004576
1st Qu.: 77.2814	1st Qu.: 40.7932	1st Qu.: 31.04	1st Qu.:117.3 1st Qu.:1.396238
Median :145.6833	Median : 58.3298	Median : 64.29	Median :264.7 Median :3.119382
Mean :138.2773	Mean : 73.0875	Mean : 65.19	Mean :308.3 Mean : 3.203413
3rd Qu.:189.5790	3rd Qu.:101.0182	3rd Qu.:101.48	3rd Qu.:495.7 3rd Qu.:4.743507
Max. :260.6431	Max. :197.0369	Max. :178.88	Max. :878.1 Max. :8.083268
FTb	FTd	FTr	Dief Dcef Gef
Min. :0.005218	Min. :0.06474	Min. :0.2995	Min. : 0 Min. :0 Min. : 0.0
1st Qu.:0.010325	1st Qu.:0.06474	1st Qu.:0.2995	1st Qu.: 0 1st Qu.:0 1st Qu.:101.2
Median :0.022076	Median :0.06474	Median :0.2995	Median : 0 Median :0 Median :242.6
Mean :0.062300	Mean : 0.06474	Mean :0.2995	Mean : 0 Mean :0 Mean :292.6
3rd Qu.:0.096613	3rd Qu.:0.06474	3rd Qu.:0.2995	3rd Qu.: 0 3rd Qu.:0 3rd Qu.:479.4
Max. :0.344085	Max. :0.06474	Max. :0.2995	Max. : 0 Max. :0 Max. :855.8
Def	Bef	Ref	
Min. :0	Min. : 0.0	Min. : 0	
1st Qu.:0	1st Qu.:101.2	1st Qu.:0	
Median :0	Median :242.6	Median :0	
Mean :0	Mean :292.6	Mean : 0	
3rd Qu.:0	3rd Qu.:479.4	3rd Qu.:0	
Max. :0	Max. :855.8	Max. : 0	

Finalmente, esta función le otorga estos datos a la función `calcGef` para que produzca un objeto de clase `Gef` como resultado. Esta función tiene como argumentos principales los mismos

4. DESARROLLO DEL CÓDIGO

que los que tiene **calcGO** 4.3, es decir, **modeRad** y **dataRad**. Y además, como es lógico, con todos los argumentos mencionados con anterioridad en **fTheta** y **fInclin**.

```

1 gef_prom <- calcGef(lat = lat, modeTrk = 'two', modeRad = 'prom',
2                         dataRad = prom,
3                         beta = lat-10, alpha = 0,
4                         iS = 2, alb = 0.2,
5                         horizBright = TRUE, HCPV = FALSE)
6 show(gef_prom)
```

```

Object of class Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates      Bod      Bnd      Gd      Dd      Bd      Gefd      Defd      Befd
  <char>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: Jan. 2024 14.13536 4.924221 6.522313 1.440413 4.924221 6.348801 1.384087 4.825736
2: Feb. 2024 15.42754 5.034287 6.875052 1.672079 5.034287 6.680139 1.599929 4.933601
3: Mar. 2024 16.58107 5.163713 7.329138 1.998110 5.163713 7.104641 1.902356 5.060439
4: Apr. 2024 17.64047 6.408617 8.843422 2.265896 6.408617 8.578222 2.158071 6.280444
5: May. 2024 18.70771 7.617499 10.178196 2.394606 7.617499 9.885240 2.284334 7.465149
6: Jun. 2024 19.87238 9.102430 11.606533 2.329653 9.102430 11.293417 2.230338 8.920381
7: Jul. 2024 18.51695 10.037233 11.801533 2.029150 9.589205 11.495648 1.948530 9.397421
8: Aug. 2024 17.34098 8.640959 10.777404 1.947410 8.640959 10.493150 1.869393 8.468140
9: Sep. 2024 16.25295 6.698488 8.831006 1.948075 6.698488 8.584604 1.864962 6.564518
10: Oct. 2024 15.16994 4.546024 6.418653 1.711039 4.546024 6.226290 1.631551 4.455104
11: Nov. 2024 14.00493 4.638289 6.247341 1.452953 4.638289 6.076159 1.393353 4.545523
12: Dec. 2024 12.70717 3.439788 4.825181 1.254616 3.439788 4.685547 1.198824 3.370992

Yearly values:
  Dates      Bod      Bnd      Gd      Dd      Bd      Gefd      Defd      Befd
  <int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 5988.455 2326.882 3058.651 684.4232 2312.993 2973.115 654.591 2266.733
-----
Mode of tracking: two
Inclination limit: 90
```

Sin embargo, como argumento importante está **modeShd**, el cual permite incluir el efecto de las sombras entre módulos al objeto **Gef** mediante el uso de la función **calcShd**. Esta opción añade las variables **Gef0**, **Def0** y **Bef0**, las cuales son las componentes de radiación efectiva previas a aplicar el efecto de las sombras con el fin de poder comparar.

```

1 struct <- list(W=23.11, L=9.8, Nrow=2, Ncol=8)
2 distances <- data.table(Lew=40, Lns=30, H=0)
3 gef_shd <- calcShd(radEf = gef_prom, modeShd = 'prom',
4                      struct = struct, distances = distances)
5 show(gef_shd)
```

```

Object of class Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates      Gef0d      Def0d      Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
```

```

<char>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: Jan. 2024  6.348801  1.384087  4.825736  6.522313  1.440413  4.924221  6.104126  1.343455  4.621693
2: Feb. 2024  6.680139  1.599929  4.933601  6.875052  1.672079  5.034287  6.406274  1.553670  4.705996
3: Mar. 2024  7.104641  1.902356  5.060439  7.329138  1.998110  5.163713  6.788630  1.848127  4.798657
4: Apr. 2024  8.578222  2.158071  6.280444  8.843422  2.265896  6.408617  8.295340  2.112064  6.043569
5: May. 2024  9.885240  2.284334  7.465149  10.178196  2.394606  7.617499  9.688308  2.253942  7.298609
6: Jun. 2024  11.293417  2.230338  8.920381  11.606533  2.329653  9.102430  11.115054  2.205314  8.767042
7: Jul. 2024  11.495648  1.948530  9.397421  11.801533  2.029150  9.589205  11.308971  1.924962  9.234312
8: Aug. 2024  10.493150  1.869393  8.468140  10.777404  1.947410  8.640959  10.196758  1.830334  8.210807
9: Sep. 2024  8.584604  1.864962  6.564518  8.831006  1.948075  6.698488  8.228309  1.810198  6.262986
10: Oct. 2024 6.226290  1.631551  4.455104  6.418653  1.711039  4.546024  6.018374  1.595528  4.283212
11: Nov. 2024 6.076159  1.393353  4.545523  6.247341  1.452953  4.638289  5.875732  1.359514  4.378935
12: Dec. 2024 4.685547  1.198824  3.370992  4.825181  1.254616  3.439788  4.575893  1.179346  3.280817

Yearly values:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 2973.115 654.591 2266.733 3058.651 684.4232 2312.993 2886.328 640.9157 2193.621
-----
Mode of tracking: two
Inclination limit: 90

```

```

1 gef_shd2 <- calcGef(lat = lat, modeTrk = 'two', dataRad = prom,
2                               modeShd = 'prom', struct = struct, distances = distances)
3 show(gef_shd2)

```

```

Object of class  Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <char>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: Jan. 2024  6.348801  1.384087  4.825736  6.522313  1.440413  4.924221  6.104126  1.343455  4.621693
2: Feb. 2024  6.680139  1.599929  4.933601  6.875052  1.672079  5.034287  6.406274  1.553670  4.705996
3: Mar. 2024  7.104641  1.902356  5.060439  7.329138  1.998110  5.163713  6.788630  1.848127  4.798657
4: Apr. 2024  8.578222  2.158071  6.280444  8.843422  2.265896  6.408617  8.295340  2.112064  6.043569
5: May. 2024  9.885240  2.284334  7.465149  10.178196  2.394606  7.617499  9.688308  2.253942  7.298609
6: Jun. 2024  11.293417  2.230338  8.920381  11.606533  2.329653  9.102430  11.115054  2.205314  8.767042
7: Jul. 2024  11.495648  1.948530  9.397421  11.801533  2.029150  9.589205  11.308971  1.924962  9.234312
8: Aug. 2024  10.493150  1.869393  8.468140  10.777404  1.947410  8.640959  10.196758  1.830334  8.210807
9: Sep. 2024  8.584604  1.864962  6.564518  8.831006  1.948075  6.698488  8.228309  1.810198  6.262986
10: Oct. 2024 6.226290  1.631551  4.455104  6.418653  1.711039  4.546024  6.018374  1.595528  4.283212
11: Nov. 2024 6.076159  1.393353  4.545523  6.247341  1.452953  4.638289  5.875732  1.359514  4.378935
12: Dec. 2024 4.685547  1.198824  3.370992  4.825181  1.254616  3.439788  4.575893  1.179346  3.280817

Yearly values:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 2973.115 654.591 2266.733 3058.651 684.4232 2312.993 2886.328 640.9157 2193.621
-----
Mode of tracking: two
Inclination limit: 90

```

El argumento `modeShd` puede ser de distintas maneras:

- **area:** el efecto de las sombras se calcula como una reducción proporcional de las irradiancias difusa circunsolar y directa.

4. DESARROLLO DEL CÓDIGO

```

1 gef_shdarea <- calcGef(lat, modeTrk = 'two', dataRad = prom,
2                           modeShd = 'area',
3                           struct = struct, distances = distances)
4 show(gef_shdarea)

```

```

Object of class Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd    Gefd    Defd    Befd
  <char>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: Jan. 2024  6.348801 1.384087 4.825736 6.522313 1.440413 4.924221 5.877879 1.305883 4.433019
2: Feb. 2024  6.680139 1.599929 4.933601 6.875052 1.672079 5.034287 6.291348 1.534257 4.610483
3: Mar. 2024  7.104641 1.902356 5.060439 7.329138 1.998110 5.163713 6.743478 1.840379 4.761253
4: Apr. 2024  8.578222 2.158071 6.280444 8.843422 2.265896 6.408617 8.254928 2.105491 6.009730
5: May. 2024  9.885240 2.284334 7.465149 10.178196 2.394606 7.617499 9.660175 2.249601 7.274817
6: Jun. 2024 11.293417 2.230338 8.920381 11.606533 2.329653 9.102430 11.089573 2.201739 8.745137
7: Jul. 2024 11.495648 1.948530 9.397421 11.801533 2.029150 9.589205 11.282303 1.921596 9.211011
8: Aug. 2024 10.493150 1.869393 8.468140 10.777404 1.947410 8.640959 10.154416 1.824754 8.174045
9: Sep. 2024  8.584604 1.864962 6.564518 8.831006 1.948075 6.698488 8.177410 1.802375 6.219910
10: Oct. 2024  6.226290 1.631551 4.455104 6.418653 1.711039 4.546024 5.950189 1.583714 4.226840
11: Nov. 2024  6.076159 1.393353 4.545523 6.247341 1.452953 4.638289 5.705306 1.330740 4.237284
12: Dec. 2024  4.685547 1.198824 3.370992 4.825181 1.254616 3.439788 4.440179 1.155239 3.169210

Yearly values:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd    Gefd    Defd    Befd
  <int>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: 2024 2973.115 654.591 2266.733 3058.651 684.4232 2312.993 2856.633 636.0199 2168.822
-----
Mode of tracking: two
Inclination limit: 90

```

- **prom:** cuando `modeTrk` es `two`, se puede calcular el efecto de las sombras de un seguidor promedio.

```

1 gef_shdprom <- calcGef(lat, modeTrk = 'two', dataRad = prom,
2                           modeShd = c('area', 'prom'),
3                           struct = struct, distances = distances)
4 show(gef_shdprom)

```

```

Object of class Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd    Gefd    Defd    Befd
  <char>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: Jan. 2024  6.348801 1.384087 4.825736 6.522313 1.440413 4.924221 6.104126 1.343455 4.621693
2: Feb. 2024  6.680139 1.599929 4.933601 6.875052 1.672079 5.034287 6.406274 1.553670 4.705996
3: Mar. 2024  7.104641 1.902356 5.060439 7.329138 1.998110 5.163713 6.788630 1.848127 4.798657
4: Apr. 2024  8.578222 2.158071 6.280444 8.843422 2.265896 6.408617 8.295340 2.112064 6.043569
5: May. 2024  9.885240 2.284334 7.465149 10.178196 2.394606 7.617499 9.688308 2.253942 7.298609

```

```

6: Jun. 2024 11.293417 2.230338 8.920381 11.606533 2.329653 9.102430 11.115054 2.205314 8.767042
7: Jul. 2024 11.495648 1.948530 9.397421 11.801533 2.029150 9.589205 11.308971 1.924962 9.234312
8: Aug. 2024 10.493150 1.869393 8.468140 10.777404 1.947410 8.640959 10.196758 1.830334 8.210807
9: Sep. 2024 8.584604 1.864962 6.564518 8.831006 1.948075 6.698488 8.228309 1.810198 6.262986
10: Oct. 2024 6.226290 1.631551 4.455104 6.418653 1.711039 4.546024 6.018374 1.595528 4.283212
11: Nov. 2024 6.076159 1.393353 4.545523 6.247341 1.452953 4.638289 5.875732 1.359514 4.378935
12: Dec. 2024 4.685547 1.198824 3.370992 4.825181 1.254616 3.439788 4.575893 1.179346 3.280817

Yearly values:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 2973.115 654.591 2266.733 3058.651 684.4232 2312.993 2886.328 640.9157 2193.621
-----
Mode of tracking: two
Inclination limit: 90

```

- **bt:** cuando `modeTrk` es `horiz`, se puede calcular el efecto del *backtracking* en las sombras.

```

1 gef_shdhoriz <- calcGef(lat, modeTrk = 'horiz', dataRad = prom,
2                           modeShd = 'area',
3                           struct = struct, distances = distances)
4 show(gef_shdhoriz)

```

```

Object of class  Gef

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <char>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: Jan. 2024 4.274445 1.0909303 3.118987 4.528022 1.166334 3.285391 3.826940 1.0166151 2.745797
2: Feb. 2024 5.173537 1.3974587 3.699745 5.414413 1.484046 3.839622 4.709780 1.3191237 3.314324
3: Mar. 2024 6.270377 1.8008592 4.379272 6.512568 1.906181 4.498391 5.856407 1.7298195 4.036342
4: Apr. 2024 8.160354 2.1103041 5.938446 8.429640 2.222836 6.072611 7.744288 2.0426359 5.590049
5: May. 2024 9.639011 2.2544315 7.260788 9.932830 2.366831 7.416258 9.158384 2.1802588 6.854334
6: Jun. 2024 11.005388 2.1942042 8.675874 11.320680 2.294944 8.861907 10.355140 2.1029750 8.116855
7: Jul. 2024 11.220872 1.9183453 9.163290 11.527430 2.000253 9.358648 10.747413 1.8585724 8.749603
8: Aug. 2024 10.066277 1.8239013 8.112148 10.352216 1.904515 8.290847 9.601132 1.7626031 7.708301
9: Sep. 2024 7.732062 1.7621525 5.864625 7.991813 1.852070 6.013507 7.317424 1.6984219 5.513717
10: Oct. 2024 5.023316 1.4757157 3.471271 5.250215 1.568278 3.591050 4.691499 1.4182254 3.196944
11: Nov. 2024 4.211801 1.1318865 3.014748 4.452659 1.209397 3.166130 3.846165 1.0701542 2.710845
12: Dec. 2024 3.024846 0.9640813 2.008270 3.237139 1.039367 2.135901 2.849995 0.9330218 1.864479

Yearly values:
  Dates    Gef0d    Def0d    Bef0d      Gd      Dd      Bd      Gefd      Defd      Befd
  <int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 2618.414 607.6589 1975.038 2714.415 640.9193 2030.645 2463.159 583.5528 1843.889
-----
Mode of tracking: horiz
Inclination limit: 90

```

```

1 gef_shdbt <- calcGef(lat, modeTrk = 'horiz', dataRad = prom,
2                           modeShd = c('area', 'bt'),
3                           struct = struct, distances = distances)
4 xyplot(gef_shdbt)

```

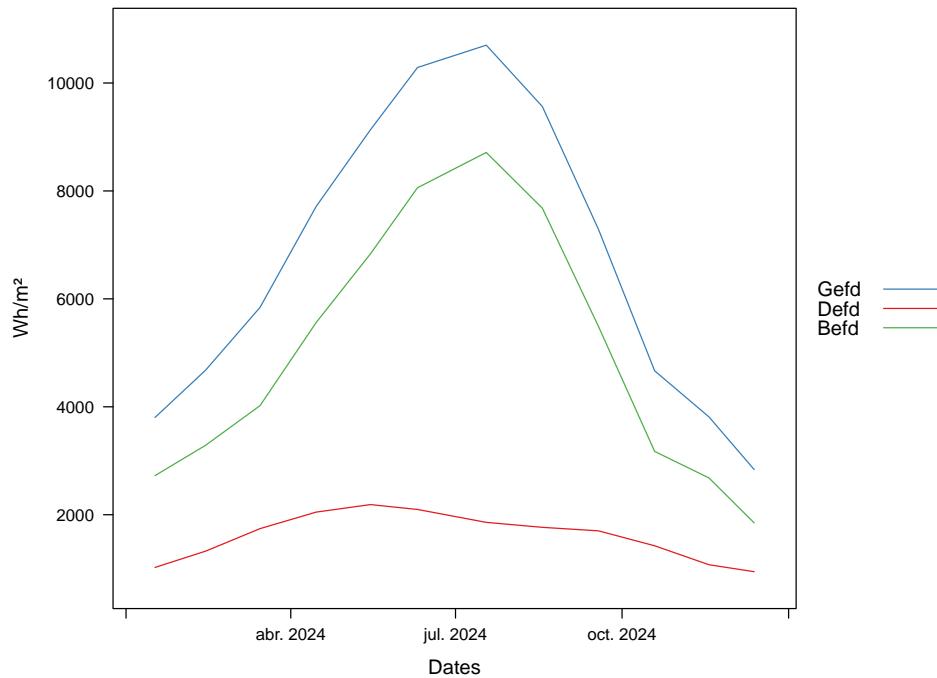


FIGURA 4.27: Representación gráfica de un objeto `Gef` resultado de la función `calcGef`.

4.5. Producción eléctrica de un SFCR

Con la radiación efectiva, se puede estimar la producción eléctrica que va a tener un sistema fotovoltaico conectado a red. Esta estimación se puede calcular mediante la función `prodGCPV`, la cual mediante la función `fProd` procesa un objeto de clase `Gef` y obtiene un objeto `ProdGCPV`.

Como se puede ver en la figura 4.28, `prodGCPV` funciona gracias a la siguiente función:

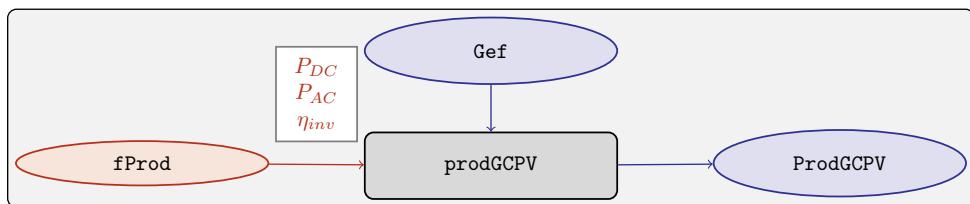


FIGURA 4.28: Estimación de la producción eléctrica de un SFCR mediante la función `prodGCPV`, la cual emplea la función `fProd` para el computo de la potencia a la entrada (P_{DC}), a la salida (P_{AC}) y el rendimiento (η_{inv}) del inversor.

- **fProd**: simula el comportamiento de un sistema fotovoltaico conectado a red bajo diferentes condiciones de temperatura e irradiancia. Tiene los siguientes argumentos:
 - **inclin**: puede ser tanto un objeto de clase `Gef` como un `data.frame` (o `data.table`). Sin embargo, si es un `data.frame`, debe contener como mínimo una columna para `Gef` y otra para `Ta`.

- **module:** una lista de valores numéricos con la información sobre el módulo fotovoltaico:
 - **Vocn:** tensión de circuito abierto en STC (V_{oc}^*) (condiciones estandar de medida). Por defecto, tiene un valor de 57,2V.
 - **Iscn:** corriente de cortocircuito en STC (I_{sc}^*). Por defecto, tiene un valor de 4,7A.
 - **Vmn:** tensión en el punto de máxima potencia en STC (I_{MPP}^*). Por defecto, tiene un valor de 46,08V.
 - **Imn:** corriente de cortocircuito en STC (I_{MPP}^*). Por defecto, tiene un valor de 4,35A).
 - **Ncs:** número de células en serie dentro del módulo. Por defecto, tiene un valor de 96.
 - **Ncp:** número de células en paralelo dentro del módulo. Por defecto, tiene un valor de 1.
 - **CoefVT:** coeficiente de disminución de la tensión de cada célula con la temperatura (dV_{oc}/dT_c). Por defecto, tiene un valor de $-0,0023V/^\circ C$.
 - **TONC:** temperatura de operación nominal de célula ($TONC$). Por defecto, tiene un valor de $47^\circ C$.
- **generator:** lista de valores numéricos con la información sobre el generador:
 - **Nms:** número de módulos en serie. Por defecto, tiene un valor de 12.
 - **Nmp:** número de módulos en paralelo. Por defecto, tiene un valor de 11.
- **inverter:** lista de valores numéricos con la información del inversor DC/AC.
 - **Ki:** coeficientes de la curva de eficiencia del inversor. Se puede presentar en un vector de 3 valores (por defecto, `c(0.01, 0.025, 0.05)`) o una matriz de 9 valores (si tiene dependencia del voltage).
 - **Pinv:** potencia nominal del inversor. Por defecto, tiene un valor de 25000W.
 - **Vmin:** mínima tensión del rango MPP del inversor. Por defecto, tiene un valor de 420V.
 - **Vmax:** máxima tensión del rango MPP del inversor. Por defecto, tiene un valor de 750V.
 - **Gumb:** irradiancia umbral de funcionamiento del inversor. Por defecto, tiene un valor de $20W/m^2$.
- **effSys:** una lista de valores numéricos con la información sobre las pérdidas del sistema.
 - **ModQual:** tolerancia media del set de módulos (%). Por defecto, tiene un valor de 3.
 - **ModDisp:** pérdidas por dispersión en los módulos (%). Por defecto, tiene un valor de 2.
 - **OhmDC:** pérdidas por efecto Joule en el cableado de DC (%). Por defecto, tiene un valor de 1.5.
 - **OhmAC:** pérdidas por efecto Joule en el cableado de AC (%). Por defecto, tiene un valor de 1.5.
 - **MPP:** error promedio del algoritmo de búsqueda del MPP del inversor (%). Por defecto, tiene un valor de 1.
 - **TrafoMT:** pérdidas por el transformador MT (%). Por defecto, tiene un valor de 1.
 - **Disp:** pérdidas por las paradas del sistema (%). Por defecto, tiene un valor de 0.5.

4. DESARROLLO DEL CÓDIGO

```

1 inclin <- calcGef(lat, dataRad = prom, keep.night = FALSE)
2 module <- list(Vocn=51.91, Iscn=14.07, Vmn=43.76, Imn=13.03,
3                 Ncs=24, Ncp=6, CoefVT=0.0049, TONC=45)
4 generator <- list(Nms=22, Nmp=130)
5 inverter <- list(Ki=c(0.002, 0.005, 0.008), Pinv=1.5e6,
6                      Vmin=822, Vmax=1300, Gumb=20)
7 effSys <- list(ModQual=3, ModDisp=2, OhmDC=1.5, OhmAC=1.5,
8                  MPP=1, TrafoMT=1, Disp=0.5)
9 prod <- fProd(inclin = inclin, module = module,
10                  generator = generator, inverter = inverter,
11                  effSys = effSys)
12 summary(prod)

```

Dates	Tc	Voc	Isc	Vmpp
Min. :2024-01-17 08:00:00.00	Min. :11.59	Min. :1051	Min. : 0.6777	Min. : 841.5
1st Qu.:2024-04-15 09:00:00.00	1st Qu.:24.99	1st Qu.:1101	1st Qu.: 327.7952	1st Qu.: 937.6
Median :2024-06-10 17:00:00.00	Median :32.56	Median :1122	Median : 814.6229	Median : 987.0
Mean :2024-06-27 20:53:22.75	Mean :33.75	Mean :1119	Mean : 816.9968	Mean : 985.7
3rd Qu.:2024-09-18 11:00:00.00	3rd Qu.:40.84	3rd Qu.:1142	3rd Qu.:1264.4385	3rd Qu.:1043.2
Max. :2024-12-13 16:00:00.00	Max. :60.35	Max. :1177	Max. :1782.5330	Max. :1100.7
Impp	Vdc	Idc	Pac	Pdc
Min. : 0.6669	Min. : 841.5	Min. : 0.6669	Min. : 0	Min. : 0
1st Qu.: 322.4441	1st Qu.: 937.6	1st Qu.: 322.4441	1st Qu.: 299492	1st Qu.: 313719
Median : 798.7364	Median : 987.0	Median : 798.7364	Median : 693438	Median : 723980
Mean : 800.4704	Mean : 985.7	Mean : 800.4704	Mean : 669255	Mean : 699503
3rd Qu.:1239.9440	3rd Qu.:1043.2	3rd Qu.:1239.9440	3rd Qu.:1020991	3rd Qu.:1066437
Max. :1738.4696	Max. :1100.7	Max. :1738.4696	Max. :1303098	Max. :1362356
EffI				
Min. : 0.0000				
1st Qu.: 0.9839				
Median : 0.9865				
Mean : 0.9594				
3rd Qu.: 0.9870				
Max. : 0.9872				

Esta función brinda tales datos a la función **prodGCPV** para que produzca un objeto de clase **ProdGCPV** como resultado. Esta función tiene como argumentos principales los mismo que **calcGef**, ya que parte de un objeto tipo **Gef**, y los argumentos de la función **fProd**.

```

1 prodFixed <- prodGCPV(lat, modeTrk = 'fixed', dataRad = prom)
2 show(prodFixed)

```

```

Object of class ProdGCPV

Source of meteorological information: prom-
Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly averages:
  Dates      Eac      Edc      Yf
  <char>    <num>    <num>    <num>
1: Jan. 2024 6678.640 6978.809 4.095437
2: Feb. 2024 7194.835 7519.763 4.411976
3: Mar. 2024 7843.267 8195.794 4.809603
4: Apr. 2024 8929.962 9336.190 5.475980
5: May. 2024 9484.132 9914.605 5.815805
6: Jun. 2024 9862.108 10309.878 6.047585
7: Jul. 2024 10019.327 10475.108 6.143994
8: Aug. 2024 9703.467 10142.955 5.950304

```

```

9: Sep. 2024  8757.170  9151.202 5.370022
10: Oct. 2024 6908.624  7220.274 4.236467
11: Nov. 2024 6431.264  6720.101 3.943743
12: Dec. 2024 5229.190  5463.318 3.206614

Yearly values:
  Dates      Eac      Edc      Yf
  <int>    <num>    <num>    <num>
1: 2024 2959931 3093711 1815.072
-----
Mode of tracking: fixed
  Inclination: 27.2
  Orientation: 0
-----
Generator:
  Modules in series: 22
  Modules in parallel: 130
  Nominal power (kWp): 1630.8

```

```

1 prod2x <- prodGCPV(lat, modeTrk = 'two', dataRad = prom)
2 show(prod2x)

```

```

Object of class ProdGCPV

Source of meteorological information: prom-
Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly averages:
  Dates      Eac      Edc      Yf
  <char>    <num>    <num>    <num>
1: Jan. 2024 9725.125 10159.052 5.963585
2: Feb. 2024 10163.306 10616.084 6.232284
3: Mar. 2024 10793.363 11273.933 6.618644
4: Apr. 2024 12779.088 13349.623 7.836319
5: May. 2024 14473.608 15121.428 8.875422
6: Jun. 2024 16252.186 16981.660 9.966072
7: Jul. 2024 15916.049 16632.343 9.759948
8: Aug. 2024 14511.287 15163.412 8.898528
9: Sep. 2024 12350.603 12903.142 7.573565
10: Oct. 2024 9458.655 9878.812 5.800182
11: Nov. 2024 9177.823 9586.046 5.627972
12: Dec. 2024 7284.466 7606.879 4.466938

Yearly values:
  Dates      Eac      Edc      Yf
  <int>    <num>    <num>    <num>
1: 2024 4358566 4553392 2672.735
-----
Mode of tracking: two
  Inclination limit: 90
-----
Generator:
  Modules in series: 22
  Modules in parallel: 130
  Nominal power (kWp): 1630.8

```

```

1 prodHoriz <- prodGCPV(lat, modeTrk = 'horiz', dataRad = prom)
2 xyplot(prodHoriz)

```

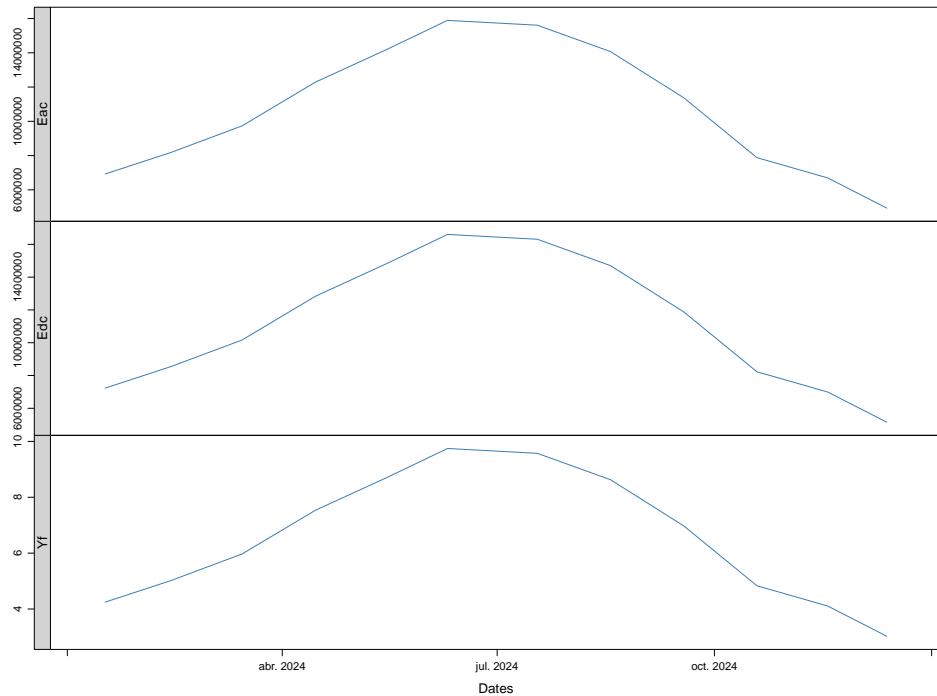


FIGURA 4.29: Representación gráfica de un objeto *ProdGCPV* resultado de la función *prodGCPV*.

4.6. Producción eléctrica de un SFB

De igual forma que en el apartado anterior, se puede estimar la producción eléctrica de un sistema fotovoltaico de bombeo.

Como se puede ver en la figura 4.30, *prodPVPS* funciona gracias a la siguiente función:

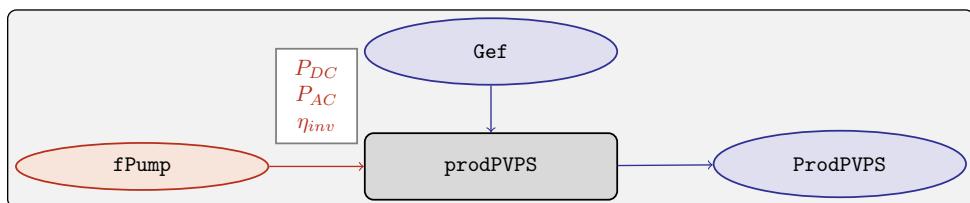


FIGURA 4.30: Estimación de la producción eléctrica de un SFB mediante la función *prodPVPS*, la cual emplea la función *fPump* para el computo del rendimiento de las diferentes partes de una bomba centrífuga alimentada por un convertidor de frecuencia.

- **fPump**: calcula el rendimiento de las diferentes partes de una bomba centrífuga alimentada por un convertidor de frecuencia siguiendo las leyes de afinidad. Tiene solo dos argumentos:

- **pump**: lista que contiene los parámetros de la bomba que va a ser simulada. Puede ser una fila de **pumpCoef**:

```

1 CoefSP8A44 <- pumpCoef [Qn == 8 & stages == 44]
2 show(CoefSP8A44)

```

	Qn	stages	Qmax	Pmn	a	b	c	g	h	i	j	k	l
	<int>	<int>	<num>	<int>	<num>	<num>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	8	44	12	7500	0.1043011	-0.101288	-0.726	-0.24	0.42	0.64	-0.0058	0.095	0.2013

- **H:** el salto manometrico total.

```
1 fSP8A44 <- fPump(pump = CoefSP8A44, H = 40)
```

Obtiene como resultado los siguientes valores y funciones:

- **lim:** rango de valores de la potencia eléctrica de salida.

```
1 show(fSP8A44$lim)
```

```
[1] 190.100 4084.218
```

- **fQ:** función que relaciona el caudal con la potencia eléctrica.

```
1 show(fSP8A44$fQ)
```

```
function (x, deriv = 0L)
{
  deriv <- as.integer(deriv)
  if (deriv < 0L || deriv > 3L)
    stop("'deriv' must be between 0 and 3")
  if (deriv > 0L) {
    z0 <- double(z$n)
    z[c("y", "b", "c")] <- switch(deriv, list(y = z$b, b = 2 *
      z$c, c = 3 * z$d), list(y = 2 * z$c, b = 6 * z$d,
      c = z0), list(y = 6 * z$d, b = z0, c = z0))
    z[["d"]] <- z0
  }
  res <- .splinefun(x, z)
  if (deriv > 0 && z$method == 2 && any(ind <- x <= z$x[1L]))
    res[ind] <- ifelse(deriv == 1, z$y[1L], 0)
  res
}
<bytecode: 0x000001d77d768450>
<environment: 0x000001d77b538ff0>
```

- **fPb:** función que relaciona la potencia del eje de la bomba con la potencia eléctrica del motor.

```
1 show(fSP8A44$fPb)
```

```
function (x, deriv = 0L)
{
  deriv <- as.integer(deriv)
  if (deriv < 0L || deriv > 3L)
    stop("'deriv' must be between 0 and 3")
  if (deriv > 0L) {
    z0 <- double(z$n)
    z[c("y", "b", "c")] <- switch(deriv, list(y = z$b, b = 2 *
      z$c, c = 3 * z$d), list(y = 2 * z$c, b = 6 * z$d,
      c = z0), list(y = 6 * z$d, b = z0, c = z0))
    z[["d"]] <- z0
  }
  res <- .splinefun(x, z)
  if (deriv > 0 && z$method == 2 && any(ind <- x <= z$x[1L]))
    res[ind] <- ifelse(deriv == 1, z$y[1L], 0)
  res
}
<bytecode: 0x000001d77d768450>
<environment: 0x000001d77d693d80>
```

4. DESARROLLO DEL CÓDIGO

- **fPh**: función que relaciona la potencia hidráulica con la potencia eléctrica del motor.

```
1 show(fSP8A44$fPh)
```

```
function (x, deriv = 0L)
{
  deriv <- as.integer(deriv)
  if (deriv < 0L || deriv > 3L)
    stop("'deriv' must be between 0 and 3")
  if (deriv > 0L) {
    z0 <- double(z$n)
    z[c("y", "b", "c")] <- switch(deriv, list(y = z$b, b = 2 *
      z$c, c = 3 * z$d), list(y = 2 * z$c, b = 6 * z$d,
      c = z0), list(y = 6 * z$d, b = z0, c = z0))
    z[["d"]] <- z0
  }
  res <- .splinefun(x, z)
  if (deriv > 0 && z$method == 2 && any(ind <- x <= z$x[1L]))
    res[ind] <- ifelse(deriv == 1, z$y[1L], 0)
  res
}
<bytecode: 0x000001d77d768450>
<environment: 0x000001d7806b14f8>
```

- **fFreq**: función que relaciona la frecuencia con la potencia eléctrica del motor.

```
1 show(fSP8A44$fFreq)
```

```
function (x, deriv = 0L)
{
  deriv <- as.integer(deriv)
  if (deriv < 0L || deriv > 3L)
    stop("'deriv' must be between 0 and 3")
  if (deriv > 0L) {
    z0 <- double(z$n)
    z[c("y", "b", "c")] <- switch(deriv, list(y = z$b, b = 2 *
      z$c, c = 3 * z$d), list(y = 2 * z$c, b = 6 * z$d,
      c = z0), list(y = 6 * z$d, b = z0, c = z0))
    z[["d"]] <- z0
  }
  res <- .splinefun(x, z)
  if (deriv > 0 && z$method == 2 && any(ind <- x <= z$x[1L]))
    res[ind] <- ifelse(deriv == 1, z$y[1L], 0)
  res
}
<bytecode: 0x000001d77d768450>
<environment: 0x000001d7801abe40>
```

Se pueden realizar operaciones con este objeto:

```
1 SP8A44 = with(fSP8A44,{
2   Pac = seq(lim[1],lim[2],l=10)
3   Pb = fPb(Pac)
4   etam = Pb/Pac
5   Ph = fPh(Pac)
6   etab = Ph/Pb
7   f = fFreq(Pac)
8   Q = fQ(Pac)
9   result = data.table(Q,Pac,Pb,Ph,etam,etab,f)})
10 show(SP8A44)
```

	Q	Pac	Pb	Ph	etam	etab	f
	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	0.3133325	190.1000	124.8346	34.15325	0.65666786	0.2735880	20.47033

```

2:  2.0718468 622.7798 429.6728 225.83130 0.6899274 0.5255890 22.33036
3:  4.0764128 1055.4595 752.8970 444.32900 0.7133358 0.5901591 25.51459
4:  5.6406747 1488.1393 1087.3665 614.83354 0.7306887 0.5654336 28.73213
5:  6.9474993 1920.8190 1429.7984 757.27743 0.7443692 0.5296393 31.78514
6:  8.1028841 2353.4988 1778.0156 883.21437 0.7554776 0.4967416 34.69527
7:  9.1607296 2786.1786 2130.4683 998.51953 0.7646560 0.4686855 37.49608
8: 10.1514390 3218.8583 2486.0213 1106.50685 0.7723301 0.4450915 40.21428
9: 11.0937480 3651.5381 2843.8295 1209.21854 0.7788032 0.4252078 42.86977
10: 12.0000000 4084.2179 3203.2578 1308.00000 0.7843014 0.4083343 45.47737

```

Esta función entrega todos estos resultados a **prodPVPS**, la cual calcula los resultados en base a la potencia del generador a simular y devuelve un objeto de clase **ProdPVPS**.

```

1 prodsfb <- prodPVPS(lat, modeTrk = 'fixed', dataRad = prom,
2                         pump = CoefSP8A44, H = 40, Pg = SP8A44$Pac[10])
3 xyplot(prodsfb)

```

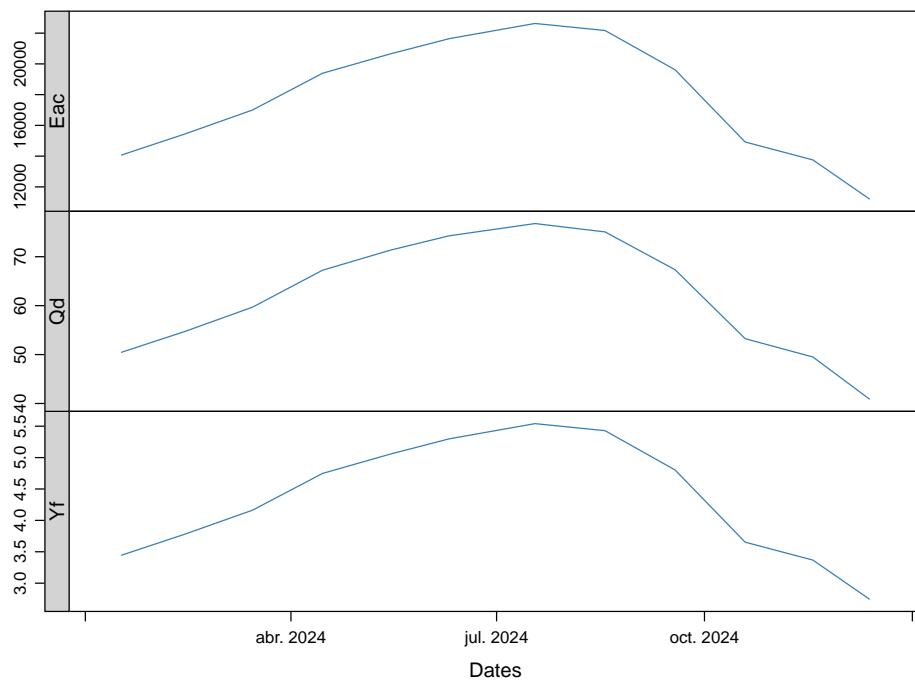


FIGURA 4.31: Representación gráfica de un objeto clase **ProdPVPS** producto de la función **prodPVPS**.

4.7. Optimización de distancias

Por último, el paquete **solar2** contiene una función que permite calcular un conjunto de combinaciones de distancias entre los elementos de un sistema fotovoltaico conectado a red, con el fin de que el usuario posteriormente pueda valorar cuál es la opción mas rentable en base a los precios del cableado y de la ocupación del terreno.

Esta función es **optimShd**, la cual en base a una resolución (determinada por el argumento **res**, el cual, indica el incremento de la secuencia de distancias) obtiene la producción de cada combinación y la plasma en un objeto de clase **Shade**.

4. DESARROLLO DEL CÓDIGO

```

1 struct2x <- list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 3)
2 dist2x <- list(Lew = c(30, 45), Lns = c(20, 40))
3 ShdM2x <- optimShd(lat, dataRad = prom, modeTrk = 'two',
4                         modeShd = c('area', 'prom'),
5                         distances = dist2x, struct = struct2x,
6                         res = 5,
7                         prog = FALSE) #Se quita la barra de progreso
8 shadeplot(ShdM2x)

```

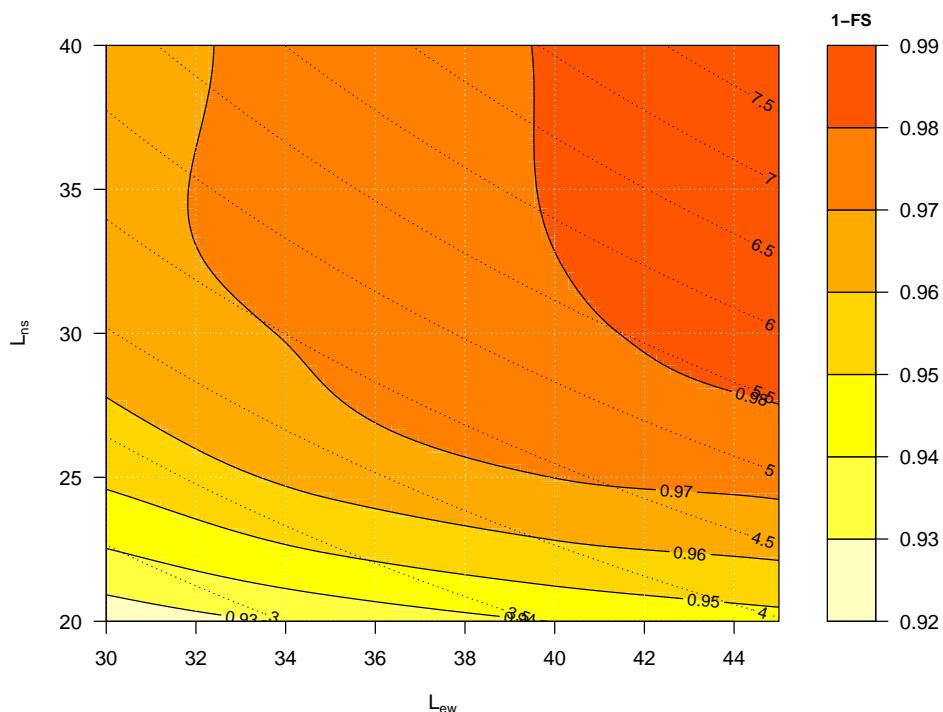


FIGURA 4.32: Ábaco para planta de seguimiento a doble eje producto de la función `optimShd`, e interpretado gráficamente por la función `shadeplot`.

```

1 structHoriz = list(L = 4.83)
2 distHoriz = list(Lew = structHoriz$L * c(2,5))
3 Shd12HorizBT <- optimShd(lat = lat, dataRad = prom,
4                             modeTrk = 'horiz',
5                             betaLim = 60,
6                             distances = distHoriz, res = 2,
7                             struct = structHoriz,
8                             modeShd = 'bt',
9                             prog = FALSE) #Se quita la barra de progreso
10 show(Shd12HorizBT)

```

```

Object of class Shade

Source of meteorological information: prom-
Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly averages:
Dimensions of structure:

```

```

$L
[1] 4.83

Shade calculation mode:
[1] "bt"
Productivity without shadows:
Object of class ProdGCPV

Source of meteorological information: prom-

Latitude of source: 37.2 degrees
Latitude for calculations: 37.2 degrees

Monthly avarages:
  Dates      Eac      Edc      Yf
  <char>    <num>    <num>    <num>
1: Jan. 2024 6789.039 7088.612 4.163135
2: Feb. 2024 8054.032 8409.761 4.938847
3: Mar. 2024 9614.838 10040.667 5.895956
4: Apr. 2024 12160.009 12701.260 7.456691
5: May. 2024 14028.543 14655.892 8.602502
6: Jun. 2024 15691.671 16395.713 9.622356
7: Jul. 2024 15451.474 16145.991 9.475064
8: Aug. 2024 13931.234 14555.274 8.542831
9: Sep. 2024 11241.111 11740.699 6.893209
10: Oct. 2024 7780.058 8123.781 4.770842
11: Nov. 2024 6583.802 6874.444 4.037281
12: Dec. 2024 4875.306 5091.940 2.989607

Yearly values:
  Dates      Eac      Edc      Yf
  <int>    <num>    <num>    <num>
1: 2024 3850450 4022013 2361.151
-----

Mode of tracking: horiz
Inclination limit: 60
-----
Generator:
  Modules in series: 22
  Modules in parallel: 130
  Nominal power (kWp): 1630.8

Summary of results:
  Lew          H          FS          GRR          Yf
Min.   : 9.66  Min.   :0   Min.   :0.04702  Min.   :2.000  Min.   :2036
1st Qu.:13.16 1st Qu.:0   1st Qu.:0.05607  1st Qu.:2.725  1st Qu.:2139
Median :16.66  Median :0   Median :0.07152  Median :3.449  Median :2192
Mean   :16.66  Mean   :0   Mean   :0.07933  Mean   :3.449  Mean   :2174
3rd Qu.:20.16 3rd Qu.:0   3rd Qu.:0.09424  3rd Qu.:4.174  3rd Qu.:2229
Max.   :23.66  Max.   :0   Max.   :0.13783  Max.   :4.899  Max.   :2250

```

```

1 structFixed = list(L = 5)
2 distFixed = list(D = structFixed$L*c(1,3))
3 Shd12Fixed <- optimShd(lat = lat, dataRad = prom,
4                           modeTrk = 'fixed',
5                           distances = distFixed, res = 2,
6                           struct = structFixed,
7                           modeShd = 'area',
8                           prog = FALSE) #Se quita la barra de progreso
9 shadeplot(Shd12Fixed)

```

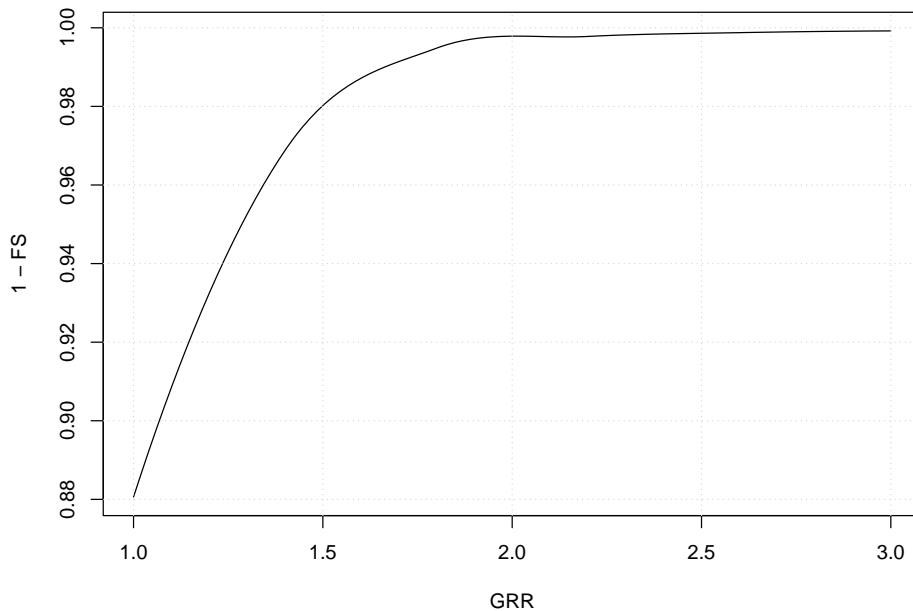


FIGURA 4.33: Ábaco para planta fija producto de la función `optimShd`, e interpretado gráficamente por la función `shadeplot`.

4.8. Aspectos técnicos de la elaboración de un paquete en R

4.8.1. Estructura básica del paquete

En la creación de un paquete en R, la estructura de los archivos es clave para asegurar un desarrollo organizado y que R pueda interactuar correctamente con el código y los datos. Los paquetes de R son esencialmente un conjunto de archivos organizados en un directorio específico. El contenido mínimo requerido incluye:

- Un archivo **DESCRIPTION**, que proporciona la información esencial del paquete.
- Un archivo **NAMESPACE**, que controla qué funciones y objetos son visibles fuera del paquete.
- Subdirectorios como **R/** y **man/**:
 - **R/**: contiene los archivos de código **.R**, que son las funciones, clases y métodos definidos en el paquete.
 - **man/**: contiene las páginas de ayuda y documentación para las funciones, métodos y clases del paquete.

La estructura básica de un paquete puede generarse con `package.skeleton()`, una función que crea los archivos y carpetas necesarios para empezar a trabajar en el desarrollo.

4.8.2. DESCRIPTION

El fichero **DESCRIPTION** es fundamental, ya que incluye la información descriptiva y técnica del paquete, como el nombre, la versión, los autores y las dependencias. Un ejemplo típico de este archivo es el siguiente:

```
Package: pkgname
Version: 0.5-1
Date: 2004-01-01
Title: My First Collection of Functions
Authors@R: c(person("Joe", "Developer", role = c("aut", "cre"),
                  email = "Joe.Developer@some.domain.net"),
            person("Pat", "Developer", role = "aut"),
            person("A.", "User", role = "ctb",
                  email = "A.User@whereever.net"))
Author: Joe Developer and Pat Developer, with contributions from A. User
Maintainer: Joe Developer <Joe.Developer@some.domain.net>
Depends: R (>= 1.8.0), nlme
Suggests: MASS
Description: A short (one paragraph) description of what
the package does and why it may be useful.
License: GPL (>= 2)
URL: http://www.r-project.org, http://www.another.url
```

Los campos principales de este archivo son:

- **Package:** nombre del paquete.
- **Version:** versión del paquete. Generalmente sigue un esquema de numeración semántica (**major.minor-patch**)⁹.
- **Title:** un título breve pero descriptivo de lo que hace el paquete.
- **Authors@R:** especifica el o los autores con sus respectivos roles, como “aut” (autor) y “cre” (creador principal).
- **Maintainer:** persona responsable del mantenimiento del paquete, con su correo electrónico.
- **Depends:** lista de dependencias, es decir, otros paquetes de los que depende el correcto funcionamiento del paquete.
- **Suggests:** lista de paquete que no son obligatorios, pero que pueden ser útiles.
- **Description:** una breve descripción del propósito del paquete.
- **License:** tipo de licencia bajo la cual se distribuye el paquete (GPL, MIT, etc.).

Este archivo es crucial para que los usuarios y el sistema **R** identifiquen las características y requisitos del paquete

⁹Un esquema de numeración semántica es un sistema de versiones que sigue un patrón específico para asignar números a las versiones de software. Se utiliza para indicar claramente la magnitud de los cambios realizados y su impacto en la compatibilidad. Una versión **major** o mayor se refiere a modificaciones grandes o incompatibles con versiones anteriores; **minor** o menor es una versión que incluye mejoras o nuevas funciones compatibles con versiones anteriores y **patch** o parche es una versión que incluye correcciones menores o mejoras que no afectan a la funcionalidad.

4.8.3. NAMESPACE

El archivo **NAMESPACE** es el encargado de gestionar el espacio de nombres del paquete, permitiendo definir qué funciones y objetos serán visibles (exportados) y cuáles se mantendrán internos. Además, es útil para definir qué funciones o métodos de otros paquetes serán importados para su uso dentro del paquete.

R usa un sistema de gestión de **espacio de nombres** que permite al autor del paquete especificar:

- Las **variables** del paquete que se **exportan** (y son, por tanto, accesibles a los usuarios).
- Las **variables** que se **importan** de otros paquetes.
- Las **clases y métodos S3 y S4** que deben registrarse.

El **NAMESPACE** controla la estrategia de búsqueda de variables que utilizan las funciones del paquete:

- En primer lugar, busca entre las creadas localmente (por el código de la carpeta R/).
- En segundo lugar, busca entre las variables importadas explícitamente de otros paquetes.
- En tercer lugar, busca en el **NAMESPACE** del paquete **base**.
- Por último, busca siguiendo el camino habitual (usando **search()**).

```
1 search()
```

[1] ".GlobalEnv"	"package:jsonlite"	"package:httr2"	"package:zoo"
[5] "package:solaR2"	"package:latticeExtra"	"package:lattice"	"package:data.table"
[9] "ESSR"	"package:stats"	"package:graphics"	"package:grDevices"
[13] "package:utils"	"package:datasets"	"package:methods"	"Autoloads"
[17] "package:base"			

Manejo de variables

- Exportar variables:

```
1 export(f, g)
```

Esto asegura que las variables **f** y **g** sean accesibles desde fuera del paquete.

- Importar **todas** las variables de otro paquete:

```
1 import(pkgExt)
```

- Importar variables **concretas** de otro paquete:

```
1 importFrom(pkgExt, var1, var2)
```

Manejo de clases y métodos

- Para registrar un **método** para una **clase** determinada:

```
1 S3method(print, myClass)
```

Esto permite definir cómo se imprimen objetos de la clase **myClass**

- Para los paquetes que utilizan clases y métodos **S4**, es necesario agregar una dependencia explícita en el archivo **DESCRIPTION**:

```
1 import("methods")
```

- Para registrar clases **S4**:

```
1 exportClasses(class1, class2)
```

- Para registrar métodos **S4**:

```
1 exportMethods(method1, method2)
```

- Para importar métodos y clases **S4** de otro paquete:

```
1 importClassesFrom(package, ...)
2 importMethodsFrom(package, ...)
```

4.8.4. Documentación

La documentación en R sigue un formato específico llamado **Rd** (*R documentation*), que está inspirado en **L^AT_EX**. Cada función, método o clase del paquete debe tener una página de documentación asociada, que generalmente se encuentra en el subdirectorio **man/**. Estas páginas incluyen información sobre el uso de la función, argumentos, detalles de la implementación y ejemplos de uso.

```
\name{load}
\alias{load}
\title{Reload Saved Datasets}
\description{
  Reload the datasets written to a file with the function
  \code{save}.
}
\usage{
  load(file, envir = parent.frame())
}
\arguments{
  \item{file}{a connection or a character string giving the
    name of the file to load.}
  \item{envir}{the environment where the data should be
    loaded.}
}
\seealso{
  \code{\link{save}}.
}
\examples{
  ## save all data
  save(list = ls(), file= "all.RData")

  ## restore the saved values to the current environment
  load("all.RData")

  ## restore the saved values to the workspace
  load("all.RData", .GlobalEnv)
}
\keyword{file}
```

El formato tiene varios componentes:

- **name**: el nombre de la función.
- **alias**: nombres alternativos o alias de la función.
- **title**: título breve que describe la función.
- **description**: una descripción de lo que hace la función.
- **usage**: la sintaxis de la función de lo que hace la función.
- **arguments**: explicación de los argumentos que recibe la función.
- **seealso**: enlaces a funciones relacionadas.
- **examples**: ejemplos de cómo utilizar la función.

Esta estructura de documentación permite a los usuarios comprender rápidamente cómo utilizar las funciones del paquete y verificar su funcionalidad con ejemplos prácticos.

Ejemplo práctico de aplicación

Una vez explicado cómo funciona el paquete, se puede realizar una demostración práctica tomando como ejemplo los módulos fotovoltaicos que tiene en su azotea la Escuela Técnica Superior de Ingeniería y Diseño Industrial (en adelante, la ETSIDI).

Se tomará de base un estudio realizado por profesores de la escuela [Adr+17], en el cual comparan la producción energética de seis tipos de tecnologías fotovoltaicas.

En este ejemplo se realizará el mismo análisis tomando tres herramientas distintas: **solaR**, para poder tomar como referencia el paquete del que sale, apreciando las mejoras del programa. **PVSyst**, ya que es uno de los softwares más usados en el ámbito de la fotovoltaica y puede servir como punto de referencia. Y, por último, **solaR2**.

5.1. solaR

Se empieza iniciizando el paquete:

```
1 library(solaR)
```

```
Cargando paquete requerido: zoo
Adjuntando el paquete: 'zoo'

The following objects are masked from 'package:base':
  as.Date, as.Date.numeric

Cargando paquete requerido: lattice
Cargando paquete requerido: latticeExtra
Time Zone set to UTC.
```

En el estudio anterior, se recopilaron datos intradiarios de irradiación intradiaria. Sin embargo, estos datos fueron tomados en un plano inclinado, por lo que ni **solaR**, ni **solaR2** son capaces de interpretarlos correctamente. En su lugar, para este ejemplo se van a tomar datos horarios de la plataforma **PVGIS** [PVG24] de los años 2013, 2014 y 2015 en la localización de la ETSIDI.

5. EJEMPLO PRÁCTICO DE APLICACIÓN

```

1 etsidi_1315 <- readBDi(file = 'TFG/data/PVGIS_1315.csv',
2                         lat = 40.4, time.col = 'Dates',
3                         format = '%Y-%m-%d %H:%M:%S')
4 xyplot(etsidi_1315)

```

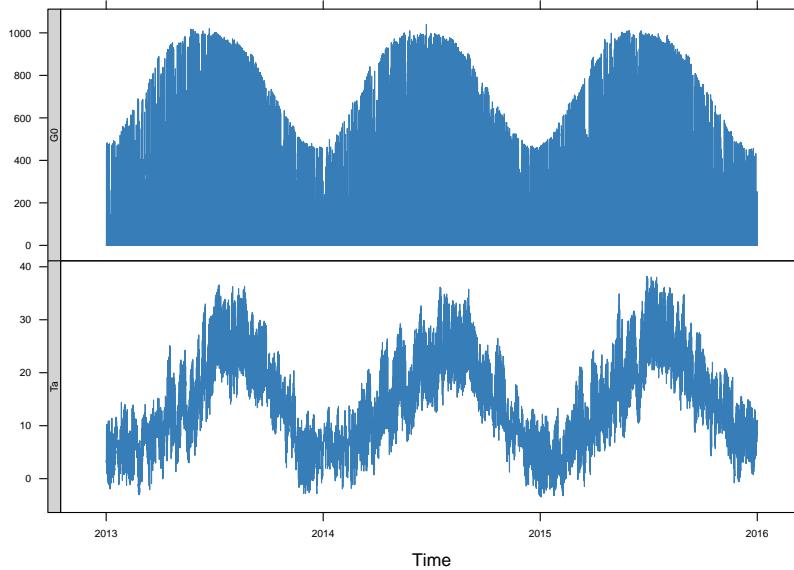


FIGURA 5.1: *Representación gráfica de un objeto Meteo formado con la información meteorológica de la azotea de la ETSIDI entre los años 2013 y 2015 (información horaria).*

Una vez se tienen estos datos, se puede calcular la producción que van a tener los diferentes sistemas fotovoltaicos.

Para ello, se necesitan los parámetros de los diferentes sistemas. En la tabla 5.1, se pueden ver los distintos parámetros de los módulos fotovoltaicos.

TABLA 5.1: *Parámetros técnicos de diferentes tipos de células solares.*

Parámetros Técnicos	mc-Si	pc-Si
Potencia se salida (Wp)	250	220
Voltaje en P_{max} (Vmp)	29.9	29.0
Corriente en P_{max} (Imp)	8.37	7.59
Voltaje en circuito abierto (Voc)	37.1	36.5
Corriente en cortocircuito (Isc)	8.76	8.15
Eficiencia del módulo (%)	15.5	14.4
α_{Isc} (%/K)	0.0043	0.06
β_{Voc} (%/K)	-0.338	-0.37
$\gamma_{P_{mpp}}$ (%/K)	-0.469	-0.45
Temperatura NOC (°C)	43.7	46

Se almacena esta información en listas con la información de cada módulo.

```

1 ## mc-Si
2 module1 <- list(Vocn = 37.1,
3                   Iscn = 8.76,
4                   Vmn = 29.9,
5                   Imn = 8.37,
6                   Ncs = 60,
7                   Ncp = 1,
8                   CoefVT = 0.00338,
9                   TONC = 43.7)
10 ## pc-Si
11 module2 <- list(Vocn = 36.5,
12                   Iscn = 8.15,
13                   Vmn = 29,
14                   Imn = 7.59,
15                   Ncs = 60,
16                   Ncp = 1,
17                   CoefVT = 0.0037,
18                   TONC = 46)

```

Una vez se tiene la información de cada tipo de módulo, en la tabla 5.2 se pueden ver la información de la agrupación de cada sistema.

TABLA 5.2: *Sistemas fotovoltaicos.*

Sistema	Tecnología	Año de Fabricación	Módulos en Serie	Módulos en Paralelo	Sistema STC	Potencia del (W _{pSTC})	Tamaño (m ²)
1	mc-Si	2012	5	1	1250		8
2	pc-Si	2009	5	1	1100		8.2

De la misma manera, se almacenará esta información en listas.

```

1 ## mc-Si
2 generator1 <- list(Nms = 5, Nmp = 1)
3 ## pc-Si
4 generator2 <- list(Nms = 5, Nmp = 1)

```

Una vez se tienen todos los parámetros del sistema fotovoltaico, se requieren los parámetros del inversor que tienen estos sistemas. Para facilitar el estudio, en el artículo explican que se usa el mismo inversor para todos los sistemas. Los parámetros de este se pueden ver en la tabla 5.3.

5. EJEMPLO PRÁCTICO DE APLICACIÓN

TABLA 5.3: *Características del inversor.*

Inversor	SMA Sunny Boy-1200
Potencia máxima DC	1320 W
Corriente máxima DC	12.6 A
Tensión máxima DC	400 V
Rango de tensión fotovoltaica (mpp)	100-320 V
Potencia máxima DC	1320 W
Potencia nominal de salida	1200 W
Maxima potencia aparente	1200 VA
Corriente máxima AC	6.1 A
Eficiencia	92.1 %

Se almacena esta información en otra lista:

```

1 inverter <- list(Ki = c(0.002, 0.005, 0.008),
2   Pinv = 1200,
3   Vmin = 100,
4   Vmax = 320,
5   Gumb = 20)

```

Una vez recopilada toda la información (la información que falta se deja sin añadir para que el propio paquete añada sus valores por defecto), se puede calcular la producción que tuvieron los sistemas:

```

1 prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
2   dataRad = etsidi_1315,
3   beta = 30, alfa = -19,
4   module = module1, generator = generator1,
5   inverter = inverter)
6 prod2 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
7   dataRad = etsidi_1315,
8   beta = 30, alfa = -19,
9   module = module2, generator = generator2,
10  inverter = inverter)
11

```

```

1 show(as.zooY(prod1))

```

Eac	Edc	Yf
2013 1681.077	1757.235	1343.449
2014 1698.613	1775.426	1357.463
2015 1749.536	1828.569	1398.158

```

1 show(as.zooY(prod2))

```

Eac	Edc	Yf
2013 1451.873	1517.779	1319.225
2014 1464.483	1530.833	1330.683
2015 1506.544	1574.704	1368.901

```
1 compare(prod1, prod2)
```

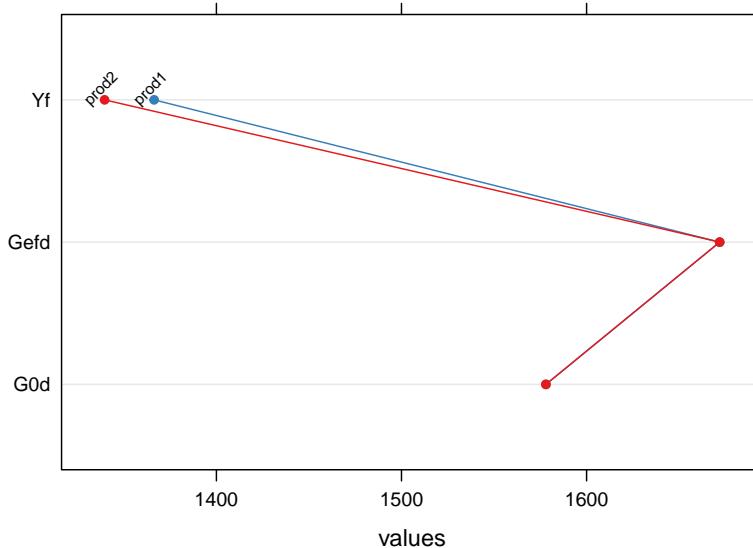


FIGURA 5.2: Comparación gráfica entre las productividades de los sistemas.

5.2. PVsyst

Con la herramienta **PVsyst**, se ha generado un año promedio de datos de irradiación en la localización y con estos datos se han obtenido dos informes (uno por cada sistema).

Por comodidad, en este documento se van a extraer solo unas tablas con los resultados principales. Sin embargo, los informes completos están disponibles en el [github](#) del documento.

En la tabla 5.4 se tienen los resultados de la simulación de los sistemas.

TABLA 5.4: Energía media mensual estimada por **PVSyst** en KWh.

	E_{AC}	Y_f
mc-Si	1704.2	1363
pc-Si	1464.5	1331

5.3. solaR2

Con los datos obtenidos en la sección 5.1, hacemos la misma operación pero con el paquete **solaR2**.

```
1 library(solaR2)
```

5. EJEMPLO PRÁCTICO DE APLICACIÓN

```
Cargando paquete requerido: data.table
data.table 1.15.4 using 6 threads (see ?getDTthreads). Latest news: r-dataratable.com
Cargando paquete requerido: lattice
Cargando paquete requerido: latticeExtra
Time Zone set to UTC.
```

Para ello importamos de la misma manera los datos de radiación.

```
1 etsidi_1315 <- readBDI(file = 'TFG/data/PVGIS_1315.csv',
2                           lat = 40.4, dates.col = 'Dates',
3                           format = '%Y-%m-%d %H:%M:%S')
4 xyplot(etsidi_1315)
```

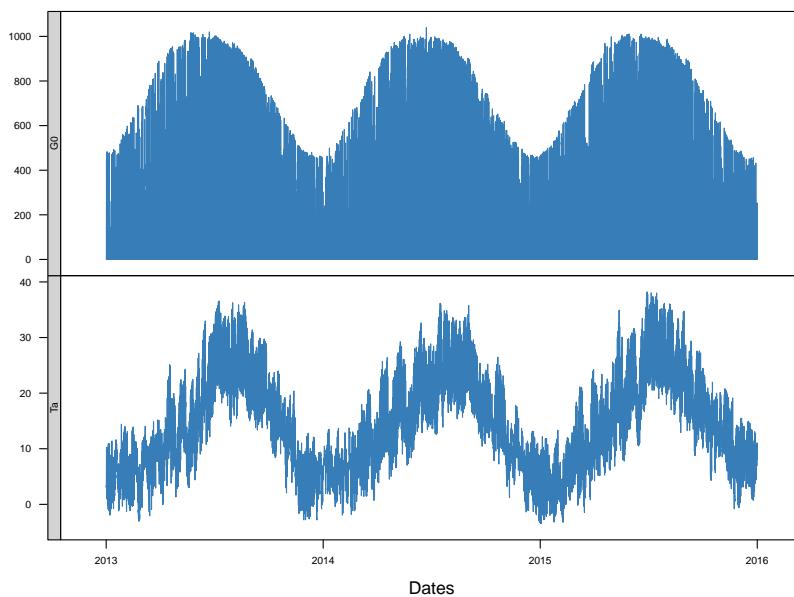


FIGURA 5.3: *Representación gráfica de un objeto Meteo formado con la información meteorológica de la azotea de la ETSIDI entre los años 2013 y 2015 (horaria).*

Con estos datos se procede al cálculo de la producción (los datos de los componentes del sistema son los mismos que los realizados en la sección 5.1).

```
1 prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
2                      dataRad = etsidi_1315,
3                      beta = 30, alpha = -19,
4                      module = module1, generator = generator1,
5                      inverter = inverter)
6 prod2 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
7                      dataRad = etsidi_1315,
8                      beta = 30, alpha = -19,
9                      module = module2, generator = generator2,
10                     inverter = inverter)
```

```
1 show(as.data.tableY(prod1))
```

```
Dates      Eac      Edc      Yf
<int>    <num>    <num>    <num>
1: 2013 1681.077 1757.235 1343.449
2: 2014 1698.613 1775.426 1357.463
3: 2015 1749.536 1828.569 1398.158
```

```
1 show(as.data.tableY(prod2))
```

```
Dates      Eac      Edc      Yf
<int>    <num>    <num>    <num>
1: 2013 1451.873 1517.779 1319.225
2: 2014 1464.483 1530.833 1330.683
3: 2015 1506.544 1574.704 1368.901
```

```
1 compare(prod1, prod2)
```

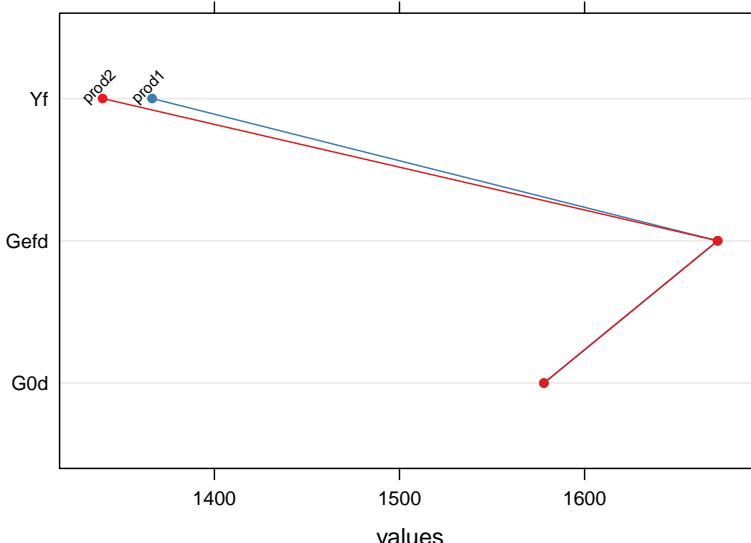


FIGURA 5.4: *Comparación gráfica entre las productividades de los sistemas.*

5.4. Comparación y conclusiones

Como se puede observar en las secciones anteriores, tanto el paquete **solar** como el paquete **solar2** ofrecen los mismos resultados, ya que toman las mismas referencias y estudios para realizar los cálculos. Además, ambos son muy aproximados a los cálculos realizados por **PVSyst**. Sin embargo, el paquete **solar2**, aparte de la corrección de algunos errores, presenta unas claras ventajas frente a su antecesor. Estas son:

- **Modularidad:** el paquete **solar2** presenta muchas funciones que son capaces de realizar pequeñas operaciones, al contrario que **solar**, que no permite esto.

5. EJEMPLO PRÁCTICO DE APLICACIÓN

- **Eficiencia:** al estar basado en **data.table**, el paquete gana eficiencia en operaciones complejas. Para mostrar esto vamos a utilizar el paquete **microbenchmark**.

```
1 ## Con el paquete solaR
2 library(microbenchmark)
3 ## se recortan los datos a un solo año
4 etsidi_13 <- etsidi_1315[as.Date('2013-01-01'), as.Date('2013-12-31')]
5 prodGCPVcustom <- function(){
6   prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
7                      dataRad = etsidi_13, beta = 30, alfa = -19,
8                      module = module1, generator = generator1,
9                      inverter = inverter)
10 }
11 microbenchmark(prodGCPVcustom(), times = 20)
```

```
Unit: milliseconds
      expr      min       lq     mean   median      uq     max neval
prodGCPVcustom() 761.8088 782.3594 801.0824 796.9506 822.6448 832.7593    20
```

```
1 ## Con el paquete solaR2
2 library(microbenchmark)
3 etsidi_13 <- etsidi_1315[as.Date('2013-01-01'), as.Date('2013-12-31')]
4 prodGCPVcustom <- function(){
5   prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
6                      dataRad = etsidi_13, beta = 30, alpha = -19,
7                      module = module1, generator = generator1,
8                      inverter = inverter)
9 }
10 microbenchmark(prodGCPVcustom(), times = 20)
```

```
Unit: milliseconds
      expr      min       lq     mean   median      uq     max neval
prodGCPVcustom() 521.7805 524.6307 530.9213 528.4282 533.4909 574.6818    20
```

En los resultados anteriores se pueden ver los tiempos de ejecución que han tenido las expresiones, el tiempo que interesa es **mean** el cual es el tiempo medio de ejecución. Mientras que con **solaR** se tiene un tiempo de ejecución medio de *805ms*, en **solaR2** es de *533ms* representando una notable mejoría.

6

CAPÍTULO

Conclusiones

6.1. Aportaciones

Al partir de un proyecto ya creado, `solar`, surge la necesidad de explicar cuáles son las aportaciones de este paquete. Para ello, se puede usar el repositorio donde se ha alojado este proyecto y comentar los datos que este ofrece.

6.1.1. Blame

En GitHub, el término *blame* se refiere a una función que permite ver quién fue el autor de cada línea específica de código en un archivo. Por ejemplo, si se quisiera ver quién ha realizado cada línea de la función `fCompD` en GitHub, se podría acceder a su página en el repositorio, y pulsar en el botón *blame*.

Una vez en esta pestaña (figura 6.1), se pueden distinguir varias partes:

- A la izquierda, se puede ver la fecha en la que se modificó esa línea por última vez.
- A continuación, se tiene el nombre del *commit*¹ en el que se cambió.
- Después, se puede tener acceso al *blame prior*, en el cual se puede ver cómo era el resto del archivo cuando se realizó ese *commit*.
- Por último, se muestra el número de línea y su contenido.

Este análisis resulta revelador a la hora de comprender los aportes de este proyecto, ya que se pueden entender los cambios que se han hecho y comprenderlos en el contexto de cada momento del proyecto.

6.1.2. Insights

GitHub también cuenta con una serie de herramientas y características que proporcionan información detallada y análisis sobre el rendimiento, la actividad y la salud de un repositorio. Entre ellas, se encuentra *contributors*. Esta herramienta muestra información sobre las contri-

6. CONCLUSIONES

solarR / R / fCompD.R

fdelgadol Global variables

```

2 weeks ago Global variables 1 util::globalVariables('lat')
2
3 fComp <- function(sol, Gdb, corr = 'CRH', f)
4 {
5   if((cor >= 0.9 & !is.na(corr)) | (cor <= -0.9 & !is.na(corr))) {
6     warning("Wrong descriptor of correlation Fd-kde. Set CRH")
7     corr <- "CRH"
8   }
9   if(class(sol)[1] != "Sol") {
10     sol <- sol[, cmatch(lat = unique(lat), Df1 = Dates)]
11   }
12   if(class(Gdb)[1] == "Raster") {
13     dt <- copy(data.table(Gdb))
14     dt[!(Dates %in% names(dt))] <- NULL
15     dt[, Dates := ordered(Dates)]
16     setnames(dt, c("Date"))
17     setkey(dt, "Date")
18   }
19   lat <- unique(Dates)
20   lat <- unique(dt$lat)
21   dt[, lat := NULL]
22   jdtm(lat = getdate(sol))
23   Gdb <- dt@data@W, lat)
24 }
25
26 if(is.data.table(Gdb) == TRUE) {
27   Gdb <- as.list(Gdb)
28   Gdb <- getData(Gdb)$GDB
29
30 ix.maj <- (Gd@Bsd)
31
32 ## the Direct and Diffuse data is not given
33 if(corr != "none") {
34   Fd <- melt(corr,
35             Gdb = rbind(Gdb, sol),
36             Part = "Diffuse", Gdb,
37             Lat = rbind(lat, Gdb),
38             DOD = Fd@X001@sol, Gdb,
39             CLINED = Fd@CLINED@sol, Gdb,
40             user = f(sol, Gdb))
41   Kt <- Fd@Kt
42   Fd <- Fd@F
43   Gdb <- Fd + Gd
44   Gdb <- Gd + Gdb
45 }
46 ## the Direct and Diffuse data is given
47 else {
48   Gdb <- getData(Gdb)$Gdb
49   Gdb <- getData(Gdb)[["Bsd"]]
50   Gdb <- getData(Gdb)[["Bsd"]]
51   Fd <- Gdb
52 }
53 }
54
55 result <- data.table(Dates = index0(sol), Fd, Kt, Gdb = Gd, DOD, BOD)
56 setkey(result, "Dates")
57
58 result
59 }
60
61 lat import
62 }
```

FIGURA 6.1: Pestaña blame de GitHub de la función `fCompD`.

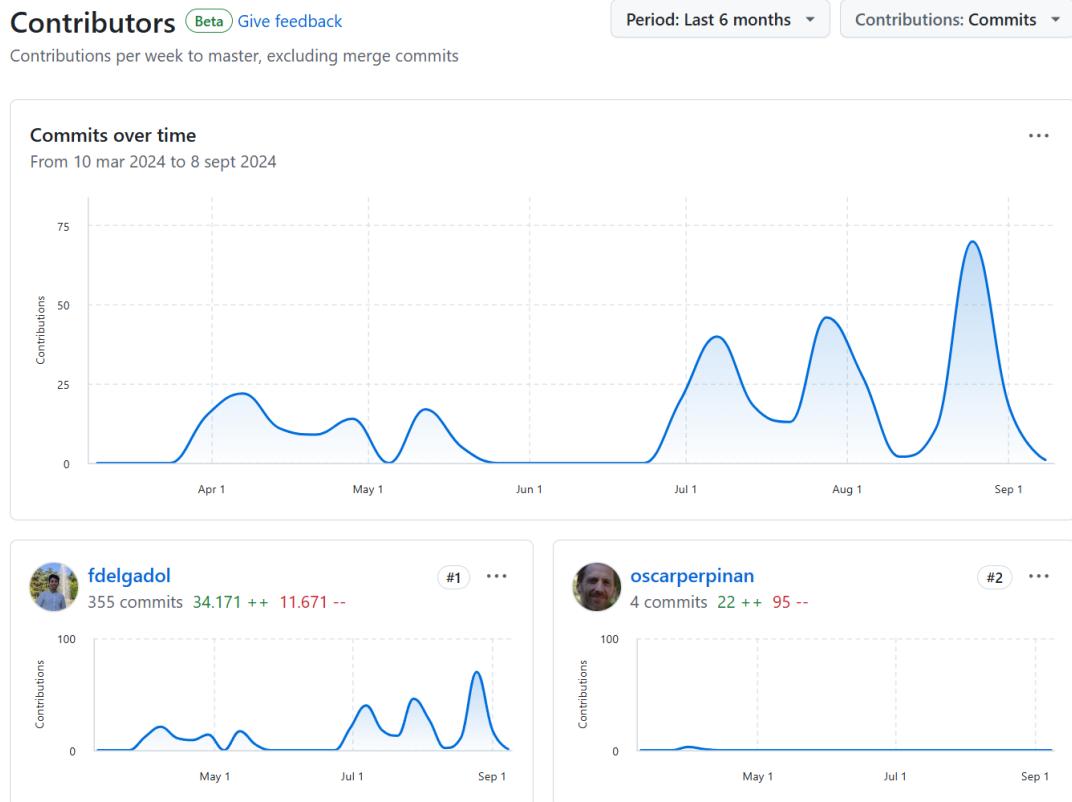


FIGURA 6.2: Sección contributors de GitHub para los últimos 6 meses.

buciones individuales de los miembros del equipo. Con ella, se puede ver el número de líneas añadidas y eliminadas y el número de *commits* realizado por cada miembro (6.2).

Todas estas herramientas dejan ver la magnitud de los cambios que se han producido en este proyecto en comparación con el anterior.

6.2. Comparación de resultados

Como se ve en el capítulo 5, los resultados son idénticos a los que ofrecía **solaR**. Sin embargo, el número de funciones es mayor, lo que permite calcular datos aislados facilitando mucho las labores de investigación. Además, los tiempos de ejecución y el uso de memoria han disminuido mucho.

```

1 lat <- 37.2;
2
3 G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,
4      2814, 2179)
5
6 Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,
7      17.2, 15.2)
8
9 prom <- list(G0dm = G0dm, Ta = Ta)
10
11 profvis({prodFixed <- prodGCPV(lat = lat, dataRad = prom,
12           keep.night = FALSE)})
```

¹Un *commit* es una operación que guarda los cambios realizados en el repositorio. Funciona como un punto de control en el historial del proyecto.

6. CONCLUSIONES

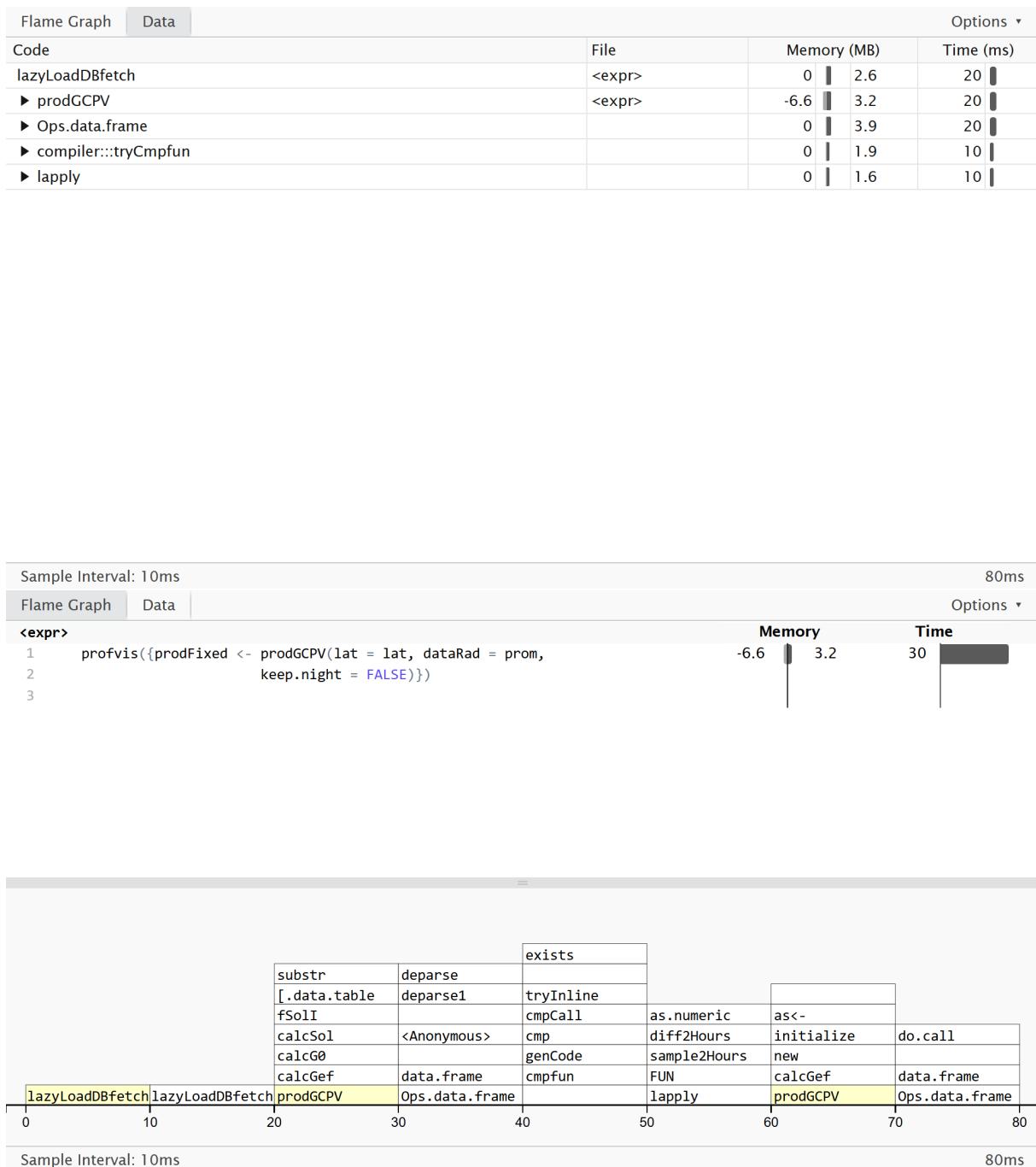


FIGURA 6.3: Resultado de rendimiento de la función `prodGCPV` obtenidos con la función `profvis`. Arriba, la pestaña Data; y abajo, la pestaña Flame Graph

6.3. Desarrollo futuro

Pese a que **solar2** ya es un paquete muy capaz, aún quedan varios puntos a mejorar:

- **Interfaz de usuario:** el lenguaje R es un lenguaje orientado a objetos, por lo que un paquete basado en este, puede entorpecer a los usuarios menos entendidos en este lenguaje. Por ello, cabe la posibilidad de añadir al paquete una interfaz de usuario que permita a los usuarios introducir sus datos de forma guiada.

- **Mejora de funciones:** muchas funciones pueden seguir desarrollándose, mejorando su eficiencia o simplemente incluyendo nuevas funcionalidades.
- **Toma de datos:** Aunque **solaR2** ya cuente con la función **readSIAR**, mejoraría notablemente incorporando una función que sea capaz de obtener datos meteorológicos de forma ilimitada (la función **readSIAR** tiene un límite de 100 registros por minuto).
- **Uso de paquetes especializados en datos espaciales:** Paquetes como **terra** [Hij24] podrían ser útiles para aumentar la eficiencia y rapidez de **solaR2**.

Bibliografía

- [LJ60] B. Y. H. Liu y R. C. Jordan. “The interrelationship and characteristic distribution of direct, diffuse, and total solar radiation”. En: *Solar Energy* 4 (1960), págs. 1-19.
- [Pag61] J. K. Page. “The calculation of monthly mean solar radiation for horizontal and inclined surfaces from sunshine records for latitudes 40N-40S”. En: *U.N. Conference on New Sources of Energy*. Vol. 4. 98. 1961, págs. 378-390.
- [Coo69] P.I. Cooper. “The Absorption of Solar Radiation in Solar Stills”. En: *Solar Energy* 12 (1969).
- [Spe71] J.W. Spencer. “Fourier Series Representation of the Position of the Sun”. En: 2 (1971). URL: <http://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html>.
- [CR79] M. Collares-Pereira y Ari Rabl. “The average distribution of solar radiation: correlations between diffuse and hemispherical and between daily and hourly insolation values”. En: *Solar Energy* 22 (1979), págs. 155-164.
- [Sta85] Richard Stallman. *GNU Emacs*. Un editor de texto extensible, personalizable, auto-documentado y en tiempo real. 1985. URL: <https://www.gnu.org/software/emacs/>.
- [Mic88] Joseph J. Michalsky. “The Astronomical Almanac’s algorithm for approximate solar position (1950-2050)”. En: *Solar Energy* 40.3 (1988), págs. 227-235. ISSN: 0038-092X. DOI: [DOI:10.1016/0038-092X\(88\)90045-X](https://doi.org/10.1016/0038-092X(88)90045-X).
- [RBD90] D.T. Reindl, W.A. Beckman y J.A. Duffie. “Evaluation of hourly tilted surface radiation models”. En: *Solar Energy* 45.1 (1990), págs. 9-17. ISSN: 0038-092X. DOI: [https://doi.org/10.1016/0038-092X\(90\)90061-G](https://doi.org/10.1016/0038-092X(90)90061-G). URL: <https://www.sciencedirect.com/science/article/pii/0038092X9090061G>.
- [Pan+91] D. Panico et al. “Backtracking: a novel strategy for tracking PV systems”. En: *IEEE Photovoltaic Specialists Conference*. 1991, págs. 668-673.
- [Dom+03] Carsten Dominik et al. *Org Mode*. Un sistema de organización de notas, planificación de proyectos y autoría de documentos con una interfaz de texto plano. 2003. URL: <https://orgmode.org>.
- [ZG05] Achim Zeileis y Gabor Grothendieck. “zoo: S3 Infrastructure for Regular and Irregular Time Series”. En: *Journal of Statistical Software* 14.6 (2005), págs. 1-27. DOI: [10.18637/jss.v014.i06](https://doi.org/10.18637/jss.v014.i06).
- [Sar08] Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. New York: Springer, 2008. ISBN: 978-0-387-75968-5. URL: <http://lmdvr.r-forge.r-project.org>.
- [Str11] L. Strous. *Position of the Sun*. 2011. URL: <http://aa.quae.nl/en/reken/zonpositie.html>.

- [Per12] Oscar Perpiñán. “solaR: Solar Radiation and Photovoltaic Systems with R”. En: *Journal of Statistical Software* 50.9 (2012), págs. 1-32. DOI: [10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09).
- [Adr+17] T. Adrada Guerra et al. “Comparative Energy Performance Analysis of Six Primary Photovoltaic Technologies in Madrid (Spain)”. En: *Energies* 10.6 (2017), pág. 772. DOI: [10.3390/en10060772](https://doi.org/10.3390/en10060772). URL: <https://doi.org/10.3390/en10060772>.
- [JG20] Michael Schmutz Jan Remund Stefan Müller y Pascal Graf. *Meteonorm Version 8*. Versión 8.0. Meteotest. Berna, Suiza, 2020. URL: <https://meteonorm.com>.
- [Uni20] European Union. *NextGenerationEU*. 2020. URL: https://next-generation-eu.europa.eu/index_es.
- [BOE22a] BOE. *Real Decreto-ley 10/2022, de 13 de mayo, por el que se establece con carácter temporal un mecanismo de ajuste de costes de producción para la reducción del precio de la electricidad en el mercado mayorista*. 2022. URL: <https://www.boe.es/buscar/act.php?id=BOE-A-2022-7843>.
- [BOE22b] BOE. *Real Decreto-ley 6/2022, de 29 de marzo, por el que se adoptan medidas urgentes en el marco del Plan Nacional de respuesta a las consecuencias económicas y sociales de la guerra en Ucrania*. 2022. URL: <https://www.boe.es/buscar/doc.php?id=BOE-A-2022-4972>.
- [dem22] Ministerio para transición ecológica y el reto demográfico. *Plan + Seguridad Energética*. 2022. URL: <https://www.miteco.gob.es/es/ministerio/planes-estrategias/seguridad-energetica.html#planSE>.
- [Eur22] Consejo Europeo. *REPowerEU*. 2022. URL: <https://www.consilium.europa.eu/es/policies/eu-recovery-plan/repowereu/>.
- [Hac22] Ministerio de Hacienda. *Mecanismo de Recuperación y Resiliencia*. 2022. URL: <https://www.hacienda.gob.es/es-ES/CDI/Paginas/FondosEuropeos/Fondos-relacionados-COVID/MRR.aspx>.
- [Mer+23] Olaf Mersmann et al. *microbenchmark: Accurate Timing Functions*. Proporciona infraestructura para medir y comparar con precisión el tiempo de ejecución de las expresiones de R. 2023. URL: <https://github.com/joshuaulrich/microbenchmark>.
- [Min23] pesca y alimentación Ministerio de agricultura. *Sistema de Información Agroclimática para el Regadío*. 2023. URL: <https://servicio.mapa.gob.es/websiar/>.
- [Per23] O. Perpiñán. *Energía Solar Fotovoltaica*. 2023. URL: <https://oscarperpinan.github.io/esf/>.
- [R C23] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: <https://www.R-project.org/>.
- [UNE23] UNEF. “Fomentando la biodiversidad y el crecimiento sostenible”. En: *Informe anual UNEF* (2023). URL: <https://www.unef.es/es/recursos-informes?idMultimediaCategoria=18>.
- [Wan+23] Chris Wanstrath et al. *GitHub*. 2023. URL: <https://github.com/>.
- [SAM24] System Advisor Model (SAM). *SAM: System Advisor Model*. <https://sam.nrel.gov/>. 2024.
- [Bar+24] Tyson Barrett et al. *data.table: Extension of ‘data.frame’*. R package version 1.15.99, <https://Rdatatable.gitlab.io/data.table>, <https://github.com/Rdatatable/data.table>. 2024. URL: <https://r-datable.com>.

- [Hij24] Robert J. Hijmans. *terra: Spatial Data Analysis*. R package version 1.7-78. 2024. URL: <https://CRAN.R-project.org/package=terra>.
- [Nat24] National Renewable Energy Laboratory. *Best Research-Cell Efficiency Chart*. <https://www.nrel.gov/pv/cell-efficiency.html>. 2024.
- [Pro24] ESS Project. *Emacs Speaks Statistics (ESS)*. Un paquete adicional para GNU Emacs diseñado para apoyar la edición de scripts y la interacción con varios programas de análisis estadístico. 2024. URL: <https://ess.r-project.org/>.
- [PVG24] PVGIS. *PVGIS: Photovoltaic Geographical Information System*. <https://ec.europa.eu/jrc/en/pvgis>. 2024.
- [PVS24] PVsyst. *PVsyst: Software for Photovoltaic Systems*. <https://www.pvsyst.com/>. 2024.
- [R C24] R Core Team. *The Comprehensive R Archive Network (CRAN)*. <https://cran.r-project.org/>. R package repository. R Foundation for Statistical Computing, Vienna, Austria. 2024.
- [Sis24] Sisifo. *Sisifo: Solar Energy Simulation Software*. <https://www.sisifo.org/>. 2024.
- [Wic+24] H. Wickham et al. *profvis: Interactive Visualizations for Profiling R Code*. R package version 0.3.8.9000. 2024. URL: <https://github.com/rstudio/profvis>.

APÉNDICE A

Manual de referencia de solaR2

En este apéndice se incluye el manual de referencia del paquete solaR2. Este manual se genera en base a los archivos de documentación (**.Rd**) propios de un paquete de R, y en el cual se recoge la información de todas las funciones, objetos y set de datos que contiene el paquete.

Se distribuye siguiendo la siguiente nomenclatura:

- **Constructores:** se trata de funciones que devuelven un objeto de una clase propia del paquete. Como identificador, se añade la letra **A** antes del nombre.
- **Clases:** la definición de las clases de los objetos definidos por este paquete. Como identificador, se añade la letra **B** antes del nombre.
- **Utilidades:** funciones que sirven de apoyo a los cálculos de las funciones constructoras. Como identificador, se añade la letra **C** antes del nombre.
- **Métodos:** métodos para los objetos definidos en el paquete. Como identificador, se añade la letra **D** antes del nombre.

Package ‘solaR2’

September 8, 2024

Type Package

Title Radiation and Photovoltaic Systems

Version 0.10

Encoding UTF-8

Description Calculation methods of solar radiation and performance of photovoltaic systems from daily and intradaily irradiation data sources.

URL <https://solarization.github.io/solaR2/>

BugReports <https://github.com/solarization/solaR2/issues>

License GPL-3

LazyData yes

Depends R (>= 4.0.0), data.table, lattice, latticeExtra

Imports RColorBrewer, graphics, grDevices, stats, methods, utils

Suggests zoo, sp, raster, rasterVis, tdr, meteoForecast, htr2,
jsonlite, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Oscar Perpiñán-Lamigueiro [aut]
(<<https://orcid.org/0000-0002-4134-7196>>),
Francisco Delgado-López [aut, cre]

Maintainer Francisco Delgado-López <f.delgadol@alumnos.upm.es>

Contents

solaR2-package	3
A1_calcSol	5
A2_calcG0	6
A3_calcGef	9
A4_prodGCPV	11
A5_prodPVPS	15
A6_calcShd	17
A7_optimShd	18
A8_Meteo2Meteo	22
A8_readBD	23
A8_readG0dm	25

A8_readSIAR	26
B1_Meteo-class	27
B2_Sol-class	28
B3_G0-class	29
B4_Gef-class	30
B5_ProdGCPV-class	32
B6_ProdPVPS-class	33
B7_Shade-class	34
C_corrFdKt	36
C_fBTd	38
C_fBTi	39
C_fCompD	40
C_fCompl	41
C_fInclin	43
C_fProd	45
C_fPump	47
C_fSolD	48
C_fSolI	50
C_fSombra	52
C_fTemp	54
C_fTheta	55
C_HQCurve	57
C_local2Solar	58
C_NmgPVPS	59
C_sample2Diff	61
C_solarAngles	62
C_utils-angle	64
C_utils-time	64
D_as.data.tableD-methods	65
D_as.data.tableI-methods	66
D_as.data.tableM-methods	68
D_as.data.tableY-methods	69
D_compare-methods	70
D_getData-methods	71
D_getG0-methods	71
D_getLat-methods	71
D_indexD-methods	72
D_indexI-methods	72
D_levelplot-methods	73
D_Losses-methods	73
D_mergesolaR-methods	74
D_shadeplot-methods	75
D_window-methods	75
D_writeSolar-methods	76
D_xyplot-methods	78
E_aguiar	79
E_helios	79
E_prodEx	80
E_pumpCoef	80
E_SIAR	81
E_solaR.theme	82

Description

The solaR2 package allows for reproducible research both for photovoltaics (PV) systems performance and solar radiation. It includes a set of classes, methods and functions to calculate the sun geometry and the solar radiation incident on a photovoltaic generator and to simulate the performance of several applications of the photovoltaic energy. This package performs the whole calculation procedure from both daily and intradaily global horizontal irradiation to the final productivity of grid-connected PV systems and water pumping PV systems.

Details

solaRd is designed using a set of S4 classes whose core is a group of slots with multivariate time series. The classes share a variety of methods to access the information and several visualization methods. In addition, the package provides a tool for the visual statistical analysis of the performance of a large PV plant composed of several systems.

Although solaRd is primarily designed for time series associated to a location defined by its latitude/longitude values and the temperature and irradiation conditions, it can be easily combined with spatial packages for space-time analysis.

Please note that this package needs to set the timezone to UTC. Every ‘data.table’ object created by the package will have an index with this time zone as a synonym of mean solar time..

You can check it after loading solaR2 with:

```
Sys.getenv('TZ')
```

If you need to change it, use:

```
Sys.setenv(TZ = 'YourTimeZone')
```

Index of functions and classes:

G0-class	Class "G0": irradiation and irradiance on the horizontal plane.
Gef-class	Class "Gef": irradiation and irradiance on the generator plane.
HQCurve	H-Q curves of a centrifugal pump
Meteo-class	Class "Meteo"
NmgPVPS	Nomogram of a photovoltaic pumping system
ProdGCPV-class	Class "ProdGCPV": performance of a grid connected PV system.
ProdPVPS-class	Class "ProdPVPS": performance of a PV pumping system.
Shade-class	Class "Shade": shadows in a PV system.
Sol-class	Class "Sol": Apparent movement of the Sun from the Earth
aguiar	Markov Transition Matrices for the Aguiar et al. procedure
as.data.tableD	Methods for Function as.data.frameD
as.data.tableI	Methods for Function as.data.frameI
as.data.tableM	Methods for Function as.data.frameM
as.data.tableY	Methods for Function as.data.frameY

calcG0	Irradiation and irradiance on the horizontal plane.
calcGef	Irradiation and irradiance on the generator plane.
calcShd	Shadows on PV systems.
calcSol	Apparent movement of the Sun from the Earth
compare	Compare G0, Gef and ProdGCPV objects
compareLosses	Losses of a GCPV system
corrFdKt	Correlations between the fraction of diffuse irradiation and the clearness index.
d2r	Conversion between angle units.
diff2Hours	Small utilities for difftime objects.
fBTd	Daily time base
fCompD	Components of daily global solar irradiation on a horizontal surface
fCompI	Calculation of solar irradiance on a horizontal surface
fInclin	Solar irradiance on an inclined surface
fProd	Performance of a PV system
fPump	Performance of a centrifugal pump
fSolD	Daily apparent movement of the Sun from the Earth
fSolI	Instantaneous apparent movement of the Sun from the Earth
fSombra	Shadows on PV systems
fTemp	Intradaily evolution of ambient temperature
fTheta	Angle of incidence of solar irradiation on a inclined surface
getData	Methods for function getData
getG0	Methods for function getG0
getLat	Methods for Function getLat
helios	Daily irradiation and ambient temperature from the Helios-IES database
hour	Utilities for time indexes.
indexD	Methods for Function indexD
indexI	Methods for Function indexI
levelplot-methods	Methods for function levelplot.
local2Solar	Local time, mean solar time and UTC time zone.
mergesolaR	Merge solaR objects
optimShd	Shadows calculation for a set of distances between elements of a PV grid connected plant.
prodEx	Productivity of a set of PV systems of a PV plant.
prodGCPV	Performance of a grid connected PV system.
prodPVPS	Performance of a PV pumping system
pumpCoef	Coefficients of centrifugal pumps.
readBD	Daily or intradaily values of global horizontal irradiation and ambient temperature from a local file or a data.frame.
readG0dm	Monthly mean values of global horizontal irradiation.
readSIAR	Meteorological data exported from the SIAR network

A1_calcSol

5

shadeplot	Methods for Function shadeplot
solaR.theme	solaR theme
window	Methods for extracting a time window
writeSolar	Exporter of solaR results
xyplot-methods	Methods for function xyplot in Package 'solaR'

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

A1_calcSol*Apparent movement of the Sun from the Earth***Description**Compute the apparent movement of the Sun from the Earth with the functions [fSolD](#) and [fSolI](#).**Usage**

```
calcSol(lat, BTd, sample = 'hour', BTi,
       EoT = TRUE, keep.night = TRUE,
       method = 'michalsky')
```

Arguments

lat	Latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
BTd	Daily time base, a POSIXct object which may be the result of fBTd . It is not considered if BTi is provided.
sample	Increment of the intradaily sequence. It is a character string, containing one of "sec", "min", "hour". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s". It is used by seq.POSIXt . It is not considered if BTi is provided.
BTi	Intradaily time base, a POSIXct object to be used by fSolI . It may be the result of fBTi .
EoT	logical, if TRUE the Equation of Time is used. Default is TRUE.
keep.night	logical, if TRUE (default) the night is included in the time series.
method	character, method for the sun geometry calculations to be chosen from 'cooper', 'spencer', 'michalsky' and 'strous'. See references for details.

ValueA [Sol-class](#) object.**Author(s)**

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Cooper, P.I., Solar Energy, 12, 3 (1969). "The Absorption of Solar Radiation in Solar Stills"
- Spencer, Search 2 (5), 172, <https://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html>
- Strous: <https://www.aa.quae.nl/en/reken/zonpositie.html>
- Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:10.18637/jss.v050.i09

Examples

```
BTd = fBTd(mode = 'serie')

lat = 37.2
sol = calcSol(lat, BTd[100])
print(as.data.tableD(sol))

library(lattice)
xyplot(as.data.tableI(sol))

solStrous = calcSol(lat, BTd[100], method = 'strous')
print(as.data.tableD(solStrous))

solSpencer = calcSol(lat, BTd[100], method = 'spencer')
print(as.data.tableD(solSpencer))

solCooper = calcSol(lat, BTd[100], method = 'cooper')
print(as.data.tableD(solCooper))
```

A2_calcG0

Irradiation and irradiance on the horizontal plane.

Description

This function obtains the global, diffuse and direct irradiation and irradiance on the horizontal plane from the values of *daily* and *intradaily* global irradiation on the horizontal plane. It makes use of the functions `calcSol`, `fCompD`, `fCompI`, `fBTd` and `readBDd` (or equivalent).

Besides, if information about maximum and minimum temperatures values are available it obtains a series of temperature values with `fTemp`.

Usage

```
calcG0(lat, modeRad = 'prom', dataRad,
       sample = 'hour', keep.night = TRUE,
       sunGeometry = 'michalsky',
       corr, f, ...)
```

Arguments

lat	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
modeRad	A character string, describes the kind of source data of the global irradiation and ambient temperature. It can be modeRad = 'prom' for monthly mean calculations. With this option, a set of 12 values inside dataRad must be provided, as defined in readG0dm . modeRad = 'aguiar' uses a set of 12 monthly average values (provided with dataRad) and produces a synthetic daily irradiation time series following the procedure by Aguiar et al. (see reference below). If modeRad = 'bd' the information of <i>daily</i> irradiation is read from a file, a data.table defined by dataRad, a zoo or a Meteo object. (See readBDd , dt2Meteo and zoo2Meteo for details). If modeRad = 'bdI' the information of <i>intradaily</i> irradiation is read from a file, a data.table defined by dataRad, a zoo or a Meteo object. (See readBDi , dt2Meteo and zoo2Meteo for details).
dataRad	<ul style="list-style-type: none"> • If modeRad = 'prom' or modeRad = 'aguiar', a numeric with 12 values or a named list whose components will be processed with readG0dm. • If modeRad = 'bd' a character (name of the file to be read with readBDd), a data.table (to be processed with dt2Meteo), a zoo (to be processed with zoo2Meteo), a Meteo object, or a list as defined by readBDd, dt2Meteo or zoo2Meteo. The resulting object will include a column named Ta, with information about ambient temperature. • If modeRad = 'bdI' a character (name of the file to be read with readBDi), a data.table (to be processed with dt2Meteo), a zoo (to be processed with zoo2Meteo), a Meteo object, or a list as defined by readBDi, dt2Meteo or zoo2Meteo. The resulting object will include a column named Ta, with information about ambient temperature.
sample	character, containing one of "sec", "min", "hour". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s" (used by seq.POSIXt). It is not used when modeRad = "bdI".
keep.night	logical. When it is TRUE (default) the time series includes the night.
sunGeometry	character, method for the sun geometry calculations. See calcSol , fSolD and fSolI .
corr	A character, the correlation between the fraction of diffuse irradiation and the clearness index to be used. With this version several options are available, as described in corrFdKt . For example, the FdKtPage is selected with corr = 'Page' while the FdKtCPR with corr = 'CPR'. If corr = 'user' the use of a correlation defined by a function f is possible. If corr = 'none' the object defined by dataRad should include information about global, diffuse and direct daily irradiation with columns named G0d, D0d and B0d, respectively (or G0, D0 and B0 if modeRad = 'bdI'). If corr is missing, then it is internally set to CPR when modeRad = 'bd', to Page when modeRad = 'prom' and to BRL when modeRad = 'bdI'.
f	A function defininig a correlation between the fraction of diffuse irradiation and the clearness index. It is only neccessary when corr = 'user'
...	Additional arguments for fCompD or fCompI

Value

A G_0 object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)
- Aguiar, Collares-Pereira and Conde, "Simple procedure for generating sequences of daily radiation values using a library of Markov transition matrices", Solar Energy, Volume 40, Issue 3, 1988, Pages 269–279

See Also

[calcSol](#), [fCompD](#), [fCompI](#), [readG0dm](#), [readBDd](#), [readBDi](#), [dt2Meteo](#), [corrFdKt](#).

Examples

```
G0dm = c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742, 7.919, 7.027, 5.369, 3.562, 2.814, 2.179)*1000;
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2,
15.2)

g0 <- calcG0(lat = 37.2, modeRad = 'prom', dataRad = list(G0dm = G0dm, Ta = Ta))
print(g0)
xyplot(g0)

## Aguiar et al.

g0 <- calcG0(lat = 37.2, modeRad = 'aguiar', dataRad = G0dm)
print(g0)
xyplot(g0)

##Now the G0I component of g0 is used as
##the bdI argument to calcG0 in order to
##test the intradaily correlations of fd-kt

BDi = as.data.tableI(g0)
BDi$Ta = 25 ##Information about temperature must be contained in BDi

g02 <- calcG0(lat = 37.2,
               modeRad = 'bdI',
               dataRad = list(lat = 37.2, file = BDi),
               corr = 'none')

print(g02)

g03 <- calcG0(lat = 37.2,
               modeRad = 'bdI',
               dataRad = list(lat = 37.2, file = BDi),
               corr = 'BRL')
print(g03)
```

```
xyplot(Fd ~ Kt, data = g03, pch = 19, alpha = 0.3)
```

A3_calcGef*Irradiation and irradiance on the generator plane.***Description**

This function obtains the global, diffuse and direct irradiation and irradiance on the generator plane from the values of *daily* or *intradaily* global irradiation on the horizontal plane. It makes use of the functions [calcG0](#), [fTheta](#), [fInclin](#). Besides, it can calculate the shadows effect with the [calcShd](#) function.

Usage

```
calcGef(lat,
        modeTrk = 'fixed',
        modeRad = 'prom',
        dataRad,
        sample = 'hour',
        keep.night = TRUE,
        sunGeometry = 'michalsky',
        corr, f,
        betaLim = 90, beta = abs(lat)-10, alpha = 0,
        iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
        modeShd = '',
        struct = list(),
        distances = data.table(),
        ...)
```

Arguments

<code>lat</code>	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
<code>modeTrk</code>	character, to be chosen from 'fixed', 'two' or 'horiz'. When <code>modeTrk</code> = 'fixed' the surface is fixed (inclination and azimuth angles are constant). The performance of a two-axis tracker is calculated with <code>modeTrk</code> = 'two', and <code>modeTrk</code> = 'horiz' is the option for an horizontal N-S tracker. Its default value is <code>modeTrk</code> = 'fixed'
<code>modeRad, dataRad</code>	Information about the source data of the global irradiation. See calcG0 for details.
<code>sample, keep.night</code>	See calcSol for details.
<code>sunGeometry</code>	character, method for the sun geometry calculations. See calcSol , fSolD and fSolI .
<code>corr, f</code>	See calcG0 for details.
<code>beta</code>	numeric, inclination angle of the surface (degrees). It is only needed when <code>modeTrk</code> = 'fixed'.

betaLim	numeric, maximum value of the inclination angle for a tracking surface. Its default value is 90 (no limitation)
alpha	numeric, azimuth angle of the surface (degrees). It is measured from the south ($\alpha = 0$), and it is negative to the east and positive to the west. It is only needed when modeTrk = 'fixed'. Its default value is alpha = 0
iS	integer, degree of dirtiness. Its value must be included in the set (1,2,3,4). iS = 1 corresponds to a clean surface while iS = 4 is the selection for a dirty surface. Its default value is 2.
alb	numeric, albedo reflection coefficient. Its default value is 0.2
modeShd, struct, distances	See calcShd for details.
horizBright	logical, if TRUE, the horizon brightness correction proposed by Reind et al. is used.
HCPV	logical, if TRUE the diffuse and albedo components of the <i>effective</i> irradiance are set to zero. HCPV is the acronym of High Concentration PV system.
...	Additional arguments for calcSol and calcG0

Value

A Gef object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Hay, J. E. and McKay, D. C.: Estimating Solar Irradiance on Inclined Surfaces: A Review and Assessment of Methodologies. *Int. J. Solar Energy*, (3):pp. 203, 1985.
- Martin, N. and Ruiz, J.M.: Calculation of the PV modules angular losses under field conditions by means of an analytical model. *Solar Energy Materials & Solar Cells*, 70:25–38, 2001.
- D. T. Reindl and W. A. Beckman and J. A. Duffie: Evaluation of hourly tilted surface radiation models, *Solar Energy*, 45:9-17, 1990.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[calcG0](#), [fTheta](#), [fInclin](#), [calcShd](#).

Examples

```
lat <- 37.2

###12 Average days.

G0dm = c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742, 7.919, 7.027, 5.369,
       3.562, 2.814, 2.179)*1000;
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2,
      15.2)
```

```
##Fixed surface, default values of inclination and azimuth.

gef <- calcGef(lat = lat, modeRad = 'prom', dataRad = list(G0dm = G0dm, Ta = Ta))
print(gef)
xyplot(gef)

##Two-axis surface, no limitation angle.

gef2 <- calcGef(lat = lat, modeRad = 'prom',
                  dataRad = list(G0dm = G0dm, Ta = Ta),
                  modeTrk = 'two')
print(gef2)
xyplot(gef2)

struct = list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 8)
distances = data.table(Lew = 40, Lns = 30, H = 0)

gefShd <- calcGef(lat = lat, modeRad = 'prom',
                     dataRad = list(G0dm = G0dm, Ta = Ta),
                     modeTrk = 'two',
                     modeShd = c('area', 'prom'),
                     struct = struct, distances = distances)
print(gefShd)
```

A4_prodGCPV

Performance of a grid connected PV system.

Description

Compute every step from solar angles to effective irradiance to calculate the performance of a grid connected PV system.

Usage

```
prodGCPV(lat,
          modeTrk = 'fixed',
          modeRad = 'prom',
          dataRad,
          sample = 'hour',
          keep.night = TRUE,
          sunGeometry = 'michalsky',
          corr, f,
          betaLim = 90, beta = abs(lat)-10, alpha = 0,
          iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
          module = list(),
          generator = list(),
          inverter = list(),
          effSys = list(),
          modeShd = '',
          struct = list(),
          distances = data.table(),
          ...)
```

Arguments

lat	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
modeTrk	A character string, describing the tracking method of the generator. See calcGef for details.
modeRad, dataRad	Information about the source data of the global irradiation. See calcG0 for details.
sample, keep.night	See calcSol for details.
sunGeometry	character, method for the sun geometry calculations. See calcSol , fSolD and fSolI .
corr, f	See calcG0 for details.
betaLim, beta, alpha, iS, alb, horizBright, HCPV	See calcGef for details.
module	<p>list of numeric values with information about the PV module,</p> <p>Vocn open-circuit voltage of the module at Standard Test Conditions (default value 57.6 volts.)</p> <p>Iscn short circuit current of the module at Standard Test Conditions (default value 4.7 amperes.)</p> <p>Vmn maximum power point voltage of the module at Standard Test Conditions (default value 46.08 amperes.)</p> <p>Imn Maximum power current of the module at Standard Test Conditions (default value 4.35 amperes.)</p> <p>Ncs number of cells in series inside the module (default value 96)</p> <p>Ncp number of cells in parallel inside the module (default value 1)</p> <p>CoefVT coefficient of decrement of voltage of each cell with the temperature (default value 0.0023 volts per celsius degree)</p> <p>TONC nominal operational cell temperature, celsius degree (default value 47).</p>
generator	<p>list of numeric values with information about the generator,</p> <p>Nms number of modules in series (default value 12)</p> <p>Nmp number of modules in parallel (default value 11)</p>
inverter	<p>list of numeric values with information about the DC/AC inverter,</p> <p>Ki vector of three values, coefficients of the efficiency curve of the inverter (default c(0.01, 0.025, 0.05)), or a matrix of nine values (3x3) if there is dependence with the voltage (see references).</p> <p>Pinv nominal inverter power (W) (default value 25000 watts.)</p> <p>Vmin, Vmax minimum and maximum voltages of the MPP range of the inverter (default values 420 and 750 volts)</p> <p>Gumb minimum irradiance for the inverter to start (W/m²) (default value 20 W/m²)</p>
effSys	<p>list of numeric values with information about the system losses,</p> <p>ModQual average tolerance of the set of modules (%), default value is 3</p> <p>ModDisp module parameter dispersion losses (%), default value is 2</p> <p>OhmDC Joule losses due to the DC wiring (%), default value is 1.5</p> <p>OhmAC Joule losses due to the AC wiring (%), default value is 1.5</p>

MPP average error of the MPP algorithm of the inverter (%), default value is 1
 TrafoMT losses due to the MT transformer (%), default value is 1
 Disp losses due to stops of the system (%), default value is 0.5
 modeShd, struct, distances
 See [calcShd](#) for details.
 ... Additional arguments for [calcG0](#) or [calcGef](#)

Details

The calculation of the irradiance on the horizontal plane is carried out with the function [calcG0](#). The transformation to the inclined surface makes use of the [fTheta](#) and [fInclin](#) functions inside the [calcGef](#) function. The shadows are computed with [calcShd](#) while the performance of the PV system is simulated with [fProd](#).

Value

A ProdGCPV object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fProd](#), [calcGef](#), [calcShd](#), [calcG0](#), [compare](#), [compareLosses](#), [mergesolaR](#)

Examples

```
library(lattice)
library(latticeExtra)

lat <- 37.2;

G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,
         2814, 2179)

Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,
       17.2, 15.2)

prom <- list(G0dm = G0dm, Ta = Ta)

###Comparison of different tracker methods
prodFixed <- prodGCPV(lat = lat, dataRad = prom,
                        keep.night = FALSE)

prod2x <- prodGCPV(lat = lat, dataRad = prom,
                      modeTrk = 'two',
                      keep.night = FALSE)
```

```

prodHoriz <- prodGCPV(lat = lat,dataRad = prom,
                       modeTrk = 'horiz',
                       keep.night = FALSE)

##Comparison of yearly productivities
compare(prodFixed, prod2x, prodHoriz)
compareLosses(prodFixed, prod2x, prodHoriz)

##Comparison of power time series
ComparePac <- data.table(Dates = indexI(prod2x),
                           two = as.data.tableI(prod2x)$Pac,
                           horiz = as.data.tableI(prodHoriz)$Pac,
                           fixed = as.data.tableI(prodFixed)$Pac)

AngSol <- as.data.tableI(as(prodFixed, 'Sol'))

ComparePac <- merge(AngSol, ComparePac, by = 'Dates')

ComparePac[, Month := as.factor(month(Dates))]

xyplot(two + horiz + fixed ~ AzS|Month, data = ComparePac,
       type = 'l',
       auto.key = list(space = 'right',
                      lines = TRUE,
                      points = FALSE),
       ylab = 'Pac')

####Shadows
#Two-axis trackers
struct2x <- list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 8)
dist2x <- data.table(Lew = 40, Lns = 30, H = 0)
prod2xShd <- prodGCPV(lat = lat, dataRad = prom,
                        modeTrk = 'two',
                        modeShd = 'area',
                        struct = struct2x,
                        distances = dist2x)
print(prod2xShd)

#Horizontal N-S tracker
structHoriz <- list(L = 4.83);
distHoriz <- data.table(Lew = structHoriz$L*4);

#Without Backtracking
prodHorizShd <- prodGCPV(lat = lat, dataRad = prom,
                           sample = '10 min',
                           modeTrk = 'horiz',
                           modeShd = 'area', betaLim = 60,
                           distances = distHoriz,
                           struct = structHoriz)
print(prodHorizShd)

xyplot(r2d(Beta)~r2d(w),
       data = prodHorizShd,
       type = 'l',

```

```

main = 'Inclination angle of a horizontal axis tracker',
xlab = expression(omega (degrees)),
ylab = expression(beta (degrees)))

#With Backtracking
prodHorizBT <- prodGCPV(lat = lat, dataRad = prom,
                           sample = '10 min',
                           modeTrk = 'horiz',
                           modeShd = 'bt', betaLim = 60,
                           distances = distHoriz,
                           struct = structHoriz)

print(prodHorizBT)

xyplot(r2d(Beta)~r2d(w),
       data = prodHorizBT,
       type = 'l',
       main = 'Inclination angle of a horizontal axis tracker\n with backtracking',
       xlab = expression(omega (degrees)),
       ylab = expression(beta (degrees)))

compare(prodFixed, prod2x, prodHoriz, prod2xShd,
        prodHorizShd, prodHorizBT)

compareLosses(prodFixed, prod2x, prodHoriz, prod2xShd,
              prodHorizShd, prodHorizBT)

compareYf2 <- mergesolaR(prodFixed, prod2x, prodHoriz, prod2xShd,
                           prodHorizShd, prodHorizBT)

xyplot(prodFixed + prod2x +prodHoriz + prod2xShd + prodHorizShd + prodHorizBT ~ Dates,
       data = compareYf2, type = 'l', ylab = 'kWh/kWp',
       main = 'Daily productivity',
       auto.key = list(space = 'right'))

```

Description

Compute every step from solar angles to effective irradiance to calculate the performance of a PV pumping system.

Usage

```
prodPVPS(lat,
          modeTrk = 'fixed',
          modeRad = 'prom',
          dataRad,
          sample = 'hour',
          keep.night = TRUE,
          sunGeometry = 'michalsky',
          corr, f,
```

```
betaLim = 90, beta = abs(lat)-10, alpha = 0,
iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
pump , H,
Pg, converter= list(),
effSys = list(),
...)
```

Arguments

lat	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
modeTrk	A character string, describing the tracking method of the generator. See calcGef for details.
modeRad, dataRad	Information about the source data of the global irradiation. See calcG0 for details.
sample, keep.night	See calcSol for details.
sunGeometry	character, method for the sun geometry calculations. See calcSol , fSolD and fSolI .
corr, f	See calcG0 for details.
betaLim, beta, alpha, iS, alb, horizBright, HCPV	See calcGef for details.
pump	A list extracted from pumpCoef
H	Total manometric head (m)
Pg	Nominal power of the PV generator (Wp)
converter	list containing the nominal power of the frequency converter, Pnom, and Ki, vector of three values, coefficients of the efficiency curve.
effSys	list of numeric values with information about the system losses,
	ModQual average tolerance of the set of modules (%), default value is 3
	ModDisp module parameter dispersion losses (%), default value is 2
	OhmDC Joule losses due to the DC wiring (%), default value is 1.5
	OhmAC Joule losses due to the AC wiring (%), default value is 1.5
...	Additional arguments for calcSol , calcG0 and calcGef .

Details

The calculation of the irradiance on the generator is carried out with the function [calcGef](#). The performance of the PV system is simulated with [fPump](#).

Value

A [ProdPVPS](#) object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Abella, M. A., Lorenzo, E. y Chenlo, F.: PV water pumping systems based on standard frequency converters. *Progress in Photovoltaics: Research and Applications*, 11(3):179–191, 2003, ISSN 1099-159X.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[NmgPVPS](#), [fPump](#), [pumpCoef](#)

A6_calcShd

Shadows on PV systems.

Description

Compute the irradiance and irradiation including shadows for two-axis and horizontal N-S axis trackers and fixed surfaces. It makes use of the function [fSombra](#) for the shadows factor calculation. It is used by the function [calcGef](#).

Usage

```
calcShd(radEf,
        modeShd = '',
        struct = list(),
        distances = data.table())
```

Arguments

radEf	A Gef object. It may be the result of the calcGef function.
modeShd	character, defines the type of shadow calculation. In this version of the package the effect of the shadow is calculated as a proportional reduction of the circumsolar diffuse and direct irradiances. This type of approach is selected with modeShd = 'area'. In future versions other approaches which relate the geometric shadow and the electrical connections of the PV generator will be available. If radEf@modeTrk = 'horiz' it is possible to calculate the effect of backtracking with modeShd = 'bt'. If modeShd = c('area', 'bt') the backtracking method will be carried out and therefore no shadows will appear. Finally, for two-axis trackers it is possible to select modeShd = 'prom' in order to calculate the effect of shadows on an average tracker (see fSombra6). The result will include three variables (Gef0, Def0 and Bef0) with the irradiance/irradiation without shadows as a reference.
struct	list. When radEf@modeTrk = 'fixed' or modeTrk = 'horiz' only a component named L, which is the height (meters) of the tracker, is needed. For two-axis trackers (radEf@modeTrk = 'two'), an additional component named W, the width of the tracker, is required. Moreover, only when radEf@modeTrk = 'two' two components named Nrow and Ncol are included under this list. These components define, respectively, the number of rows and columns of the whole set of two-axis trackers in the PV plant.

`distances` `data.frame`.

When `radEf@modeTrk = 'fixed'` it includes a component named D for the distance between fixed surfaces. An additional component named H can be included with the relative height between surfaces.

When `radEf@modeTrk = 'horiz'` it only includes a component named Lew, being the distance between horizontal NS trackers along the East-West direction.

When `radEf@modeTrk = 'two'` it includes a component named Lns being the distance between trackers along the North-South direction, a component named Lew, being the distance between trackers along the East-West direction and a (optional) component named H with the relative height between surfaces.

The distances, in meters, are defined between axis of the trackers.

Value

A Gef object including three additional variables (`Gef0`, `Def0` and `Bef0`) in the slots `GefI`, `GefD`, `Gefdm` and `Gefy` with the irradiance/irradiation without shadows as a reference.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

`calcG0`, `fTheta`, `fInclin`, `calcShd`.

A7_optimShd

Shadows calculation for a set of distances between elements of a PV grid connected plant.

Description

The optimum distance between trackers or static structures of a PV grid connected plant depends on two main factors: the ground requirement ratio (defined as the ratio of the total ground area to the generator PV array area), and the productivity of the system including shadow losses. Therefore, the optimum separation may be the one which achieves the highest productivity with the lowest ground requirement ratio.

However, this definition is not complete since the terrain characteristics and the costs of wiring or civil works could alter the decision. This function is a help for choosing this distance: it computes the productivity for a set of combinations of distances between the elements of the plant.

Usage

```
optimShd(lat,
         modeTrk = 'fixed',
         modeRad = 'prom',
         dataRad,
         sample = 'hour',
         keep.night = TRUE,
         sunGeometry = 'michalsky',
         betaLim = 90, beta = abs(lat)-10, alpha = 0,
         iS = 2, alb = 0.2, HCPV = FALSE,
         module = list(),
         generator = list(),
         inverter = list(),
         effSys = list(),
         modeShd = '',
         struct = list(),
         distances = data.table(),
         res = 2,
         prog = TRUE)
```

Arguments

lat	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
modeTrk	character, to be chosen from 'fixed', 'two' or 'horiz'. When modeTrk = 'fixed' the surface is fixed (inclination and azimuth angles are constant). The performance of a two-axis tracker is calculated with modeTrk = 'two', and modeTrk = 'horiz' is the option for an horizontal N-S tracker. Its default value is modeTrk = 'fixed'
modeRad, dataRad	Information about the source data of the global irradiation. See calcG0 for details. For this function the option modeRad = 'bdI' is not supported.
sample	character, containing one of "sec", "min", "hour". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s" (used by seq.POSIXt)
keep.night	logical When it is TRUE (default) the time series includes the night.
sunGeometry	character, method for the sun geometry calculations. See calcSol , fSolD and fSolI .
betaLim, beta, alpha, iS, alb, HCPV	See calcGef for details.
module	list of numeric values with information about the PV module, Vocn open-circuit voltage of the module at Standard Test Conditions (default value 57.6 volts.) Iscn short circuit current of the module at Standard Test Conditions (default value 4.7 amperes.) Vm _n maximum power point voltage of the module at Standard Test Conditions (default value 46.08 amperes.) Im _n Maximum power current of the module at Standard Test Conditions (default value 4.35 amperes.)

	Ncs	number of cells in series inside the module (default value 96)
	Ncp	number of cells in parallel inside the module (default value 1)
	CoefVT	coefficient of decrement of voltage of each cell with the temperature (default value 0.0023 volts per celsius degree)
	TONC	nominal operational cell temperature, celsius degree (default value 47).
generator		list of numeric values with information about the generator,
	Nms	number of modules in series (default value 12)
	Nmp	number of modules in parallel (default value 11)
inverter		list of numeric values with information about the DC/AC inverter,
	Ki	vector of three values, coefficients of the efficiency curve of the inverter (default c(0.01, 0.025, 0.05)), or a matrix of nine values (3x3) if there is dependence with the voltage (see references).
	Pinv	nominal inverter power (W) (default value 25000 watts.)
	Vmin, Vmax	minimum and maximum voltages of the MPP range of the inverter (default values 420 and 750 volts)
	Gumb	minimum irradiance for the inverter to start (W/m ²) (default value 20 W/m ²)
effSys		list of numeric values with information about the system losses,
	ModQual	average tolerance of the set of modules (%), default value is 3
	ModDisp	module parameter dispersion losses (%), default value is 2
	OhmDC	Joule losses due to the DC wiring (%), default value is 1.5
	OhmAC	Joule losses due to the AC wiring (%), default value is 1.5
	MPP	average error of the MPP algorithm of the inverter (%), default value is 1
	TrafoMT	losses due to the MT transformer (%), default value is 1
	Disp	losses due to stops of the system (%), default value is 0.5
modeShd		character, defines the type of shadow calculation. In this version of the package the effect of the shadow is calculated as a proportional reduction of the cir- cum solar diffuse and direct irradiances. This type of approach is selected with modeShd = 'area'. In future versions other approaches which relate the geo- metric shadow and the electrical connections of the PV generator will be avail- able. If modeTrk = 'horiz' it is possible to calculate the effect of backtracking with modeShd = 'bt'. If modeShd = c('area', 'bt') the backtracking method will be carried out and therefore no shadows will appear. Finally, for two-axis trackers it is possible to select modeShd = 'prom' in order to calculate the effect of shadows on an average tracker (see fSombra6). The result will include three variables (Gef0, Def0 and Bef0) with the irradiance/irradiation without shadows as a reference.
struct		list. When modeTrk = 'fixed' or modeTrk = 'horiz' only a component named L, which is the height (meters) of the tracker, is needed. For two-axis trackers (modeTrk = 'two'), an additional component named W, the width of the tracker, is required. Moreover, two components named Nrow and Ncol are included un- der this list. These components define, respectively, the number of rows and columns of the whole set of trackers in the PV plant.
distances		list, whose three components are vectors of length 2: Lew (only when modeTrk = 'horiz' or modeTrk = 'two'), minimum and max- imum distance (meters) between horizontal NS and two-axis trackers along the East-West direction.

Lns (only when modeTrk = 'two'), minimum and maximum distance (meters) between two-axis trackers along the North-South direction.

D (only when modeTrk = 'fixed'), minimum and maximum distance (meters) between fixed surfaces.

These distances, in meters, are defined between the axis of the trackers.

res numeric; optimShd constructs a sequence from the minimum to the maximum value of distances, with res as the increment, in meters, of the sequence.

prog logical, show a progress bar; default value is TRUE

Details

optimShd calculates the energy produced for every combination of distances as defined by distances and res. The result of this function is a [Shade-class](#) object. A method of shadeplot for this class is defined ([shadeplot-methods](#)), and it shows the graphical relation between the productivity and the distance between trackers or fixed surfaces.

Value

A [Shade](#) object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O.: Grandes Centrales Fotovoltaicas: producción, seguimiento y ciclo de vida. PhD Thesis, UNED, 2008. https://www.researchgate.net/publication/39419806_Grandes_Centrales_Fotovoltaicas_Produccion_Seguimiento_y_Ciclo_de_Vida.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[prodGCPV](#), [calcShd](#)

Examples

```
library(lattice)
library(latticeExtra)

lat = 37.2;
G0dm = c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562, 2814,
2179)
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom = list(G0dm = G0dm, Ta = Ta)

###Two-axis trackers
struct2x = list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 3)
dist2x = list(Lew = c(30, 45),Lns = c(20, 40))

ShdM2x <- optimShd(lat = lat, dataRad = prom, modeTrk = 'two',
```

```

modeShd = c('area','prom'),
distances = dist2x, struct = struct2x,
res = 5)

shadeplot(ShdM2x)

pLew = xyplot(Yf~GRR,data = ShdM2x,groups = factor(Lew),type = c('l','g'),
main = 'Productivity for each Lew value')
pLew+glayer(panel.text(x[1], y[1], group.value))

pLns = xyplot(Yf~GRR,data = ShdM2x,groups = factor(Lns),type = c('l','g'),
main = 'Productivity for each Lns value')
pLns+glayer(panel.text(x[1], y[1], group.value))

## 1-axis tracker with Backtracking
structHoriz = list(L = 4.83);
distHoriz = list(Lew = structHoriz$L * c(2,5));

Shd12HorizBT <- optimShd(lat = lat, dataRad = prom,
                           modeTrk = 'horiz',
                           betaLim = 60,
                           distances = distHoriz, res = 2,
                           struct = structHoriz,
                           modeShd = 'bt')

shadeplot(Shd12HorizBT)

xyplot(diff(Yf)~GRR[-1],data = Shd12HorizBT,type = c('l','g'))

###Fixed system
structFixed = list(L = 5);
distFixed = list(D = structFixed$L*c(1,3));
Shd12Fixed <- optimShd(lat = lat, dataRad = prom,
                        modeTrk = 'fixed',
                        distances = distFixed, res = 2,
                        struct = structFixed,
                        modeShd = 'area')
shadeplot(Shd12Fixed)

```

A8_Meteo2Meteo

Transformation of intradaily meteorological data into daily and daily into monthly data.

Description

Functions for the class Meteo that transforms an intradaily Meteo object into a daily and a daily into a monthly.

Usage

Meteoi2Meteod(G0i)

Meteod2Meteom(G0d)

A8_readBD

23

Arguments

G0i	A Meteo object with intradaily data
G0d	A Meteo object with daily data

Value

A Meteo object

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also[readBDd](#), [readG0dm](#), [readSIAR](#)**Examples**

```
G0dm = c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742, 7.919,
       7.027, 5.369, 3.562, 2.814, 2.179) * 1000;
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2,
      28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom = list(G0dm = G0dm, Ta = Ta)

g0 = calcG0(lat = 37.2, dataRad = prom, modeRad = 'aguiar')
G0i = as.data.tableI(g0)
G0i = dt2Meteo(G0i, lat = 37.2)
G0i

G0d = MeteoI2Meteod(G0i)
G0d

G0m = MeteoD2Meteom(G0d)
G0m
```

A8_readBD*Daily or intradaily values of global horizontal irradiation and ambient temperature from a local file or a data.frame.*

Description

Constructor for the class Meteo with values of *daily* or *intradaily* values of global horizontal irradiation and ambient temperature from a local file or a data.frame.

Usage

```
readBDd(file, lat,
        format = '%d/%m/%Y',
        header = TRUE, fill = TRUE, dec = '.', sep = ';',
        dates.col = 'Dates', ta.col = 'Ta',
        g0.col = 'G0', keep.cols = FALSE, ...)
```

```

readBDi(file, lat,
        format = '%d/%m/%Y %H:%M:%S',
        header = TRUE, fill = TRUE, dec = '.',
        sep = ';', dates.col = 'Dates', times.col,
        ta.col = 'Ta', g0.col = 'G0', keep.cols = FALSE, ...)

dt2Meteo(file, lat, source = '', type)

zoo2Meteo(file, lat, source = '')

```

Arguments

file	The name of the file (readBDd and readBDi), data.frame (or data.table) (dt2Meteo) or zoo (zoo2Meteo) which the data are to be read from. It should contain a column G0d with <i>daily</i> (readBDd) or G0 with <i>intradaily</i> (readBDi) values of global horizontal irradiation (Wh/m ²). It should also include a column named Ta with values of ambient temperature. However, if the object is only a vector with irradiation values, it will converted to a data.table with two columns named G0 and Ta (filled with constant values)
	If the Meteo object is to be used with calcG0 (or fCompD, fCompI) and the option corr = 'none' , the file/data.frame must include three columns named G0, B0 and D0 with values of global, direct and diffuse irradiation on the horizontal plane.
	Only for daily data: if the ambient temperature is not available, the file should include two columns named TempMax and TempMin with daily values of maximum and minimum ambient temperature, respectively (see fTemp for details).
header, fill, dec, sep	See fread
format	character string with the format of the dates or time index. (Default for daily time bases: %d/%m/%Y). (Default for intradaily time bases: %d/%m/%Y %H:%M:%S)
lat	numeric, latitude (degrees) of the location.
dates.col	character string with the name of the column which contains the dates of the time series.
times.col	character string with the name of the column which contains the time index of the series in case is in a different column than the dates.
source	character string with information about the source of the values. (Default: the name of the file).
ta.col, g0.col	character, the name of the columns with the information of ambient temperature and radiation in the provided file
keep.cols	If keep.cols=FALSE (default value), the Meteo object does not include the columns that are not important for the rest of operations
...	Arguments for fread
type	character, type of the data in dt2Meteo. To choose between 'prom', 'bd' and 'bdI'. If it is not provided, the function dt2Meteo calculate the type.

Value

A Meteo object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[fread](#), [readG0dm](#).

Examples

```
data(helios)
names(helios) = c('Dates', 'G0d', 'TempMax', 'TempMin')

bd = dt2Meteo(helios, lat = 41, source = 'helios-IES', type = 'bd')

getData(bd)

xyplot(bd)
```

A8_readG0dm

Monthly mean values of global horizontal irradiation.

Description

Constructor for the class Meteo with 12 values of monthly means of irradiation.

Usage

```
readG0dm(G0dm, Ta = 25, lat = 0,
         year= as.POSIXlt(Sys.Date())$year+1900,
         promDays = c(17,14,15,15,15,10,18,18,18,19,18,13),
         source = '')
```

Arguments

G0dm	numeric, 12 values of monthly means of daily global horizontal irradiation (Wh/m ²).
Ta	numeric, 12 values of monthly means of ambient temperature (degrees Celsius).
lat	numeric, latitude (degrees) of the location.
year	numeric (Default: current year).
promDays	numeric, set of the average days for each month.
source	character string with information about the source of the values.

Value

Meteo object

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also[readBDd](#)**Examples**

```
G0dm =
c(2.766,3.491,4.494,5.912,6.989,7.742,7.919,7.027,5.369,3.562,2.814,2.179) * 1000;
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
BD <- readG0dm(G0dm = G0dm, Ta = Ta, lat = 37.2)
print(BD)
getData(BD)
xyplot(BD)
```

A8_readSIAR

*Meteorological data from the SIAR network.***Description**

Download, interpolate and transform meteorological data fromm the SIAR network.

Usage

```
readSIAR(Lon = 0, Lat = 0,
         inicio = paste(year(Sys.Date())-1, '01-01', sep = '-'),
         final = paste(year(Sys.Date())-1, '12-31', sep = '-'),
         tipo = 'Mensuales', n_est = 3)
```

Arguments

Lon	numeric, longitude (degrees) of the location.
Lat	numeric, latitude (degrees) of the location.
inicio	character or Date, first day of the records.
final	character or Date, last day of the records.
tipo	character, tipe of the records. To choose between Mensuales, Semanales, Diarios, Horarios.
n_est	integer, select that number of stations closest to the given point and then perform an IDW (Inverse Distance Weighting) interpolation with these data.

Value

A Meteo object

Author(s)

Francisco Delgado López, Oscar Perpiñán Lamigueiro.

See Also[readG0dm](#), [readBDd](#)

Examples

```
library(httr2)
library(jsonlite)

SIAR = readSIAR(Lon = -3.603, Lat = 40.033,
## Aranjuez, Community of Madrid, Spain
      inicio = '2023-01-01',
      final = '2023-05-01',
      tipo = 'Mensuales', n_est = 3)
SIAR
```

B1_Meteo-class	<i>Class "Meteo"</i>
----------------	----------------------

Description

A class for meteorological data.

Objects from the Class

Objects can be created by the family of [readBDd](#) functions.

Slots

latm: Latitude (degrees) of the meteorological station or source of the data.
data: A `data.table` object with the time series of daily irradiation (G_0 , Wh/m²), the ambient temperature (Ta) or the maximum and minimum ambient temperature (TempMax and TempMin).
source: A character with a short description of the source of the data.
type: A character, `prom`, `bd` or `bdI` depending on the constructor.

Methods

getData `signature(object = "Meteo")`: extracts the data slot as a `data.table` object.
getG0 `signature(object = "Meteo")`: extracts the irradiation as vector.
getLat `signature(object = "Meteo")`: extracts the latitude value.
indexD `signature(object = "Meteo")`: extracts the index of the data slot.
xyplot `signature(x = "formula", data = "Meteo")`: plot the content of the object according to the `formula` argument.
xyplot `signature(x = "Meteo", data = "missing")`: plot the data slot using the `xyplot` method for `zoo` objects.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[readBDd](#), [readBDi](#), [zoo2Meteo](#), [dt2Meteo](#), [readG0dm](#),

B2_Sol-class*Class "Sol": Apparent movement of the Sun from the Earth*

Description

A class which describe the apparent movement of the Sun from the Earth.

Objects from the Class

Objects can be created by [calcSol](#).

Slots

lat: numeric, latitude (degrees) as defined in the call to [calcSol](#).

sold: Object of class "data.table" created by [fSolD](#).

solI: Object of class "data.table" created by [fSolI](#).

method: character, method for the sun geometry calculations.

sample: difftime, increment of the intradaily sequence.

Methods

as.data.tableD `signature(object = "Sol")`: conversion to a data.table with daily values.

as.data.tableI `signature(object = "Sol")`: conversion to a data.table with intradaily values.

getLat `signature(object = "Sol")`: latitude (degrees) as defined in the call to [calcSol](#).

indexD `signature(object = "Sol")`: index of the sold slot.

indexI `signature(object = "Sol")`: index of the solI object.

xyplot `signature(x = "formula", data = "Sol")`: displays the contents of a Sol object with the xyplot method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[G0](#), [Gef](#).

<i>B3_G0-class</i>	<i>Class "G0": irradiation and irradiance on the horizontal plane.</i>
--------------------	--

Description

This class contains the global, diffuse and direct irradiation and irradiance on the horizontal plane, and ambient temperature.

Objects from the Class

Objects can be created by the function [calcG0](#).

Slots

G0D: Object of class `data.table` created by [fCompD](#). It includes daily values of:

Fd: numeric, the diffuse fraction

Ktd: numeric, the clearness index

G0d: numeric, the global irradiation on a horizontal surface (Wh/m²)

D0d: numeric, the diffuse irradiation on a horizontal surface (Wh/m²)

B0d: numeric, the direct irradiation on a horizontal surface (Wh/m²)

G0I: Object of class `data.table` created by [fCompI](#). It includes values of:

kt: numeric, clearness index

G0: numeric, global irradiance on a horizontal surface, (W/m²)

D0: numeric, diffuse irradiance on a horizontal surface, (W/m²)

B0: numeric, direct irradiance on a horizontal surface, (W/m²)

G0dm: Object of class `data.table` with monthly mean values of daily irradiation.

G0y: Object of class `data.table` with yearly sums of irradiation.

Ta: Object of class `data.table` with intradaily ambient temperature values.

Besides, this class contains the slots from the [Sol](#) and [Meteo](#) classes.

Extends

Class "[Meteo](#)", directly. Class "[Sol](#)", directly.

Methods

as.data.tableD `signature(object = "G0")`: conversion to a `data.table` with daily values.

as.data.tableI `signature(object = "G0")`: conversion to a `data.table` with intradaily values.

as.data.tableM `signature(object = "G0")`: conversion to a `data.table` with monthly values.

as.data.tableY `signature(object = "G0")`: conversion to a `data.frame` with yearly values.

indexD `signature(object = "G0")`: index of the `solD` slot.

indexI `signature(object = "G0")`: index of the `solI` slot.

getLat `signature(object = "G0")`: latitude of the inherited [Sol](#) object.

xyplot `signature(x = "G0", data = "missing")`: display the time series of daily values of irradiation.

xyplot `signature(x = "formula", data = "G0")`: displays the contents of a `G0` object with the `xyplot` method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[Sol](#), [Gef](#).

B4_Gef-class

Class "Gef": irradiation and irradiance on the generator plane.

Description

This class contains the global, diffuse and direct irradiation and irradiance on the horizontal plane, and ambient temperature.

Objects from the Class

Objects can be created by the function [calcGef](#).

Slots

GefI: Object of class `data.table` created by [fInclin](#). It contains these components:

Bo: Extra-atmospheric irradiance on the inclined surface (W/m²)

Bn: Direct normal irradiance (W/m²)

G, B, D, Di, Dc, R: Global, direct, diffuse (total, isotropic and anisotropic) and albedo irradiance incident on an inclined surface (W/m²)

Gef, Bef, Def, Dief, Dcef, Ref: Effective global, direct, diffuse (total, isotropic and anisotropic) and albedo irradiance incident on an inclined surface (W/m²)

FTb, FTd, FTr: Factor of angular losses for the direct, diffuse and albedo components

GefD: Object of class `data.table` with daily values of global, diffuse and direct irradiation.

Gefdm: Object of class `data.table` with monthly means of daily global, diffuse and direct irradiation.

Gefy: Object of class `data.table` with yearly sums of global, diffuse and direct irradiation.

Theta: Object of class `data.table` created by [fTheta](#). It contains these components:

Beta: numeric, inclination angle of the surface (radians). When `modeTrk='fixed'` it is the value of the argument beta converted from degrees to radians.

Alpha: numeric, azimuth angle of the surface (radians). When `modeTrk='fixed'` it is the value of the argument alpha converted from degrees to radians.

cosTheta: numeric, cosine of the incidence angle of the solar irradiance on the surface

iS: numeric, degree of dirtiness.

alb: numeric, albedo reflection coefficient.

modeTrk: character, mode of tracking.
modeShd: character, mode of shadows.
angGen: A list with the values of alpha, beta and betaLim.
struct: A list with the dimensions of the structure.
distances: A data.frame with the distances between structures.

Extends

Class "**G0**", directly. Class "**Meteo**", by class "G0", distance 2. Class "**Sol**", by class "G0", distance 2.

Methods

as.data.tableD signature(object = "Gef"): conversion to a data.table with daily values.
as.data.tableI signature(object = "Gef"): conversion to a data.table with intradaily values.
as.data.tableM signature(object = "Gef"): conversion to a data.table with monthly values.
as.data.tableY signature(object = "Gef"): conversion to a data.table with yearly values.
indexD signature(object = "Gef"): index of the solD slot.
indexI signature(object = "Gef"): index of the solI slot.
getLat signature(object = "Gef"): latitude of the inherited **Sol** object.
xyplot signature(x = "Gef", data = "missing"): display the time series of daily values of irradiation.
xyplot signature(x = "formula", data = "Gef"): displays the contents of a Gef object with the xyplot method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

Sol, **G0**.

B5_ProdGCPV-class *Class "ProdGCPV": performance of a grid connected PV system.*

Description

A class containing values of the performance of a grid connected PV system.

Objects from the Class

Objects can be created by [prodGCPV](#).

Slots

prodI: Object of class `data.table` created by [fProd](#). It includes these components:

Tc: cell temperature, °C.

Voc, Isc, Vmpp, Impp: open circuit voltage, short circuit current, MPP voltage and current, respectively.

Vdc, Idc: voltage and current at the input of the inverter.

Pdc: power at the input of the inverter, W

Pac: power at the output of the inverter, W

EffI: efficiency of the inverter

prodD: A `data.table` object with daily values of AC (`Eac`) and DC (`Edc`) energy (Wh), and productivity (`Yf`, Wh/Wp) of the system.

prodDm: A `data.table` object with monthly means of daily values of AC and DC energy (kWh), and productivity of the system.

prodY: A `data.table` object with yearly sums of AC and DC energy (kWh), and productivity of the system.

module: A list with the characteristics of the module.

generator: A list with the characteristics of the PV generator.

inverter: A list with the characteristics of the inverter.

effSys: A list with the efficiency values of the system.

Besides, this class contains the slots from the "[Meteo](#)", "[Sol](#)", "[G0](#)" and "[Gef](#)" classes.

Extends

Class "[Gef](#)", directly. Class "[G0](#)", by class "Gef", distance 2. Class "[Meteo](#)", by class "Gef", distance 3. Class "[Sol](#)", by class "Gef", distance 3.

Methods

as.data.tableD `signature(object = "ProdGCPV")`: conversion to a `data.table` with daily values.

as.data.tableI `signature(object = "ProdGCPV")`: conversion to a `data.table` with intradaily values.

as.data.tableM `signature(object = "ProdGCPV")`: conversion to a `data.table` with monthly values.

as.data.tableY `signature(object = "ProdGCPV")`: conversion to a `data.table` with yearly values.

indexD `signature(object = "ProdGCPV")`: index of the `solD` slot.

indexI signature(object = "ProdGCPV"): index of the **solI** object.
getLat signature(object = "ProdGCPV"): latitude of the inherited **Sol** object.
xyplot signature(x = "ProdGCPV", data = "missing"): display the time series of daily values.
xyplot signature(x = "formula", data = "ProdGCPV"): displays the contents of a ProdGCPV object with the xyplot method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[Sol](#), [G0](#), [Gef](#), [Shade](#).

B6_ProdPVPS-class	Class "ProdPVPS": performance of a PV pumping system.
-------------------	---

Description

Performance of a PV pumping system with a centrifugal pump and a variable frequency converter.

Objects from the Class

Objects can be created by [prodPVPS](#).

Slots

prodI: Object of class `data.table` with these components:
Q: Flow rate, (m³/h)
Pb, Ph: Pump shaft power and hydraulical power (W), respectively.
etam, etab: Motor and pump efficiency, respectively.
f: Frequency (Hz)
prodD: A `data.table` object with daily values of AC energy (Wh), flow (m³) and productivity of the system.
prodDm: A `data.table` object with monthly means of daily values of AC energy (kWh), flow (m³) and productivity of the system.
prodY: A `data.table` object with yearly sums of AC energy (kWh), flow (m³) and productivity of the system.
pump A list extracted from [pumpCoef](#)
H Total manometric head (m)
Pg Nominal power of the PV generator (Wp)
converter list containing the nominal power of the frequency converter, Pnom, and Ki, vector of three values, coefficients of the efficiency curve.
effSys list of numeric values with information about the system losses
Besides, this class contains the slots from the [Gef](#) class.

Extends

Class "**Gef**", directly. Class "**G0**", by class "Gef", distance 2. Class "**Meteo**", by class "Gef", distance 3. Class "**Sol**", by class "Gef", distance 3.

Methods

as.data.tableD signature(object = "ProdPVPS"): conversion to a data.table with daily values.
as.data.tableI signature(object = "ProdPVPS"): conversion to a data.table with intradaily values.
as.data.tableM signature(object = "ProdPVPS"): conversion to a data.table with monthly values.
as.data.tableY signature(object = "ProdPVPS"): conversion to a data.table with yearly values.
indexD signature(object = "ProdPVPS"): index of the **solD** slot.
indexI signature(object = "ProdPVPS"): index of the **solI** object.
getLat signature(object = "ProdPVPS"): latitude of the inherited **Sol** object.
xyplot signature(x = "ProdPVPS", data = "missing"): display the time series of daily values.
xyplot signature(x = "formula", data = "ProdPVPS"): displays the contents of a ProdPVPS object with the xyplot method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Abella, M. A., Lorenzo, E. y Chenlo, F.: PV water pumping systems based on standard frequency converters. Progress in Photovoltaics: Research and Applications, 11(3):179–191, 2003, ISSN 1099-159X.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[prodPVPS](#), [fPump](#).

Description

A class for the optimization of shadows in a PV system.

Objects from the Class

Objects can be created by [optimShd](#).

Slots

FS: numeric, shadows factor values for each combination of distances.

GRR: numeric, Ground Requirement Ratio for each combination.

Yf: numeric, final productivity for each combination.

FS.loess: A local fitting of FS with loess.

Yf.loess: A local fitting of Yf with loess.

modeShd: character, mode of shadows.

struct: A list with the dimensions of the structure.

distances: A data.frame with the distances between structures.

res numeric, difference (meters) between the different steps of the calculation.

Besides, as a reference, this class includes a [ProdGCPV](#) object with the performance of a PV systems without shadows.

Extends

Class "[ProdGCPV](#)", directly. Class "[Gef](#)", by class "ProdGCPV", distance 2. Class "[G0](#)", by class "ProdGCPV", distance 3. Class "[Meteo](#)", by class "ProdGCPV", distance 4. Class "[Sol](#)", by class "ProdGCPV", distance 4.

Methods

as.data.frame signature(x = "Shade"): conversion to a data.frame including columns for distances (Lew, Lns, and D) and results (FS, GRR and Yf).

shadeplot signature(x = "Shade"): display the results of the iteration with a level plot for the two-axis tracking, or with conventional plot for horizontal tracking and fixed systems.

xyplot signature(x = "formula", data = "Shade"): display the content of the Shade object with the xyplot method for formulas.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O.: Grandes Centrales Fotovoltaicas: producción, seguimiento y ciclo de vida. PhD Thesis, UNED, 2008. https://www.researchgate.net/publication/39419806_Grandes_Centrales_Fotovoltaicas_Produccion_Seguimiento_y_Ciclo_de_Vida.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[Gef](#), [ProdGCPV](#).

<code>C_corrFdKt</code>	<i>Correlations between the fraction of diffuse irradiation and the clearness index.</i>
-------------------------	--

Description

A set of correlations between the fraction of diffuse irradiation and the clearness index used by `fCompD` and `fCompI`.

Usage

```
## Monthly means of daily values
Ktm(sol, G0dm)
FdKtPage(sol, G0dm)
FdKtLJ(sol, G0dm)

## Daily values
Ktd(sol, G0d)
FdKtCPR(sol, G0d)
FdKtEKDd(sol, G0d)
FdKtCLIMEDd(sol, G0d)

## Intradaily values
Kti(sol, G0i)
FdKtEKDh(sol, G0i)
FdKtCLIMEDh(sol, G0i)
FdKtBRL(sol, G0i)
```

Arguments

<code>sol</code>	A <code>Sol</code> object, it may be the result of the <code>calcSol</code> function.
<code>G0dm</code>	A <code>Meteo</code> object with monthly means of radiation. It may be the result of the <code>readG0dm</code> function.
<code>G0d</code>	A <code>Meteo</code> object with daily values of radiation. It may be the result of the <code>readBDd</code> (or equivalent) function.
<code>G0i</code>	A <code>Meteo</code> object with intradaily values of radiation. It may be the result of the <code>readBDi</code> (or equivalent) function.

Value

A data.table, with two columns:

<code>Fd</code>	A numeric, the diffuse fraction.
<code>Kt</code>	A numeric, the clearness index(provided by the Kt functions).

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López; The BRL model was suggested by Kevin Ummel.

References

- Page, J. K., The calculation of monthly mean solar radiation for horizontal and inclined surfaces from sunshine records for latitudes 40N-40S. En U.N. Conference on New Sources of Energy, vol. 4, pág. 378–390, 1961.
- Collares-Pereira, M. y Rabl, A., The average distribution of solar radiation: correlations between diffuse and hemispherical and between daily and hourly insolation values. Solar Energy, 22:155–164, 1979.
- Erbs, D.G, Klein, S.A. and Duffie, J.A., Estimation of the diffuse radiation fraction for hourly, daily and monthly-average global radiation. Solar Energy, 28:293:302, 1982.
- De Miguel, A. et al., Diffuse solar irradiation model evaluation in the north mediterranean belt area, Solar Energy, 70:143-153, 2001.
- Ridley, B., Boland, J. and Lauret, P., Modelling of diffuse solar fraction with multiple predictors, Renewable Energy, 35:478-482, 2010.

See Also

[fCompD](#), [fCompI](#)

Examples

```

lat = 37.2
BTd = fBTd(mode = 'prom')
G0dm = c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742, 7.919, 7.027, 5.369,
       3.562, 2.814, 2.179)*1000;
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2,
      15.2)

prom = readG0dm(G0dm = G0dm, Ta = Ta, lat = lat)
sol = calcSol(lat = lat, BTd = BTd)

Kt = Ktm(sol = sol, G0dm = prom)
Kt

Page = FdKtPage(sol = sol, G0dm = prom)
LJ = FdKtLJ(sol = sol, G0dm = prom)
Monthly = merge(Page, LJ, by = 'Kt',
                suffixes = c('.Page', '.LJ'))
Monthly

xyplot(Fd.Page+Fd.LJ~Kt, data = Monthly,
       type = c('l', 'g'), auto.key = list(space = 'right'))

Kt = Ktd(sol = sol, G0d = prom)
Kt

CPR = FdKtCPR(sol = sol, G0d = prom)
CLIMEDd = FdKtCLIMEDd(sol = sol, G0d = prom)
Daily = merge(CPR, CLIMEDd, by = 'Kt',
              suffixes = c('.CPR', '.CLIMEDd'))
Daily

xyplot(Fd.CPR + Fd.CLIMEDd ~ Kt, data = Daily,
       type = c('l', 'g'), auto.key = list(space = 'right'))

```

C_fBTd*Daily time base*

Description

Construction of a daily time base for solar irradiation calculation

Usage

```
fBTd(mode = "prom",
      year = as.POSIXlt(Sys.Date())$year+1900,
      start = paste('01-01-', year, sep = ''),
      end = paste('31-12-', year, sep = ''),
      format = '%d-%m-%Y')
```

Arguments

mode	character, controls the type of time base to be created. With mode = 'serie' the result is a daily time series from start to end. With mode = 'prom' only twelve days, one for each month, are included. During these 'average days' the declination angle is equal to the monthly mean of this angle.
year	which year is to be used for the time base when mode = 'prom'. Its default value is the current year.
start	first day of the time base for mode = 'serie'. Its default value is the first of January of the current year.
end	last day of the time base for mode = 'serie'. Its default value is the last day of December of the current year.
format	format of start and end.

Details

This function is commonly used inside fSoID.

Value

This function returns a POSIXct object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fSoID](#), [as.POSIXct](#), [seq.POSIXt](#).

Examples

```
#Average days  
fBTd(mode = 'prom')  
  
#The day #100 of the year 2008  
BTd = fBTd(mode = 'serie', year = 2008)  
BTd[100]
```

*C_fBTi**Intra-daily time base*

Description

Construction of an intra-daily time base for solar irradiation calculation

Usage

```
fBTi(BTd, sample = 'hour')
```

Arguments

BTd	vector, it may be a result for fBTd or indexD
sample	character, identify the sample of the time set. Its default value is 'hour'.

Details

This function is commonly used inside fSolI.

Value

This function returns a POSIXct object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
#Average days  
BTd <- fBTd(mode = 'prom')  
  
#Intradaily base time for the first day  
BTi <- fBTi(BTd = BTd[1], sample = 'hour')  
BTi
```

Description

Extract the diffuse and direct components from the daily global irradiation on a horizontal surface by means of regressions between the clearness index and the diffuse fraction parameters.

Usage

```
fCompD(sol, G0d, corr = "CPR", f)
```

Arguments

<code>sol</code>	A <code>Sol</code> object from calcSol or a <code>data.table</code> object from fSolD . Both of them include a component named <code>B0d</code> , which stands for the extra-atmospheric daily irradiation incident on a horizontal surface
<code>G0d</code>	A <code>Meteo</code> object from readG0dm , readBDD , or a <code>data.table</code> object containing daily global irradiation (Wh/m^2) on a horizontal surface. See below for <code>corr = 'none'</code> .
<code>corr</code>	<p>A character, the correlation between the fraction of diffuse irradiation and the clearness index to be used.</p> <p>With this version several options are available, as described in corrFdKt. For example, the <code>FdKtPage</code> is selected with <code>corr = 'Page'</code> and the <code>FdKtCPR</code> with <code>corr = 'CPR'</code>.</p> <p>If <code>corr = 'user'</code> the use of a correlation defined by a function <code>f</code> is possible.</p> <p>If <code>corr = 'none'</code> the <code>G0d</code> object should include information about global, diffuse and direct daily irradiation with columns named <code>G0d</code>, <code>D0d</code> and <code>B0d</code>, respectively.</p>
<code>f</code>	A function defininig a correlation between the fraction of diffuse irradiation and the clearness index. It is only neccessary when <code>corr = 'user'</code>

Value

A `data.table` object which includes:

<code>Fd</code>	numeric, the diffuse fraction
<code>Ktd</code>	numeric, the clearness index
<code>G0d</code>	numeric, the global irradiation on a horizontal surface (Wh/m^2)
<code>D0d</code>	numeric, the diffuse irradiation on a horizontal surface (Wh/m^2)
<code>B0d</code>	numeric, the direct irradiation on a horizontal surface (Wh/m^2)

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also[fCompI](#)**Examples**

```

lat = 37.2;
BTd = fBTd(mode = 'serie')

SolD <- fSolD(lat, BTd[100])

G0d = 5000
fCompD(SolD, G0d, corr = "Page")
fCompD(SolD, G0d, corr = "CPR")

#define a function fKtd with the correlation of CPR
fKtd = function(sol, G0d){
  Kt = Ktm(sol, G0d)
  Fd = (0.99*(Kt <= 0.17)) + (Kt>0.17)*(1.188 -2.272 * Kt + 9.473 * Kt^2 -
  21.856 * Kt^3 + 14.648 * Kt^4)
  return(data.table(Fd, Kt))
}
#The same as with corr = "CPR"
fCompD(SolD, G0d, corr = "user", f = fKtd)

lat = -37.2;
SolDs <- fSolD(lat, BTd[283])
G0d = data.table(Dates = SolDs$Dates, G0d = 5000)
fCompD(SolDs, G0d, corr = "CPR")

lat = 37.2;
G0dm = c(2.766,3.491,4.494,5.912,6.989,7.742,7.919,7.027,5.369,3.562,2.814,2.179)*1000;
Rad = readG0dm(G0dm, lat = lat)
solD <- fSolD(lat, fBTd(mode = 'prom'))
fCompD(solD, Rad, corr = 'Page')

```

C_fCompI*Calculation of solar irradiance on a horizontal surface*

Description

From the daily global, diffuse and direct irradiation values supplied by `fCompD`, the profile of the global, diffuse and direct irradiance is calculated with the `rd` and `rg` components of `fSolI`.

Usage

```
fCompI(sol, compD, G0I, corr = 'none', f, filterG0 = TRUE)
```

Arguments

<code>sol</code>	A <code>Sol</code> object as provided by calcSol or a <code>data.table</code> object as provided by fSolI .
<code>compD</code>	A <code>data.table</code> object as provided by <code>fCompD</code> . It is not considered if <code>G0I</code> is provided.

G0I	A Meteo object from <code>readBDI</code> , <code>dt2Meteo</code> or <code>zoo2Meteo</code> , or a <code>data.table</code> object containing <i>intradaily</i> global irradiance (W/m^2) on a horizontal surface. See below for <code>corr = 'none'</code> .
corr	A character, the correlation between the fraction of intradaily diffuse irradiation and the clearness index to be used. It is ignored if G0I is not provided. With this version several correlations are available, as described in <code>corrFdKt</code> . You should choose one of <i>intradaily</i> proposals. For example, the <code>FdKtCLIMEDh</code> is selected with <code>corr = 'CLIMEDh'</code> . If <code>corr = 'user'</code> the use of a correlation defined by a function f is possible. If <code>corr = 'none'</code> the G0I object must include information about global, diffuse and direct intradaily irradiation with columns named G0, D0 and B0, respectively.
f	A function defining a correlation between the fraction of diffuse irradiation and the clearness index. It is only necessary when <code>corr = 'user'</code>
filterG0	A logical. If TRUE (default) this function sets the global irradiation values to NA when they are higher than the extra-atmospheric irradiation values.

Value

A `data.table` with these components:

kt	numeric, clearness index.
fd	numeric, diffuse fraction.
G0	numeric, global irradiance on a horizontal surface, (W/m^2)
D0	numeric, diffuse irradiance on a horizontal surface, (W/m^2)
B0	numeric, direct irradiance on a horizontal surface, (W/m^2)

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Collares-Pereira, M. y Rabl, A., The average distribution of solar radiation: correlations between diffuse and hemispherical and between daily and hourly insolation values. *Solar Energy*, 22:155–164, 1979.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:10.18637/jss.v050.i09

See Also

`fCompD`, `fSolI`, `calcSol`, `corrFdKt`.

Examples

```
lat <- 37.2

BTd <- fBTd(mode = 'serie')
solD <- fSolD(lat, BTd[100])
solI <- fSolI(solD, sample = 'hour')
G0d <- data.table(Dates = solD$Dates, G0d = 5000)
```

```

compD <- fCompD(solD, G0d, corr = "Page")
fCompI(solI, compD)

sol <- calcSol(lat, fBTd(mode = 'prom'), sample = 'hour', keep.night = FALSE)

G0dm <- c(2.766, 3.491, 4.494, 5.912, 6.989, 7.742,
         7.919, 7.027, 5.369, 3.562, 2.814, 2.179)*1000

Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9,
       24.3, 18.2, 17.2, 15.2)

BD <- readG0dm(G0dm = G0dm, Ta = Ta, lat = lat)
compD <- fCompD(sol, BD, corr = 'Page')
compI <- fCompI(sol, compD)
head(compI)

## Use of 'corr'. The help page of calcG0 includes additional examples
## with intradaily data xyplot(fd ~ kt, data = compI)

climed <- fCompI(sol, G0I = compI, corr = 'CLIMEDh')
xyplot(Fd ~ Kt, data = climed)

ekdh <- fCompI(sol, G0I = compI, corr = 'EKDh')
xyplot(Fd ~ Kt, data = ekdh)

brl <- fCompI(sol, G0I = compI, corr = 'BRL')
xyplot(Fd ~ Kt, data = brl)

```

C_fInclin*Solar irradiance on an inclined surface***Description**

The solar irradiance incident on an inclined surface is calculated from the direct and diffuse irradiance on a horizontal surface, and from the evolution of the angles of the Sun and the surface. Moreover, the effect of the angle of incidence and dust on the PV module is included to obtain the effective irradiance.

This function is used by the [calcGef](#) function.

Usage

```
fInclin(compI, angGen, iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE)
```

Arguments

<code>compI</code>	A <code>G0</code> object. It may be the result of calcG0 .
<code>angGen</code>	A <code>data.table</code> object, including at least three variables named <code>Beta</code> , <code>Alpha</code> and <code>cosTheta</code> . It may be the result of fTheta .
<code>iS</code>	integer, degree of dirtiness. Its value must be included in the set (1,2,3,4). <code>iS</code> = 1 corresponds to a clean surface while <code>iS</code> = 4 is the choice for a dirty surface. Its default value is 2
<code>alb</code>	numeric, albedo reflection coefficient. Its default value is 0.2

horizBright	logical, if TRUE, the horizon brightness correction proposed by Reind et al. is used.
HCPV	logical, if TRUE the diffuse and albedo components of the <i>effective</i> irradiance are set to zero. HCPV is the acronym of High Concentration PV system.

Details

The solar irradiance incident on an inclined surface can be calculated from the direct and diffuse irradiance on a horizontal surface, and from the evolution of the angles of the Sun and the surface. The transformation of the direct radiation is straightforward since only geometric considerations are needed. However, the treatment of the diffuse irradiance is more complex since it involves the modelling of the atmosphere. There are several models for the estimation of diffuse irradiance on an inclined surface. The one which combines simplicity and acceptable results is the proposal of Hay and McKay. This model divides the diffuse component in isotropic and anisotropic whose values depends on a anisotropy index. On the other hand, the effective irradiance, the fraction of the incident irradiance that reaches the cells inside a PV module, is calculated with the losses due to the angle of incidence and dirtiness. This behaviour can be simulated with a model proposed by Martin and Ruiz requiring information about the angles of the surface and the level of dirtiness (iS)

Value

A `data.table` object with these components:

Bo	Extra-atmospheric irradiance on the inclined surface (W/m ²)
Bn	Direct normal irradiance (W/m ²)
G, B, D, Di, Dc, R	Global, direct, diffuse (total, isotropic and anisotropic) and albedo irradiance incident on an inclined surface (W/m ²)
Gef, Bef, Def, Dief, Dcef, Ref	Effective global, direct, diffuse (total, isotropic and anisotropic) and albedo irradiance incident on an inclined surface (W/m ²)
FTb, FTd, FTr	Factor of angular losses for the direct, diffuse and albedo components

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Hay, J. E. and McKay, D. C.: Estimating Solar Irradiance on Inclined Surfaces: A Review and Assessment of Methodologies. *Int. J. Solar Energy*, (3):pp. 203, 1985.
- Martin, N. and Ruiz, J.M.: Calculation of the PV modules angular losses under field conditions by means of an analytical model. *Solar Energy Materials & Solar Cells*, 70:25–38, 2001.
- D. T. Reindl and W. A. Beckman and J. A. Duffie: Evaluation of hourly tilted surface radiation models, *Solar Energy*, 45:9-17, 1990.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

`fTheta`, `fCompI`, `calcGef`.

<i>C_fProd</i>	<i>Performance of a PV system</i>
----------------	-----------------------------------

Description

Simulate the behaviour of a grid connected PV system under different conditions of irradiance and temperature. This function is used by the [prodGCPV](#) function.

Usage

```
fProd(inclin, module, generator, inverter, effSys)
```

Arguments

<i>inclin</i>	A Gef object, a <code>data.table</code> object. In case of being <code>data.table</code> it must include a component named <i>Gef</i> (effective irradiance, W/m^2) and another named <i>Ta</i> (ambient temperature, $^\circ\text{C}$).
<i>module</i>	<p><i>Vocn</i> open-circuit voltage of the module at Standard Test Conditions (default value 51.91 volts.)</p> <p><i>Iscn</i> short circuit current of the module at Standard Test Conditions (default value 14.07 amperes.)</p> <p><i>Vmn</i> maximum power point voltage of the module at Standard Test Conditions (default value 43.76 volts.)</p> <p><i>Imn</i> Maximum power current of the module at Standard Test Conditions (default value 13.03 amperes.)</p> <p><i>Ncs</i> number of cells in series inside the module (default value 24)</p> <p><i>Ncp</i> number of cells in parallel inside the module (default value 6)</p> <p><i>CoefVT</i> coefficient of decrement of voltage of each cell with the temperature (default value 0.0049 volts per celsius degree)</p> <p><i>TONC</i> nominal operational cell temperature, celsius degree (default value 45).</p>
<i>generator</i>	<p><i>Nms</i> number of modules in series (default value 22)</p> <p><i>Nmp</i> number of modules in parallel (default value 130)</p>
<i>inverter</i>	<p><i>Ki</i> vector of three values, coefficients of the efficiency curve of the inverter (default $c(0.002, 0.005, 0.008)$), or a matrix of nine values (3×3) if there is dependence with the voltage (see references).</p> <p><i>Pinv</i> nominal inverter power (W) (default value $1.5\text{e}6$ watts.)</p> <p><i>Vmin</i>, <i>Vmax</i> minimum and maximum voltages of the MPP range of the inverter (default values 822 and 1300 volts)</p> <p><i>Gumb</i> minimum irradiance for the inverter to start (W/m^2) (default value 20 W/m^2)</p>
<i>effSys</i>	<p><i>list</i> of numeric values with information about the system losses,</p> <p><i>ModQual</i> average tolerance of the set of modules (%), default value is 3</p> <p><i>ModDisp</i> module parameter dispersion losses (%), default value is 2</p>

OhmDC Joule losses due to the DC wiring (%), default value is 1.5
OhmAC Joule losses due to the AC wiring (%), default value is 1.5
MPP average error of the MPP algorithm of the inverter (%), default value is 1
TrafoMT losses due to the MT transformer (%), default value is 1
Disp losses due to stops of the system (%), default value is 0.5

Value

If `inclin` is `data.table` or `Gef` object, the result is a `data.table` object with these components:

<code>Tc</code>	cell temperature, °C.
<code>Voc, Isc, Vmpp, Impp</code>	open circuit voltage, short circuit current, MPP voltage and current, respectively, in the conditions of irradiance and temperature provided by <code>Inclin</code>
<code>Vdc, Idc</code>	voltage and current at the input of the inverter. If no voltage limitation occurs (according to the values of <code>inverter\$Vmax</code> and <code>inverter\$Vmin</code>), their values are identical to <code>Vmpp</code> and <code>Impp</code> . If the limit values are reached a warning is produced
<code>Pdc</code>	power at the input of the inverter, W
<code>Pac</code>	power at the output of the inverter, W
<code>EffI</code>	efficiency of the inverter

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Jantsch, M., Schmidt, H. y Schmid, J.: Results on the concerted action on power conditioning and control. 11th European photovoltaic Solar Energy Conference, 1992.
- Baumgartner, F. P., Schmidt, H., Burger, B., Bründlinger, R., Haeberlin, H. and Zehner, M.: Status and Relevance of the DC Voltage Dependency of the Inverter Efficiency. 22nd European Photovoltaic Solar Energy Conference, 2007.
- Alonso Garcia, M. C.: Caracterización y modelado de asociaciones de dispositivos fotovoltaicos. PhD Thesis, CIEMAT, 2005.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

`fInclin`, `prodGCPV`, `fTemp`.

Examples

```
inclin = data.table(Gef = c(200, 400, 600, 800, 1000), Ta = 25)

#using default values
fProd(inclin)

#Using a matrix for Ki (voltage dependence)
```

```
inv1 <- list(Ki = rbind(c(-0.00019917, 7.513e-06, -5.4183e-09),
c(0.00806, -4.161e-06, 2.859e-08),
c(0.02118, 3.4002e-05, -4.8967e-08)))

fProd(inclin, inverter = inv1)

#Voltage limits of the inverter
inclin = data.table(Gef = 800,Ta = 30)
gen1 = list(Nms = 10, Nmp = 11)

prod = fProd(inclin,generator = gen1)
print(prod)

with(prod, Vdc * Idc / (Vmpp * Impp))
```

C_fPump*Performance of a centrifugal pump*

Description

Compute the performance of the different parts of a centrifugal pump fed by a frequency converter following the affinity laws.

Usage

```
fPump(pump, H)
```

Arguments

pump	list containing the parameters of the pump to be simulated. It may be a row of pumpCoef .
H	Total manometric head (m).

Value

lim	Range of values of electrical power input
fQ	Function constructed with <code>splinefun</code> relating flow and electrical power
fPb	Function constructed with <code>splinefun</code> relating pump shaft power and electrical power of the motor
fPh	Function constructed with <code>splinefun</code> relating hydraulical power and electrical power of the motor
fFreq	Function constructed with <code>splinefun</code> relating frequency and electrical power of the motor

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Abella, M. A., Lorenzo, E. y Chenlo, F.: PV water pumping systems based on standard frequency converters. Progress in Photovoltaics: Research and Applications, 11(3):179–191, 2003, ISSN 1099-159X.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[NmgPVPS](#), [prodPVPS](#), [pumpCoef](#), [splinefun](#).

Examples

```
library(latticeExtra)

data(pumpCoef)
CoefSP8A44 <- subset(pumpCoef, Qn == 8 & stages == 44)

fSP8A44 <- fPump(pump = CoefSP8A44, H = 40)
SP8A44 = with(fSP8A44,{
  Pac = seq(lim[1],lim[2],by = 100)
  Pb = fPb(Pac)
  etam = Pb/Pac
  Ph = fPh(Pac)
  etab = Ph/Pb
  f = fFreq(Pac)
  Q = fQ(Pac)
  result = data.frame(Q,Pac,Pb,Ph,etam,etab,f)})}

#Efficiency of the motor, pump and the motor-pump
SP8A44$etamb = with(SP8A44, etab*etam)
lab = c(expression(eta[motor]), expression(eta[pump]), expression(eta[mp]))
p <- xyplot(etam + etab + etamb ~ Pac,data = SP8A44,type = 'l', ylab = 'Efficiency')
p+layer(panel.text(x[1], y[1], lab[group.number], pos = 3))

#Mechanical, hydraulic and electrical power
lab = c(expression(P[pump]), expression(P[hyd]))
p <- xyplot(Pb + Ph ~ Pac,data = SP8A44,type = 'l', ylab = 'Power (W)', xlab = 'AC Power (W)')
p+layer(panel.text(x[length(x)], y[length(x)], lab[group.number], pos = 3))

#Flow and electrical power
xyplot(Q ~ Pac,data = SP8A44,type = 'l')
```

Description

Compute the daily apparent movement of the Sun from the Earth. This movement is mainly described (for the simulation of photovoltaic systems) by the declination angle, the sunrise angle and the daily extra-atmospheric irradiation.

Usage

```
fSolD(lat, BTd, method = 'michalsky')
```

Arguments

lat	Latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
BTd	Daily temporal base, a <code>POSIXct</code> object which may be the result of <code>fBTd</code> .
method	character, method for the sun geometry calculations to be chosen from 'cooper', 'spencer', 'michalsky' and 'strous'. See references for details.

Value

A `data.table` object with these components:

lat	Latitude (degrees)
decl	Declination angle (radians) for each day of year in dn or BTd
eo	Factor of correction due the eccentricity of orbit of the Earth around the Sun.
ws	Sunrise angle (in radians) for each day of year. Due to the convention which considers that the solar hour angle is negative before midday, this angle is negative.
B0d	Extra-atmospheric daily irradiation (watt-hour per squared meter) incident on a horizontal surface
EoT	Equation of Time.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Cooper, P.I., Solar Energy, 12, 3 (1969). "The Absorption of Solar Radiation in Solar Stills"
- Spencer, Search 2 (5), 172, <https://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html>
- Strous: <https://www.aa.quae.nl/en/reken/zonpositie.html>
- Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

Examples

```
BTd <- fBTd(mode = 'serie')

lat <- 37.2
fSolD(lat, BTd[100])
fSolD(lat, BTd[100], method = 'strous')
fSolD(lat, BTd[100], method = 'spencer')
fSolD(lat, BTd[100], method = 'cooper')
```

```

lat <- -37.2
fSolD(lat, BTd[283])

#Solar angles along the year
SolD <- fSolD(lat, BTd = fBTd())

library(lattice)
xyplot(SolD)

#Calculation of the daylength for several latitudes
library(latticeExtra)

Lats <- c(-60, -40, -20, 0, 20, 40, 60)
NomLats <- ifelse(Lats > 0, paste(Lats,'N', sep = ''),
                   paste(abs(Lats), 'S', sep = ''))
NomLats[Lats == 0] <- '0'

BTd <- fBTd(mode = 'serie')
mat <- matrix(nrow = length(BTd), ncol = length(Lats))
colnames(mat) <- NomLats
WsZ <- data.table(Dates = BTd, mat)

for (i in seq_along(Lats)){
  SolDaux <- fSolD(lat = Lats[i], BTd = fBTd(mode = 'serie'));
  WsZ[,i+1] <- r2h(2*abs(SolDaux$ws))}

p = xyplot(`^60S` + `^40S` + `^20S` + `^0` + `^20N` + `^40N` + `^60N` ~ Dates, data = WsZ, type = "l",
            ylab = expression(omega[s] * (h)))
plab = p+glayer(panel.text(x[1], y[1], NomLats[group.number], pos = 2))
print(plab)

```

C_fSolI*Instantaneous apparent movement of the Sun from the Earth*

Description

Compute the angles which describe the intradaily apparent movement of the Sun from the Earth.

Usage

```
fSolI(solD, sample = 'hour', BTi, EoT = TRUE, keep.night = TRUE, method = 'michalsky')
```

Arguments

solD	A <code>data.table</code> object with the result of <code>fSolD</code>
sample	Increment of the intradaily sequence. It is a character string, containing one of "sec", "min", "hour". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s". It is used by <code>seq.POSIXt</code> . It is not considered when <code>BTi</code> is provided.
BTi	Intradaily time base, a <code>POSIXct</code> object. It could be the index of the <code>G0I</code> argument to <code>calcG0</code> . <code>fSolI</code> will produce results only for those days contained both in <code>solD</code> and in <code>BTi</code> .

C_fSolI

51

EoT	logical, if TRUE (default) the Equation of Time is used.
keep.night	logical, if TRUE (default) the night is included in the time series.
method	character, method for the sun geometry calculations to be chosen from 'cooper', 'spencer', 'michalsky' and 'strous'. See references for details.

Value

A `data.table` object is returned with these components:

lat	numeric, latitude (degrees)
w	numeric, solar hour angle (radians)
aman	logical, TRUE when Sun is above the horizon
cosThzS	numeric, cosine of the solar zenith angle
AzS	numeric, solar azimuth angle (radians)
A1S	numeric, solar elevation angle (radians)
Boθ	numeric, extra-atmospheric irradiance (W/m ²)

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Cooper, P.I., Solar Energy, 12, 3 (1969). "The Absorption of Solar Radiation in Solar Stills"
- Spencer, Search 2 (5), 172, <https://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html>
- Strous: <https://www.aa.quae.nl/en/reken/zonpositie.html>
- Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235
- Collares-Pereira, M. y Rabl, A., The average distribution of solar radiation: correlations between diffuse and hemispherical and between daily and hourly insolation values. Solar Energy, 22:155–164, 1979.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fSolD](#)

Examples

```
###Angles for one day
BTd = fBTd(mode = 'serie')

#North hemisphere
lat = 37.2
solD <- fSolD(lat,BTd[100])
solI <- fSolI(solD, sample = 'hour')
print(solI)
```

```

#South hemisphere
lat = -37.2;
solDs <- fSolD(lat,BTd[283])
solIs <- fSolI(solDs, sample = 'hour')
print(solIs)

###Angles for the 12 average days
lat = 37.2;
solD <- fSolD(lat,BTd = fBTd(mode = 'prom'))
solI <- fSolI(solD, sample = '10 min', keep.night = FALSE)

library(lattice)
library(latticeExtra)

###Solar elevation angle vs. azimuth.
#This kind of graphics is useful for shadows calculations
mon = month.abb
p <- xyplot(r2d(ALS)~r2d(AzS),
             groups = month(Dates),
             data = solI, type = 'l', col = 'black',
             xlab = expression(psi[s]), ylab = expression(gamma[s]))

plab <- p + glayer({
  idx <- round(length(x)/2+1)
  panel.text(x[idx], y[idx], mon[group.value], pos = 3, offset = 0.2, cex = 0.8)})}

print(plab)

```

Description

Compute the shadows factor for two-axis and horizontal N-S axis trackers and fixed surfaces.

Usage

```

fSombra(angGen, distances, struct, modeTrk = 'fixed', prom = TRUE)

fSombra6(angGen, distances, struct, prom = TRUE)

fSombra2X(angGen, distances, struct)

fSombraHoriz(angGen, distances, struct)

fSombraEst(angGen, distances, struct)

```

Arguments

angGen	A <code>data.table</code> object, including at least variables named Beta, Alpha, AzS, ALS and cosTheta.
--------	--

distances	data.frame, with a component named Lew, being the distance (meters) between horizontal NS and two-axis trackers along the East-West direction, a component named Lns for two-axis trackers or a component named D for static surfaces. An additional component named H can be included with the relative height (meters) between surfaces. When modeTrk = 'two' (or when fSombra6 is used) this data.frame may have five rows. Each of these rows defines the distances of a tracker in a set of six ones.
struct	list. When modeTrk = 'fixed' or modeTrk = 'horiz' only a component named L, which is the height (meters) of the tracker, is needed. For two-axis trackers (modeTrk = 'two'), an additional component named W, the width of the tracker, is required. Moreover, two components named Nrow and Ncol are included under this list. These components define, respectively, the number of rows and columns of the whole set of trackers in the PV plant.
modeTrk	character, to be chosen from 'fixed', 'two' or 'horiz'. When modeTrk = 'fixed' the surface is fixed (inclination and azimuth angles are constant). The performance of a two-axis tracker is calculated with modeTrk = 'two', and modeTrk = 'horiz' is the option for an horizontal N-S tracker. Its default value is modeTrk = 'fixed'
prom	logical, only needed for two-axis tracker mode. If TRUE the shadows are averaged between the set of trackers defined by struct\$Nrow and struct\$Ncol

Details

fSombra is only a wrapper for **fSombra6** (two-axis trackers), **fSombraEst** (fixed systems) and **fSombraHoriz** (horizontal N-S axis trackers). Depending on the value of modeTrk the corresponding function is selected. **fSombra6** calculates the shadows factor in a set of six two-axis trackers. If distances has only one row, this function constructs a symmetric grid around a tracker located at (0,0,0). These five trackers are located at (-Lew, Lns, H), (0, Lns, H), (Lew, Lns, H), (-Lew, 0, H) and (Lns, 0, H). It is possible to define an irregular grid around (0,0,0) including five rows in distances. When prom = TRUE the shadows factor for each of the six trackers is calculated. Then, according to the distribution of trackers in the plant defined by struct\$Nrow and struct\$Ncol, a weighted average of the shadows factors is the result. It is important to note that the distances are defined between axis for trackers and between similar points of the structure for fixed surfaces.

Value

data.table including angGen and a variable named FS, which is the shadows factor. This factor is the ratio between the area of the generator affected by shadows and the total area. Therefore its value is 1 when the PV generator is completely shadowed.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O.: Grandes Centrales Fotovoltaicas: producción, seguimiento y ciclo de vida. PhD Thesis, UNED, 2008. https://www.researchgate.net/publication/39419806_Grandes_Centrales_Fotovoltaicas_Produccion_Seguimiento_y_Ciclo_de_Vida.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[calcShd](#), [optimShd](#), [fTheta](#), [calcSol](#)

Examples

```
lat = 37.2;
sol <- calcSol(lat, fBTd(mode = 'prom'), sample = '10 min', keep.night = FALSE)
angGen <- fTheta(sol, beta = 35);
Angles <- merge(as.data.tableI(sol), angGen)

###Two-axis tracker
#Symmetric grid
distances = data.table(Lew = 40,Lns = 30,H = 0)
struct = list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 8)

ShdFactor <- fSombra6(Angles, distances, struct, prom = FALSE)

Angles$FS = ShdFactor
xyplot(FS ~ w, groups = month(Dates), data = Angles,
       type = 'l',
       auto.key = list(space = 'right',
                       lines = TRUE,
                       points = FALSE))

#Symmetric grid defined with a five rows data.frame
distances = data.table(Lew = c(-40,0,40,-40,40),
                       Lns = c(30,30,30,0,0),
                       H = 0)
ShdFactor2 <- fSombra6(Angles, distances, struct,prom = FALSE)

#of course, with the same result
identical(ShdFactor, ShdFactor2)
```

Description

From the maximum and minimum daily values of ambient temperature, its evolution its calculated through a combination of cosine functions (ESRA method)

Usage

`fTemp(sol, BD)`

Arguments

- | | |
|------------------|---|
| <code>sol</code> | A Sol object. It may be the result of the calcSol function. |
| <code>BD</code> | A Meteo object, as provided by the readBDd function. It must include information about TempMax and TempMin. |

Details

The ESRA method estimates the dependence of the temperature on the time of the day (given as the local solar time) from only two inputs: minimum and maximum daily temperatures. It assumes that the temperature daily profile can be described using three piecewise cosine functions, dividing the day into three periods: from midnight to sunrise, from sunrise to the time of peak temperature (3 hours after midday), and to midnight.

Value

A `data.table` object with the profile of the ambient temperature.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Huld, T. , Suri, M., Dunlop, E. D., and Micale F., Estimating average daytime and daily temperature profiles within Europe, *Environmental Modelling & Software* 21 (2006) 1650-1661.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[calcSol](#), [readBDD](#).

Description

The orientation, azimuth and incidence angle are calculated from the results of `fSolI` or `calcSol` and from the information supplied by the arguments `beta` and `alpha` when the surface is fixed (`modeTrk = 'fixed'`) or the movement equations when a tracking surface is chosen (`modeTrk = 'horiz'` or `modeTrk = 'two'`). Besides, the modified movement of a horizontal NS tracker due to the back-tracking strategy is calculated if `BT = TRUE` with information about the tracker and the distance between the trackers included in the system.

This function is used by the `calcGef` function.

Usage

```
fTheta(sol, beta, alpha = 0, modeTrk = "fixed", betaLim = 90,
      BT = FALSE, struct, dist)
```

Arguments

<code>sol</code>	Sol object as provided by <code>calcSol</code> .
<code>beta</code>	numeric, inclination angle of the surface (degrees). It is only needed when <code>modeTrk = 'fixed'</code> .
<code>alpha</code>	numeric, azimuth angle of the surface (degrees). It is measured from the south (<code>alpha = 0</code>), and it is negative to the east and positive to the west. It is only needed when <code>modeTrk = 'fixed'</code> . Its default value is <code>alpha = 0</code> (surface facing to the south).
<code>modeTrk</code>	character, to be chosen from <code>'fixed'</code> , <code>'two'</code> or <code>'horiz'</code> . When <code>modeTrk = 'fixed'</code> the surface is fixed (inclination and azimuth angles are constant). The performance of a two-axis tracker is calculated with <code>modeTrk = 'two'</code> , and <code>modeTrk = 'horiz'</code> is the option for an horizontal N-S tracker. Its default value is <code>modeTrk = 'fixed'</code>
<code>betaLim</code>	numeric, maximum value of the inclination angle for a tracking surface. Its default value is 90 (no limitation))
<code>BT</code>	logical, TRUE when the backtracking technique is to be used with a horizontal NS tracker, as described by Panico et al. (see References). The default value is FALSE. In future versions of this package this technique will be available for two-axis trackers.
<code>struct</code>	Only needed when <code>BT = TRUE</code> . A list, with a component named <code>L</code> , which is the height (meters) of the tracker. In future versions the backtracking technique will be used in conjunction with two-axis trackers, and a additional component named <code>W</code> will be needed.
<code>dist</code>	Only needed when <code>BT = TRUE</code> . A <code>data.frame</code> , with a component named <code>Lew</code> , being the distance between the horizontal NS trackers along the East-West direction. In future versions an additional component named <code>Lns</code> will be needed for two-axis trackers with backtracking.

Value

A `data.table` object with these components:

<code>Beta</code>	numeric, inclination angle of the surface (radians). When <code>modeTrk = 'fixed'</code> it is the value of the argument <code>beta</code> converted from degrees to radians.
<code>Alpha</code>	numeric, azimuth angle of the surface (radians). When <code>modeTrk = 'fixed'</code> it is the value of the argument <code>alpha</code> converted from degrees to radians.
<code>cosTheta</code>	numeric, cosine of the incidence angle of the solar irradiance on the surface

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Panico, D., Garvison, P., Wenger, H. J., Shugar, D., Backtracking: a novel strategy for tracking PV systems, Photovoltaic Specialists Conference, 668-673, 1991
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fInclin](#), [fSombra](#), [calcGef](#).

C_HQCurve

H-Q curves of a centrifugal pump

Description

Compute and display the H-Q curves of a centrifugal pump fed working at several frequencies, and the iso-efficiency curve as a reference.

Usage

`HQCurve(pump)`

Arguments

<code>pump</code>	list containing the parameters of the pump to be simulated. It may be a row of pumpCoef .
-------------------	---

Value

<code>result</code>	A <code>data.frame</code> with the result of the simulation. It contains several columns with values of manometric height (<code>H</code>), frequency (<code>fe</code> and <code>fb</code>), mechanical power (<code>Pb</code>), AC electrical power (<code>Pm</code>), DC electrical power (<code>Pdc</code>) and efficiency of the pump (<code>etab</code>) and motor (<code>etam</code>).
---------------------	--

<code>plot</code>	The plot with several curves labelled with the correspondent frequencies, and the isoefficiency curve (named "ISO").
-------------------	--

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Abella, M. A., Lorenzo, E. y Chenlo, F.: PV water pumping systems based on standard frequency converters. *Progress in Photovoltaics: Research and Applications*, 11(3):179–191, 2003, ISSN 1099-159X.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[NmgPVPS](#), [prodPVPS](#), [pumpCoef](#).

Examples

```
library(lattice)
library(latticeExtra)

data(pumpCoef)

CoefSP8A44 <- subset(pumpCoef, Qn == 8&stages == 44)
CurvaSP8A44 <- HQCurve(pump = CoefSP8A44)
```

C_local2Solar

Local time, mean solar time and UTC time zone.

Description

The function `local2Solar` converts the time zone of a `POSIXct` object to the mean solar time and set its time zone to UTC as a synonym of mean solar time. It includes two corrections: the difference of longitudes between the location and the time zone, and the daylight saving time.

The function `lonHH` calculates the longitude (radians) of a time zone.

Usage

```
local2Solar(x, lon = NULL)
lonHH(tz)
```

Arguments

<code>x</code>	a <code>POSIXct</code> object
<code>lon</code>	A numeric value of the longitude (degrees) of the location. If <code>lon = NULL</code> (default), this value is assumed to be equal to the longitude of the time zone of <code>x</code> , so only the daylight saving time correction (if needed) is included.
<code>tz</code>	A character, a time zone as documented in https://en.wikipedia.org/wiki/List_of_tz_database_time_zones .

Details

Since the result of `local2Solar` is the mean solar time, the Equation of Time correction is not calculated with this function. The `eot` function includes this correction if desired.

Value

The function `local2Solar` produces a `POSIXct` object with its time zone set to UTC.

The function `lonHH` gives a numeric value.

Note

It is important to note that the `solaR2` package sets the system time zone to UTC with `Sys.setenv(TZ = 'UTC')`.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

Examples

```
t.local <- as.POSIXct("2006-01-08 10:07:52", tz = 'Europe/Madrid')

##The local time zone and the location have the same longitude (15 degrees)
local2Solar(t.local)
##But Madrid is at lon = -3
local2Solar(t.local, lon = -3)

##Daylight saving time
t.local.dst <- as.POSIXct("2006-07-08 10:07:52", tz = 'Europe/Madrid')

local2Solar(t.local.dst)
local2Solar(t.local.dst, lon = -3)
```

*C_NmgPVPS**Nomogram of a photovoltaic pumping system*

Description

This function simulate the performance of a water pump fed by a frequency converter with several PV generators of different size during a day. The result is plotted as a nomogram which relates the nominal power of the PV generator, the total water flow and the total manometric head.

Usage

```
NmgPVPS(pump, Pg, H, Gd, Ta = 30,
         lambda = 0.0045, TONC = 47, eta = 0.95,
         Gmax = 1200, t0 = 6, Nm = 6,
         title = '', theme = custom.theme.2())
```

Arguments

pump	A list extracted from pumpCoef
Pg	Sequence of values of the nominal power of the PV generator (Wp))
H	Sequence of values of the total manometric head (m)
Gd	Global irradiation incident on the generator (Wh/m ²)
Ta	Ambient temperature (°C).
lambda	Power losses factor due to temperature
TONC	Nominal operational cell temperature (°C).
eta	Average efficiency of the frequency converter
Gmax	Maximum value of irradiance (parameter of the IEC 61725)
t0	Hours from midday to sunset (parameter of the IEC 61725)

Nm	Number of samples per hour
title	Main title of the plot.
theme	Theme of the lattice plot.

Details

This function computes the irradiance profile according to the IEC 61725 "Analytical Expression for Daily Solar Profiles", which is a common reference in the official documents regarding PV pumping systems. At this version only pumps from the manufacturer Grundfos are included in [pumpCoef](#).

Value

I	list with the results of irradiance, power and flow of the system.
D	list with the results of total irradiation, electrical energy and flow for every nominal power of the generator.
param	list with the arguments used in the call to the function.
plot	trellis object containing the nomogram.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Abella, M. A., Lorenzo, E. y Chenlo, F.: PV water pumping systems based on standard frequency converters. *Progress in Photovoltaics: Research and Applications*, 11(3):179–191, 2003, ISSN 1099-159X.
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", *Journal of Statistical Software*, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fPump](#), [prodPVPS](#), [pumpCoef](#)

Examples

```
Pg = seq(4000, 8000, by = 100);
H = seq(120, 150, by = 5);

data(pumpCoef)

CoefSP8A44 <- subset(pumpCoef, Qn == 8 & stages == 44)

NmgSP8A44 <- NmgPVPS(pump = CoefSP8A44, Pg = Pg, H = H, Gd = 5000,
                         title = 'Choice of Pump', theme = custom.theme())
```

C_sample2Diff *Small utilities for difftime objects.*

Description

`diff2Hours` converts a `difftime` object into its numeric value with `units = 'hours'`.
`char2diff` converts a character description into a `difftime` object, following the code of [seq.POSIXt](#).
`sample2Hours` calculates the sampling time in hours described by a character or a `difftime`.
`P2E` (power to energy) sums a series of power values (for example, irradiance) to obtain energy aggregation (for example, irradiation) using `sample2Hours` for the units conversion.

Usage

```
diff2Hours(by)
char2diff(by)
sample2Hours(by)
P2E(x, by)
```

Arguments

<code>by</code>	A character for <code>char2diff</code> , <code>sample2Hours</code> and <code>P2E</code> , or a <code>difftime</code> for <code>diff2Hours</code> , <code>sample2Hours</code> and <code>P2E</code> .
<code>x</code>	A numeric vector.

Value

A numeric value or a `difftime` object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[Sol](#)

Examples

```
char2diff('min')
char2diff('2 s')

sample2Hours('s')
sample2Hours('30 m')

by1 <- char2diff('10 min')
sample2Hours(by1)
```

C_solarAngles	<i>Solar angles</i>
---------------	---------------------

Description

A set of functions that compute the apparent movement of the Sun from the Earth.

Usage

```
## Declination
declination(d, method = 'michalsky')

## Eccentricity
eccentricity(d, method = 'michalsky')

## Equation of time
eot(d)

## Solar time
sunrise(d, lat, method = 'michalsky',
        decl = declination(d, method = method))

## Extraterrestrial irradiation
bo0d(d, lat, method = 'michalsky',
      decl = declination(d, method = method),
      eo = eccentricity(d, method = method),
      ws = sunrise(d, lat, method = method))

## Sun hour angle
sunHour(d, BTi, sample = 'hour', EoT = TRUE,
        method = 'michalsky',
        eqtime = eot(d))

## Cosine of the zenith angle
zenith(d, lat, BTi, sample = 'hour', method = 'michalsky',
       decl = declination(d, method = method),
       w = sunHour(d, BTi, sample, method = method))

## Azimuth angle
azimuth(d, lat, BTi, sample = 'hour', method = 'michalsky',
        decl = declination(d, method = method),
        w = sunHour(d, BTi, sample, method = method),
        cosThzS = zenith(d, lat, BTi, sample,
                          method = method,
                          decl = decl,
                          w = w))
```

Arguments

d Date, a daily time base, it may be the result of [FBTd](#)

<i>C_solarAngles</i>	63
----------------------	----

method	character, method for the sun geometry calculations, to be chosen from 'cooper', 'spencer', 'michalsky' and 'strous'. See references for details.
lat	numeric, latitude (degrees) of the point of the Earth where calculations are needed. It is positive for locations above the Equator.
sample	Character, increment of the intradaily sequence.
BTi	POSIXct, intradaily time base, it may be the result of fBTi .
EoT	logical, if EoT=TRUE (default value), the function sunHour use the Equation of time
decl, eo, ws, eqtime, w, cosThzS	Arguments that compute the variables they reference (default value). It can be replaced with previously calculated values to avoid calculating the same variable twice.

Author(s)

Francisco Delgado López, Oscar Perpiñán Lamigueiro.

References

- Cooper, P.I., Solar Energy, 12, 3 (1969). "The Absorption of Solar Radiation in Solar Stills"
- Spencer, Search 2 (5), 172, <https://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html>
- Strous: <https://www.aa.quae.nl/en/reken/zonpositie.html>
- Michalsky, J., 1988: The Astronomical Almanac's algorithm for approximate solar position (1950-2050), Solar Energy 40, 227-235
- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fSolD](#), [fSolI](#), [calcSol](#)

Examples

```
d = fBTd(mode = 'serie')[100]

decl = declination(d, method = 'michalsky')
decl

w = sunHour(d, sample = 'hour', method = 'michalsky')
w

cosThzS = zenith(d, lat = 37.2, sample = 'hour',
                  method = 'michalsky',
                  decl = decl,
                  w = w)
cosThzS
```

C_utils-angle*Conversion between angle units.***Description**

Several small functions to convert angle units.

Usage

```
d2r(x)
r2d(x)
h2r(x)
h2d(x)
r2h(x)
d2h(x)
r2sec(x)
```

Arguments

x A numeric value.

Value

A numeric value:

d2r: Degrees to radians.

r2d: Radians to degrees.

h2r: Hours to radians.

r2h: Radians to hours.

h2d: Hours to degrees.

d2h: Degrees to hours.

r2sec: Radians to seconds.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

C_utils-time*Utilities for time indexes.***Description**

Several small functions to extract information from POSIXct indexes.

Usage

```
hms(x)
doy(x)
dom(x)
dst(x)
truncDay(x)
```

Arguments

x A POSIXct vector.

Value

doy and dom provide the (numeric) day of year and day of month, respectively.
hms gives the numeric value
 $\text{hour}(x)+\text{minute}(x)/60+\text{second}(x)/3600$
dst is +1 if the Daylight Savings Time flag is in force, zero if not, -1 if unknown ([DateTimeClasses](#)).
truncDay truncates the POSIXct object towards the day.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[as.POSIXct](#)

D_as.data.tableD-methods

Methods for Function as.data.tableD

Description

Convert a Sol, G0, Gef, ProdGCPV or ProdPVPS object into a data.table object with daily values.

Usage

```
## S4 method for signature 'Sol'
as.data.tableD(object, complete=FALSE, day=FALSE)
```

Arguments

object	A Sol object (or extended.)
complete	A logical.
day	A logical.

Methods

`signature(object = "Sol")` Conversion to a `data.table` object with the content of the `solD` slot. If `day=TRUE` (default is `FALSE`), the result includes three columns named `month`, `day` (day of the year) and `year`.

`signature(object = "G0")` If `complete=FALSE` (default) the result includes only the columns of `G0d`, `D0d` and `B0d` from the `G0D` slot. If `complete=TRUE` it returns the contents of the slots `solD` and `G0D`.

`signature(object = "Gef")` If `complete=FALSE` (default) the result includes only the columns of `Gefd`, `Defd` and `Befd` from the `GefD` slot. If `complete=TRUE` it returns the contents of the slots `solD`, `G0D` and `GefD`

`signature(object = "ProdGCPV")` If `complete=FALSE` (default) the result includes only the columns of `Eac`, `Edc` and `Yf` from the `prodD` slot. If `complete=TRUE` it returns the contents of the slots `solD`, `G0D`, `GefD` and `prodD`.

`signature(object = "ProdPVPS")` If `complete=FALSE` (default) the result includes only the columns of `Eac`, `Qd` and `Yf` from the `prodD` slot. If `complete=TRUE` it returns the contents of the slots `solD`, `G0D`, `GefD` and `prodD`.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
lat = 37.2
BTd = fBTd(mode = 'prom')
sol = calcSol(lat, BTd)
solD = as.data.tableD(sol)
solD

solD2 = as.data.tableD(sol, day = TRUE)
solD2

G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,
          2814, 2179)
Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,
          17.2, 15.2)
prom <- list(G0dm = G0dm, Ta = Ta)
prodfixed = prodGCPV(lat, dataRad = prom)
prodD = as.data.tableD(prodfixed, complete = TRUE, day = TRUE)
prodD
```

Description

Convert a `Sol`, `G0`, `Gef`, `ProdGCPV` or `ProdPVPS` object into a `data.table` object with daily values.

Usage

```
## S4 method for signature 'Sol'
as.data.tableI(object, complete=FALSE, day=FALSE)
```

Arguments

object	A Sol object (or extended.)
complete	A logical.
day	A logical.

Methods

`signature(object = "Sol")` If `complete=FALSE` and `day=FALSE` (default) the result includes only the content of the `solI` slot. If `complete=TRUE` the contents of the `solD` slots are included.

`signature(object = "G0")` If `complete=FALSE` and `day=FALSE` (default) the result includes only the columns of `G0`, `D0` and `B0` of the `G0I` slot. If `complete=TRUE` it returns the contents of the slots `G0I` and `solI`. If `day=TRUE` the daily values (slots `G0D` and `solD`) are also included.)

`signature(object = "Gef")` If `complete=FALSE` and `day=FALSE` (default) the result includes only the columns of `Gef`, `Def` and `Bef` of the `GefI` slot. If `complete=TRUE` it returns the contents of the slots `GefI`, `G0I` and `solI`. If `day=TRUE` the daily values (slots `GefD`, `G0D` and `solD`) are also included.)

`signature(object = "ProdGCPV")` If `complete=FALSE` and `day=FALSE` (default) the result includes only the columns of `Pac` and `Pdc` of the `prodI` slot. If `complete=TRUE` it returns the contents of the slots `prodI`, `GefI`, `G0I` and `solI`. If `day=TRUE` the daily values (slots `prodD`, `GefD`, `G0D` and `solD`) are also included.)

`signature(object = "ProdPVPS")` If `complete=FALSE` and `day=FALSE` (default) the result includes only the columns of `Pac` and `Q` of the `prodI` slot. If `complete=TRUE` it returns the contents of the slots `prodI`, `GefI`, `G0I` and `solI`. If `day=TRUE` the daily values (slots `prodD`, `GefD`, `G0D` and `solD`) are also included.)

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
lat = 37.2
BTd = fBTd(mode = 'prom')[1]
sol = calcSol(lat, BTd, keep.night = FALSE)
solI = as.data.tableI(sol)
solI

solI2 = as.data.tableI(sol, day = TRUE)
solI2

G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,
         2814, 2179)
Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,
       17.2, 15.2)
prom <- list(G0dm = G0dm, Ta = Ta)
prodfixed = prodGCPV(lat, dataRad = prom)
prodI = as.data.tableI(prodfixed, complete = TRUE, day = TRUE)
prodI
```

D_as.data.tableM-methods*Methods for Function as.data.tableM***Description**

Convert a G0, Gef, ProdGCPV or ProdPVPS object into a as.data.table object with monthly average of daily values.

Usage

```
## S4 method for signature 'G0'
as.data.tableM(object, complete=FALSE, day=FALSE)
```

Arguments

object	A G0 object (or extended.)
complete	A logical.
day	A logical

Methods

`signature(object = "G0")` The result is the G0dm slot. If day=TRUE (default is FALSE), the result includes two columns names month and year.

`signature(object = "Gef")` If complete=FALSE (default) the result is the slot Gefdm. If complete=TRUE it returns the slot G0dm.

`signature(object = "ProdGCPV")` If complete=FALSE (default) the result is the prodDm slot. If complete=TRUE the result includes the slots G0dm and Gefdm.

`signature(object = "ProdPVPS")` If complete=FALSE (default) the result is the prodDm slot. If complete=TRUE the result includes the slots G0dm and Gefdm.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
lat = 37.2
G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,
         2814, 2179)
Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,
       17.2, 15.2)
prom <- list(G0dm = G0dm, Ta = Ta)
prodfixed = prodGCPV(lat, dataRad = prom)
prodM = as.data.tableM(prodfixed, complete = TRUE, day = TRUE)
prodM
```

D_as.data.tableY-methods*Methods for Function as.data.tableY*

Description

Convert a G0, Gef, ProdGCPV or ProdPVPS object into a data.table object with yearly values.

Usage

```
## S4 method for signature 'G0'  
as.data.tableY(object, complete=FALSE, day=FALSE)
```

Arguments

object	A G0 object (or extended.)
complete	A logical.
day	A logical.

Methods

`signature(object = "G0")` The result is the G0y slot. If day = TRUE (default is FALSE), the result includes a column named year.

`signature(object = "Gef")` If complete=FALSE (default) the result is the slot Gefy. If complete=TRUE it returns the slot G0y.

`signature(object = "ProdGCPV")` If complete=FALSE (default) the result is the prody slot. If complete=TRUE the result includes the slots G0y and Gefy.

`signature(object = "ProdPVPS")` If complete=FALSE (default) the result is the prody slot. If complete=TRUE the result includes the slots G0y and Gefy.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
lat = 37.2  
G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562,  
        2814, 2179)  
Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2,  
        17.2, 15.2)  
prom <- list(G0dm = G0dm, Ta = Ta)  
prodfixed = prodGCPV(lat, dataRad = prom)  
prodY = as.data.tableY(prodfixed, complete = TRUE, day = TRUE)  
prodY
```

D_compare-methods *Compare G0, Gef and ProdGCPV objects*

Description

Compare and plot the yearly values of several objects.

Usage

```
## S4 method for signature 'G0'
compare(...)
```

Arguments

... A list of objects to be compared.

Methods

The class of the first element of ... is used to determine the suitable method. The result is plotted with [dotplot](#):

```
signature(... = "G0") yearly values of G0d, B0d and D0d.
signature(... = "Gef") yearly values of Gefd, Befd and Defd.
signature(... = "ProdGCPV") yearly values of Yf, Gefd and G0d.
```

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[dotplot](#)

Examples

```
lat = 37.2;
G0dm = c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562, 2814,
2179)
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom = list(G0dm = G0dm, Ta = Ta)

###Comparison of different tracker methods
ProdFixed <- prodGCPV(lat = lat, dataRad = prom, keep.night = FALSE)
Prod2x <- prodGCPV(lat = lat, dataRad = prom, modeTrk = 'two', keep.night = FALSE)
ProdHoriz <- prodGCPV(lat = lat, dataRad = prom, modeTrk = 'horiz', keep.night = FALSE)

compare(ProdFixed, Prod2x, ProdHoriz)

##The first element rules the method
GefFixed = as(ProdFixed, 'Gef')
compare(GefFixed, Prod2x, ProdHoriz)
```

D_getData-methods *Methods for function getData*

Description

Meteorological source data of a Meteo (or extended) object.

Methods

`signature(object = "Meteo")` returns the meteorological source data of the slot data of the object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

D_getG0-methods *Methods for function getG0*

Description

Global irradiation source data of a Meteo (or extended) object.

Methods

`signature(object = "Meteo")` returns the global irradiation values stored in a Meteo object.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

D_getLat-methods *Methods for Function getLat*

Description

Latitude angle of solaR objects.

Usage

`getLat(object, units='rad')`

Arguments

object A Sol or Meteo object (or extended.)
units A character, 'rad' or 'deg'.

Methods

This function returns the latitude angle in radians (`units='rad'`, default) or degrees (`units='deg'`).

`signature(object = "Meteo")` Value of the `latData` slot, which is defined by the argument `lat` of the `readG0dm` and `readBDd` functions, or by the `lat` component of the `dataRad` object passed to `calcG0` (or equivalent). It is the latitude of the meteorological station (or equivalent) which provided the irradiation source data. It may be different from the value used for the calculation procedure.

`signature(object = "Sol")` Value of the `lat` slot, which is defined by the argument `lat` of the `calcSol` function. It is the value used through the calculation procedure.

`signature(object = "G0")` same as for the `Sol` class.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Description

Daily time index of `solaR` objects.

Methods

`signature(object = "Meteo")` returns the index of the `data` slot (a `data.table` object.)

`signature(object = "Sol")` returns the index of the `solD` slot (a `data.table` object.)

`signature(object = "G0")` same as for `object='Sol'`

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Description

Intra-daily time index of `solaR` objects.

Methods

`signature(object = "Sol")` returns the index of the slot `solI` (a `data.table` object).

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

D_levelplot-methods *Methods for function levelplot.*

Description

Methods for function `levelplot` and `zoo` and `solaR` objects.

Methods

```
signature(x = "formula", data = "Meteo"): The Meteo object is converted into a data.table  
object, and the previous method is used.  
signature(x = "formula", data = "Sol"): idem  
signature(x = "formula", data = "G0"): idem
```

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

D_Losses-methods *Losses of a GCPV system*

Description

The function `losses` calculates the yearly losses from a `Gef` or a `ProdGCPV` object. The function `compareLosses` compares the losses from several `ProdGCPV` objects and plots the result with [dotplot](#).

Usage

```
compareLosses(...)  
losses(object)
```

Arguments

...	A list of <code>ProdGCPV</code> objects to be compared.
object	An object of <code>Gef</code> or <code>ProdGCPV</code> class..

Methods

```
signature(... = "Gef") shadows and angle of incidence (AoI) losses.  
signature(... = "ProdGCPV") shadows, AoI, generator (mainly temperature), DC and AC sys-  
tem (as detailed in effSys of fProd) and inverter losses.
```

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

See Also

[fInclin](#), [fProd](#)

Examples

```
lat = 37.2;
G0dm = c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562, 2814,
2179)
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom = list(G0dm = G0dm, Ta = Ta)

###Comparison of different tracker methods
ProdFixed <- prodGCPV(lat = lat,dataRad = prom, keep.night = FALSE)
Prod2x <- prodGCPV(lat = lat, dataRad = prom, modeTrk = 'two', keep.night = FALSE)
ProdHoriz <- prodGCPV(lat = lat,dataRad = prom, modeTrk = 'horiz', keep.night = FALSE)

losses(ProdFixed)
losses(as(ProdFixed, 'Gef'))

compareLosses(ProdFixed, Prod2x, ProdHoriz)
```

D_mergesolaR-methods *Merge solaR objects*

Description

Merge the daily time series of solaR objects

Usage

```
## S4 method for signature 'G0'
mergesolaR(...)
```

Arguments

... A list of objects to be merged.

Methods

The class of the first element of ... is used to determine the suitable method. Only the most important daily variable is merged, depending on the class of the objects:

```
signature(... = "Meteo") G0
signature(... = "G0") G0d
signature(... = "Gef") Gefd
signature(... = "ProdGCPV") Yf
signature(... = "ProdPVPS") Yf
```

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

Examples

```
lat = 37.2;
G0dm = c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562, 2814,
2179)
Ta = c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom = list(G0dm = G0dm, Ta = Ta)

###Different tracker methods
ProdFixed <- prodGCPV(lat = lat,dataRad = prom, keep.night = FALSE)
Prod2x <- prodGCPV(lat = lat, dataRad = prom, modeTrk = 'two', keep.night = FALSE)
ProdHoriz <- prodGCPV(lat = lat,dataRad = prom, modeTrk = 'horiz', keep.night = FALSE)

prod <- mergesolaR(ProdFixed, Prod2x, ProdHoriz)
head(prod)
```

D_shadeplot-methods Methods for Function shadeplot

Description

Visualization of the content of a [Shade](#) object.

Methods

`signature(x = "Shade")` display the results of the iteration with a level plot for the two-axis tracking, or with conventional plot for horizontal tracking and fixed systems.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

D_window-methods Methods for extracting a time window

Description

Method for extracting the subset of a `solaR` object whose daily time index (`indexD`) is comprised between the times `i` and `j`.

Usage

```
## S4 method for signature 'Meteo'
x[i, j, ... , drop = TRUE]
## S4 method for signature 'Sol'
x[i, j, ... , drop = TRUE]
## S4 method for signature 'G0'
x[i, j, ... , drop = TRUE]
## S4 method for signature 'Gef'
x[i, j, ... , drop = TRUE]
## S4 method for signature 'ProdGCPV'
x[i, j, ... , drop = TRUE]
## S4 method for signature 'ProdPVPS'
x[i, j, ... , drop = TRUE]
```

Arguments

x	A Meteo, Sol, etc. object.
i	an index/time value (Date or POSIXct classes) defining the start of the time window.
j	an index/time value (Date or POSIXct classes) defining the end of the time window.
..., drop	Additional arguments for <code>window.zoo</code>

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

[indexD](#)

Examples

```
lat = 37.2
sol = calcSol(lat, BTd = fBTd(mode = 'serie'))
range(indexD(sol))

start <- as.Date(indexD(sol)[1])
end <- start + 30

solWindow <- sol[start, end]
range(indexD(solWindow))
```

Description

Exports the results of the solaR functions as text files using [write.table](#)

Usage

```
## S4 method for signature 'Sol'
writeSolar(object, file, complete = FALSE,
           day = FALSE, timeScales = c('i', 'd', 'm', 'y'), sep = ',', ...)
```

Arguments

object	A Sol object (or extended.)
file	A character with the name of the file.
complete	A logical. Should all the variables be exported?
day	A logical. Should be daily values included in the intradaily file?
timeScales	A character. Use 'i' to export intradaily values, 'd' for daily values, 'm' for monthly values and 'y' for yearly values. A different file will be created for each choice.
sep	The field separator character.
...	Additional arguments for <code>write.table</code>

Methods

`signature(object = "Sol")` This function exports the slots with results using `write.table`. If `complete = FALSE` and `day = FALSE` (default) the result includes only the content of the `solI` slot. If `day = TRUE` the contents of the `solD` slot are included.

`signature(object = "G0")` If `complete = FALSE` and `day = FALSE` (default) the result includes only the columns of `G0`, `D0` and `B0` of the `G0I` slot. If `complete = TRUE` it returns the contents of the slots `G0I` and `solI`. If `day = TRUE` the daily values (slots `G0D` and `solD`) are also included.

`signature(object = "Gef")` If `complete = FALSE` and `day = FALSE` (default) the result includes only the columns of `Gef`, `Def` and `Bef` of the `GefI` slot. If `complete = TRUE` it returns the contents of the slots `GefI`, `G0I` and `solI`. If `day = TRUE` the daily values (slots `GefD`, `G0D` and `solD`) are also included.

`signature(object = "ProdGCPV")` If `complete = FALSE` and `day = FALSE` (default) the result includes only the columns of `Pac` and `Pdc` of the `prodI` slot. If `complete = TRUE` it returns the contents of the slots `prodI`, `GefI`, `G0I` and `solI`. If `day = TRUE` the daily values (slots `prodD`, `GefD`, `G0D` and `solD`) are also included.

`signature(object = "ProdPVPS")` If `complete = FALSE` and `day = FALSE` (default) the result includes only the columns of `Pac` and `Q` of the `prodI` slot. If `complete = TRUE` it returns the contents of the slots `prodI`, `GefI`, `G0I` and `solI`. If `day = TRUE` the daily values (slots `prodD`, `GefD`, `G0D` and `solD`) are also included.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

See Also

`write.table`, `fread`, `as.data.tableI`, `as.data.tableD`, `as.data.tableM`, `as.data.tableY`

Examples

```

lat <- 37.2;
G0dm <- c(2766, 3491, 4494, 5912, 6989, 7742, 7919, 7027, 5369, 3562, 2814, 2179)
Ta <- c(10, 14.1, 15.6, 17.2, 19.3, 21.2, 28.4, 29.9, 24.3, 18.2, 17.2, 15.2)
prom <- list(G0dm = G0dm, Ta = Ta)

prodFixed <- prodGCPV(lat = lat, dataRad = prom, modeRad = 'aguiar', keep.night = FALSE)

old <- setwd(tempdir())

writeSolar(prodFixed, 'prodFixed.csv')

dir()

zI <- fread("prodFixed.csv",
            header = TRUE, sep = ",")
zI

zD <- fread("prodFixed.D.csv",
            header = TRUE, sep = ",")
zD

zM <- fread("prodFixed.M.csv",
            header = TRUE, sep = ",")
zM

zY <- fread("prodFixed.Y.csv",
            header = TRUE, sep = ",")
zY

setwd(old)

```

Description

Methods for function `xyplot` in Package ‘`solaR`’

Methods

`signature(x = "data.table", data = "missing")`: This method creates an XY plot for objects of class `data.table` without specifying a `data` argument. It must contain a column named `Dates` with the time information.

`signature(x = "formula", data = "Meteo")`: The `Meteo` object is converted into a `data.table` object with `getData(x)` and displayed with the method for `data.table`.

`signature(x = "formula", data = "Sol")`: The `Sol` object is converted into a `data.table` object with `as.data.tableI(x, complete = TRUE, day = TRUE)` and displayed with the method for `data.table`.

`signature(x = "formula", data = "G0")`: Idem.

`signature(x = "Meteo", data = "missing")`: The Meteo object is converted into a `data.table` object with `getData(data)`. This `data.table` is the `x` argument for a call to `xyplot`, using the S4 method for `signature(x = "data.table", data = "missing")`.

`signature(x = "G0", data = "missing")`: The G0 object is converted into a `data.table` object with `indexD(data)`. This `data.table` is the `x` argument for a call to `xyplot`, using the S4 method for `signature(x = 'data.table', data = 'missing')`.

`signature(x = "ProdGCPV", data = "missing")`: Idem, but the variables are not superposed.

`signature(x = "ProdPVPS", data = "missing")`: Idem.

`signature(x = "formula", data = "Shade")`: Idem.

Author(s)

Oscar Perpiñán Lamigueiro, Francisco Delgado López.

E_aguiar

Markov Transition Matrices for the Aguiar et al. procedure

Description

Markov Transition Matrices and auxiliary data for generating sequences of daily radiation values.

Usage

`data(MTM)`

Format

MTM is a `data.frame` with the collection of Markov Transition Matrices defined in the paper "Simple procedure for generating sequences of daily radiation values using a library of Markov transition matrices", Aguiar et al., Solar Energy, 1998. `Ktlim` (matrix) and `Ktmtm` (vector) are auxiliary data to choose the correspondent matrix of the collection.

Author(s)

Oscar Perpiñán Lamiguero, Francisco Delgado López.

E_helios

Daily irradiation and ambient temperature from the Helios-IES database

Description

A year of irradiation, maximum and minimum ambient temperature from the HELIOS-IES database.

Usage

`data(helios)`

Format

A data frame with 355 observations on the following 4 variables:

`yyyy.mm.dd` a factor: year, month and day.
`G.θ.` a numeric vector, daily global horizontal irradiation.
`TambMax` a numeric vector, maximum ambient temperature.
`TambMin` a numeric vector, minimum ambient temperature.

Source

<http://helios.ies-def.upm.es/consulta.aspx>

E_prodEx

Productivity of a set of PV systems of a PV plant.

Description

A `data.table` object with the time evolution of the final productivity of a set of 22 systems of a large PV plant.

Usage

```
data(prodEx)
```

References

O. Perpiñán, Statistical analysis of the performance and simulation of a two-axis tracking PV system, Solar Energy, 83:11(2074–2085), 2009.https://oa.upm.es/1843/1/PERPINAN_ART2009_01.pdf

E_pumpCoef

Coefficients of centrifugal pumps.

Description

Coefficients of centrifugal pumps

Usage

```
data(pumpCoef)
```

Format

A `data.table` with 13 columns:

Qn rated flux

stages number of stages

Qmax maximum flux

Pmn rated motor power

a, b, c Coefficients of the equation $H = a \cdot f^2 + b \cdot f \cdot Q + c \cdot Q^2$.

g, h, i Coefficients of the efficiency curve of the motor (50 Hz): $\eta_m = g \cdot (\%P_{mn})^2 + h \cdot (\%P_{mn}) + i$.

j, k, l Coefficients of the efficiency curve of the pump (50 Hz): $\eta_b = j \cdot Q^2 + k \cdot Q + l$.

Details

With this version only pumps from the manufacturer Grundfos are included.

Source

<https://product-selection.grundfos.com/>

References

- Perpiñán, O, Energía Solar Fotovoltaica, 2015. (<https://oscarperpinan.github.io/esf/>)
- Perpiñán, O. (2012), "solaR: Solar Radiation and Photovoltaic Systems with R", Journal of Statistical Software, 50(9), 1-32, doi:[10.18637/jss.v050.i09](https://doi.org/10.18637/jss.v050.i09)

Description

Information about the location and operational status of the stations that make up the SIAR network

Usage

```
data(SIAR)
```

Format

`est_SIAR` is a `data.table` with 625 estations containing the following information:

`Estacion` character, name of the station.

`Codigo` character, code of the station.

`Longitud` numeric, longitude of the station in degrees (negative is for locations in the west).

`Latitud` numeric, latitud of the station in degrees.

`Altitud` integer, altitude of the station in meters.

`Fecha_Instalacion` Date, day the station was installed, and therefore, the start of its records.

`Fecha_Baja` Date, day the station was decommissioned, and therefore, the end of its records (if its value is NA, it means it is still operational).

Source

<https://servicio.mapa.gob.es/websiar/>

E_solaR.theme

solaR theme

Description

A customized theme for lattice. It is based on the `custom.theme.2` function of the `latticeExtra` package with the next values:

- `pch = 19`
- `cex = 0.7`
- `region = rev(brewer.pal(9, 'YlOrRd'))`
- `strip.background$col = 'lightgray'`
- `strip.shingle$col = 'transparent'`

Index

* **classes**
 B1_Meteo-class, 27
 B2_Sol-class, 28
 B3_G0-class, 29
 B4_Gef-class, 30
 B5_ProdGCPV-class, 32
 B6_ProdPVPS-class, 33
 B7_Shade-class, 34

* **constructors**
 A1_calcSol, 5
 A2_calcG0, 6
 A3_calcGef, 9
 A4_prodGCPV, 11
 A5_prodPVPS, 15
 A6_calcShd, 17
 A7_optimShd, 18
 A8_readBD, 23
 A8_readG0dm, 25
 A8_readSIAR, 26

* **datasets**
 E_aguiar, 79
 E_helios, 79
 E_prodEx, 80
 E_pumpCoef, 80
 E_SIAR, 81
 E_solaR.theme, 82

* **methods**
 D_as.data.tableD-methods, 65
 D_as.data.tableI-methods, 66
 D_as.data.tableM-methods, 68
 D_as.data.tableY-methods, 69
 D_compare-methods, 70
 D_getData-methods, 71
 D_getG0-methods, 71
 D_getLat-methods, 71
 D_indexD-methods, 72
 D_indexI-methods, 72
 D_levelplot-methods, 73
 D_Losses-methods, 73
 D_mergesolaR-methods, 74
 D_shadeplot-methods, 75
 D_window-methods, 75
 D_writeSolar-methods, 76

 D_xyplot-methods, 78

* **utilities**
 C_corrFdKt, 36
 C_fBTd, 38
 C_fBTi, 39
 C_fCompD, 40
 C_fCompI, 41
 C_fInclin, 43
 C_fProd, 45
 C_fPump, 47
 C_fSo1D, 48
 C_fSo1I, 50
 C_fSombra, 52
 C_fTemp, 54
 C_fTheta, 55
 C_HQCurve, 57
 C_local2Solar, 58
 C_NmgPVPS, 59
 C_sample2Diff, 61
 C_solarAngles, 62
 C_utils-angle, 64
 C_utils-time, 64
 [], G0, ANY, ANY-method (D_window-methods), 75
 [], G0-method (D_window-methods), 75
 [], Gef, ANY, ANY-method (D_window-methods), 75
 [], Gef-method (D_window-methods), 75
 [], Meteo, ANY, ANY-method (D_window-methods), 75
 [], Meteo-method (D_window-methods), 75
 [], ProdGCPV, ANY, ANY-method (D_window-methods), 75
 [], ProdPVPS-method (D_window-methods), 75
 [], ProdPVPS, ANY, ANY-method (D_window-methods), 75
 [], ProdPVPS-method (D_window-methods), 75
 [], Sol, ANY, ANY-method (D_window-methods), 75
 [], Sol-method (D_window-methods), 75

 A1_calcSol, 5
 A2_calcG0, 6
 A3_calcGef, 9

A4_prodGCPV, 11
 A5_prodPVPS, 15
 A6_calcShd, 17
 A7_optimShd, 18
 A8_Meteo2Meteo, 22
 A8_readBD, 23
 A8_readG0dm, 25
 A8_readSIAR, 26
 aguiar (E_aguiar), 79
 as.data.frame, Shade-method
 (B7_Shade-class), 34
 as.data.tableD, 77
 as.data.tableD
 (D_as.data.tableD-methods), 65
 as.data.tableD, G0-method
 (D_as.data.tableD-methods), 65
 as.data.tableD, Gef-method
 (D_as.data.tableD-methods), 65
 as.data.tableD, ProdGCPV-method
 (D_as.data.tableD-methods), 65
 as.data.tableD, ProdPVPS-method
 (D_as.data.tableD-methods), 65
 as.data.tableD, Sol-method
 (D_as.data.tableD-methods), 65
 as.data.tableD-methods
 (D_as.data.tableD-methods), 65
 as.data.tableI, 77
 as.data.tableI
 (D_as.data.tableI-methods), 66
 as.data.tableI, G0-method
 (D_as.data.tableI-methods), 66
 as.data.tableI, Gef-method
 (D_as.data.tableI-methods), 66
 as.data.tableI, ProdGCPV-method
 (D_as.data.tableI-methods), 66
 as.data.tableI, ProdPVPS-method
 (D_as.data.tableI-methods), 66
 as.data.tableI, Sol-method
 (D_as.data.tableI-methods), 66
 as.data.tableI-methods
 (D_as.data.tableI-methods), 66
 as.data.tableM, 77
 as.data.tableM
 (D_as.data.tableM-methods), 68
 as.data.tableM, G0-method
 (D_as.data.tableM-methods), 68
 as.data.tableM, Gef-method
 (D_as.data.tableM-methods), 68
 as.data.tableM, ProdGCPV-method
 (D_as.data.tableM-methods), 68
 as.data.tableM, ProdPVPS-method
 (D_as.data.tableM-methods), 68

as.data.tableM-methods
 (D_as.data.tableM-methods), 68
 as.data.tableY, 77
 as.data.tableY
 (D_as.data.tableY-methods), 69
 as.data.tableY, G0-method
 (D_as.data.tableY-methods), 69
 as.data.tableY, Gef-method
 (D_as.data.tableY-methods), 69
 as.data.tableY, ProdGCPV-method
 (D_as.data.tableY-methods), 69
 as.data.tableY, ProdPVPS-method
 (D_as.data.tableY-methods), 69
 as.data.tableY-methods
 (D_as.data.tableY-methods), 69
 as.POSIXct, 38
 azimuth (C_solarAngles), 62

 B1_Meteo-class, 27
 B2_Sol-class, 28
 B3_G0-class, 29
 B4_Gef-class, 30
 B5_ProdGCPV-class, 32
 B6_ProdPVPS-class, 33
 B7_Shade-class, 34
 bo0d (C_solarAngles), 62

 C_corrFdKt, 36
 C_fBTd, 38
 C_fBTi, 39
 C_fCompD, 40
 C_fCompI, 41
 C_fInclin, 43
 C_fProd, 45
 C_fPump, 47
 C_fSolD, 48
 C_fSolI, 50
 C_fSombra, 52
 C_fTemp, 54
 C_fTheta, 55
 C_HQCurve, 57
 C_local2Solar, 58
 C_NmgPVPS, 59
 C_sample2Diff, 61
 C_solarAngles, 62
 C_utils-angle, 64
 C_utils-time, 64
 calcG0, 9, 10, 12, 13, 16, 18, 19, 24, 29, 43, 50
 calcG0 (A2_calcG0), 6
 calcGef, 12, 13, 16, 17, 19, 30, 43, 44, 55, 57
 calcGef (A3_calcGef), 9
 calcShd, 9, 10, 13, 18, 21, 54
 calcShd (A6_calcShd), 17

INDEX

85

calcSol, 6–10, 12, 16, 19, 28, 36, 40–42, 54–56, 63
 calcSol (A1_calcSol), 5
 char2diff (C_sample2Diff), 61
 compare, 13
 compare (D_compare-methods), 70
 compare, G0-method (D_compare-methods), 70
 compare, Gef-method (D_compare-methods), 70
 compare, ProdGCPV-method (D_compare-methods), 70
 compare-methods (D_compare-methods), 70
 compareLosses, 13
 compareLosses (D_Losses-methods), 73
 compareLosses, ProdGCPV-method (D_Losses-methods), 73
 compareLosses-methods (D_Losses-methods), 73
 corrFdKt, 7, 8, 40, 42
 corrFdKt (C_corrFdKt), 36
 d2h (C_utils-angle), 64
 d2r (C_utils-angle), 64
 D_as.data.tableD-methods, 65
 D_as.data.tableI-methods, 66
 D_as.data.tableM-methods, 68
 D_as.data.tableY-methods, 69
 D_compare-methods, 70
 D_getData-methods, 71
 D_getG0-methods, 71
 D_getLat-methods, 71
 D_indexD-methods, 72
 D_indexI-methods, 72
 D_levelplot-methods, 73
 D_Losses-methods, 73
 D_mergesolaR-methods, 74
 D_shadeplot-methods, 75
 D_window-methods, 75
 D_writeSolar-methods, 76
 D_xyplot-methods, 78
 DateTimeClasses, 65
 declination (C_solarAngles), 62
 diff2Hours (C_sample2Diff), 61
 dom (C_utils-time), 64
 dotplot, 70, 73
 doy (C_utils-time), 64
 dst (C_utils-time), 64
 dt2Meteo, 7, 8, 27, 42
 dt2Meteo (A8_readBD), 23
 E_aguiar, 79
 E_helios, 79
 E_prodEx, 80
 E_pumpCoef, 80
 E_SIAR, 81
 E_solaR.theme, 82
 eccentricity (C_solarAngles), 62
 eot, 58
 eot (C_solarAngles), 62
 est_SIAR (E_SIAR), 81
 fBTd, 5, 6, 49, 62
 fBTd (C_fBTd), 38
 fBTi, 5, 63
 fBTi (C_fBTi), 39
 fCompD, 6–8, 29, 36, 37, 42
 fCompD (C_fCompD), 40
 fCompI, 6–8, 29, 36, 37, 41, 44
 fCompI (C_fCompI), 41
 FdKtBRL (C_corrFdKt), 36
 FdKtCLIMEd (C_corrFdKt), 36
 FdKtCLIMEdh, 42
 FdKtCLIMEdh (C_corrFdKt), 36
 FdKtCPR, 7, 40
 FdKtCPR (C_corrFdKt), 36
 FdKtEKDd (C_corrFdKt), 36
 FdKtEKDh (C_corrFdKt), 36
 FdKtLJ (C_corrFdKt), 36
 FdKtPage, 7, 40
 FdKtPage (C_corrFdKt), 36
 fInclin, 9, 10, 13, 18, 30, 46, 57, 74
 fInclin (C_fInclin), 43
 fProd, 13, 32, 73, 74
 fProd (C_fProd), 45
 fPump, 16, 17, 34, 60
 fPump (C_fPump), 47
 fread, 24, 25, 77
 fSolD, 5, 7, 9, 12, 16, 19, 28, 38, 40, 51, 63
 fSolD (C_fSolD), 48
 fSolI, 5, 7, 9, 12, 16, 19, 28, 41, 42, 63
 fSolI (C_fSolI), 50
 fSombra, 17, 53, 57
 fSombra (C_fSombra), 52
 fSombra2X (C_fSombra), 52
 fSombra6, 17, 20, 53
 fSombra6 (C_fSombra), 52
 fSombraEst, 53
 fSombraEst (C_fSombra), 52
 fSombraHoriz, 53
 fSombraHoriz (C_fSombra), 52
 fTemp, 6, 24, 46
 fTemp (C_fTemp), 54
 fTheta, 9, 10, 13, 18, 30, 43, 44, 54
 fTheta (C_fTheta), 55

G0, 28, 31–35
 G0-class (B3_G0-class), 29
 Gef, 17, 28, 30, 32–35, 45
 Gef-class (B4_Gef-class), 30
 getData (D_getData-methods), 71
 getData, Meteo-method
 (D_getData-methods), 71
 getData-methods (D_getData-methods), 71
 getG0 (D_getG0-methods), 71
 getG0, Meteo-method (D_getG0-methods), 71
 getG0-methods (D_getG0-methods), 71
 getLat (D_getLat-methods), 71
 getLat, G0-method (D_getLat-methods), 71
 getLat, Meteo-method (D_getLat-methods),
 71
 getLat, Sol-method (D_getLat-methods), 71
 getLat-methods (D_getLat-methods), 71

 h2d (C_utils-angle), 64
 h2r (C_utils-angle), 64
 helios (E_helios), 79
 hms (C_utils-time), 64
 HQCurve (C_HQCurve), 57

 indexD, 75, 76
 indexD (D_indexD-methods), 72
 indexD, G0-method (D_indexD-methods), 72
 indexD, Meteo-method (D_indexD-methods),
 72
 indexD, Sol-method (D_indexD-methods), 72
 indexD-methods (D_indexD-methods), 72
 indexI (D_indexI-methods), 72
 indexI, Sol-method (D_indexI-methods), 72
 indexI-methods (D_indexI-methods), 72

 Ktd (C_corrFdKt), 36
 Kti (C_corrFdKt), 36
 Ktlim (E_aguiar), 79
 Ktm (C_corrFdKt), 36
 Ktmtm (E_aguiar), 79

 levelplot, formula, G0-method
 (D_levelplot-methods), 73
 levelplot, formula, Meteo-method
 (D_levelplot-methods), 73
 levelplot, formula, Sol-method
 (D_levelplot-methods), 73
 levelplot, formula, zoo-method
 (D_levelplot-methods), 73
 levelplot-methods
 (D_levelplot-methods), 73
 local2Solar (C_local2Solar), 58
 lonHH (C_local2Solar), 58

losses (D_Losses-methods), 73
 losses, Gef-method (D_Losses-methods), 73
 losses, ProdGCPV-method
 (D_Losses-methods), 73
 losses-methods (D_Losses-methods), 73

 mergesolaR, 13
 mergesolaR (D_mergesolaR-methods), 74
 mergesolaR, G0-method
 (D_mergesolaR-methods), 74
 mergesolaR, Gef-method
 (D_mergesolaR-methods), 74
 mergesolaR, Meteo-method
 (D_mergesolaR-methods), 74
 mergesolaR, ProdGCPV-method
 (D_mergesolaR-methods), 74
 mergesolaR, ProdPVPS-method
 (D_mergesolaR-methods), 74
 mergesolaR-methods
 (D_mergesolaR-methods), 74
 Meteo, 29, 31, 32, 34–36, 54
 Meteo-class (B1_Meteo-class), 27
 Meteo2Meteom (A8_Meteo2Meteo), 22
 Meteo2Meteod (A8_Meteo2Meteo), 22
 MTM (E_aguiar), 79

 NmgPVPS, 17, 48, 57
 NmgPVPS (C_NmgPVPS), 59

 optimShd, 34, 54
 optimShd (A7_optimShd), 18

 P2E (C_sample2Diff), 61
 prodEx (E_prodEx), 80
 ProdGCPV, 35
 prodGCPV, 21, 32, 45, 46
 prodGCPV (A4_prodGCPV), 11
 ProdGCPV-class (B5_ProdGCPV-class), 32
 ProdPVPS, 16
 prodPVPS, 33, 34, 48, 57, 60
 prodPVPS (A5_prodPVPS), 15
 ProdPVPS-class (B6_ProdPVPS-class), 33
 pumpCoef, 16, 17, 33, 47, 48, 57, 59, 60
 pumpCoef (E_pumpCoef), 80

 r2d (C_utils-angle), 64
 r2h (C_utils-angle), 64
 r2sec (C_utils-angle), 64
 readBDd, 6–8, 23, 26, 27, 36, 40, 54, 55, 72
 readBDd (A8_readBD), 23
 readBDi, 7, 8, 27, 36, 42
 readBDi (A8_readBD), 23
 readG0dm, 7, 8, 23, 25–27, 36, 40, 72

INDEX

87

readG0dm (A8_readG0dm), 25
readSIAR, 23
readSIAR (A8_readSIAR), 26

sample2Hours (C_sample2Diff), 61
seq.POSIXt, 5, 7, 19, 38, 50, 61
Shade, 21, 33, 75
Shade-class (B7_Shade-class), 34
shadeplot (D_shadeplot-methods), 75
shadeplot, Shade-method
 (D_shadeplot-methods), 75
shadeplot-methods
 (D_shadeplot-methods), 75
show, G0-method (B3_G0-class), 29
show, Gef-method (B4_Gef-class), 30
show, Meteo-method (B1_Meteo-class), 27
show, ProdGCPV-method
 (B5_ProdGCPV-class), 32
show, ProdPVPS-method
 (B6_ProdPVPS-class), 33
show, Shade-method (B7_Shade-class), 34
show, Sol-method (B2_Sol-class), 28
Sol, 29–36, 54, 61
Sol-class (B2_Sol-class), 28
solaR.theme (E_solaR.theme), 82
solaR2 (solaR2-package), 3
solaR2-package, 3
sunHour (C_solarAngles), 62
sunrise (C_solarAngles), 62

truncDay (C_utils-time), 64

window (D_window-methods), 75
window-methods (D_window-methods), 75
write.table, 76, 77
writeSolar (D_writeSolar-methods), 76
writeSolar, Sol-method
 (D_writeSolar-methods), 76
writeSolar-methods
 (D_writeSolar-methods), 76

xyplot, data.table, missing-method
 (D_xyplot-methods), 78
xyplot, formula, G0-method
 (D_xyplot-methods), 78
xyplot, formula, Meteo-method
 (D_xyplot-methods), 78
xyplot, formula, Shade-method
 (D_xyplot-methods), 78
xyplot, formula, Sol-method
 (D_xyplot-methods), 78
xyplot, G0, missing-method
 (D_xyplot-methods), 78
xyplot, Meteo, missing-method
 (D_xyplot-methods), 78
xyplot, ProdGCPV, missing-method
 (D_xyplot-methods), 78
xyplot, ProdPVPS, missing-method
 (D_xyplot-methods), 78
xyplot-methods (D_xyplot-methods), 78

zenith (C_solarAngles), 62
zoo2Meteo, 7, 27, 42
zoo2Meteo (A8_readBD), 23

B

APÉNDICE

Código completo

En este apéndice se incluye el código completo. Todo el código aquí mostrado está disponible en [GitHub](#). Para facilitar la consulta, se va a seguir el mismo orden que en el manual de referencia (apéndice A).

B.1. calcSol

```
1 calcSol <- function(lat, BTd,
2                         sample = 'hour', BTi,
3                         EoT = TRUE,
4                         keep.night = TRUE,
5                         method = 'michalsky')
6 {
7   if(missing(BTd)) BTd <- truncDay(BTi)
8   solD <- fSolD(lat, BTd, method = method)
9   solI <- fSolI(solD = solD, sample = sample,
10                  BTi = BTi, keep.night = keep.night,
11                  EoT = EoT, method = method)
12
13  if(!missing(BTi)){
14    sample <- median(diff(solI$Dates))
15    sample <- format(sample)
16  }
17
18  solD[, lat := NULL]
19  solI[, lat := NULL]
20  result <- new('Sol',
21                 lat = lat,
22                 solD = solD,
23                 solI = solI,
24                 sample = sample,
25                 method = method)
26  return(result)
27 }
```

B.2. calcG0

```
1 calcG0 <- function(lat,
2                      modeRad = 'prom',
```

B. CÓDIGO COMPLETO

```
3     dataRad,
4     sample = 'hour',
5     keep.night = TRUE,
6     sunGeometry = 'michalsky',
7     corr, f, ...)
8 {
9
10 if (missing(lat)) stop('lat missing. You must provide a latitude value.')
11
12 stopifnot(modeRad %in% c('prom', 'aguiar', 'bd', 'bdI'))
13
14 if (missing(corr)){
15   corr = switch(modeRad,
16                 bd = 'CPR',
17                 aguiar = 'CPR',
18                 prom = 'Page',
19                 bdI = 'BRL'
20   )
21 }
22
23 if(is(dataRad, 'Meteo')){BD <- dataRad}
24 else{
25   BD <- switch(modeRad,
26                 bd = {
27                   if (!is.list(dataRad) || !is.data.frame(dataRad)){
28                     dataRad <- list(file = dataRad)
29                   }
30                   switch(class(dataRad$file)[1],
31                         character = {
32                           bd.default = list(file = '', lat = lat)
33                           bd = modifyList(bd.default, dataRad)
34                           res <- do.call('readBDd', bd)
35                           res
36                         },
37                         data.table = ,
38                         data.frame = {
39                           bd.default = list(file = '', lat = lat)
40                           bd = modifyList(bd.default, dataRad)
41                           res <- do.call('dt2Meteo', bd)
42                           res
43                         },
44                         zoo = {
45                           bd.default = list(file = '', lat = lat,
46                                             source = '')
47                           bd = modifyList(bd.default, dataRad)
48                           res <- do.call('zoo2Meteo', bd)
49                           res
50                         })
51               },
52               prom = {
53                 if (!is.list(dataRad)) dataRad <- list(G0dm = dataRad)
54                 prom.default <- list(G0dm = numeric(), lat = lat)
55                 prom = modifyList(prom.default, dataRad)
56                 res <- do.call('readG0dm', prom)
57               },
58               aguiar = {
59                 if (is.list(dataRad)) dataRad <- dataRad$G0dm
60                 BTd <- fBTd(mode = 'serie')
```

```

61     sold <- fSolD(lat, BTd)
62     G0d <- markovG0(dataRad, sold)
63     res <- dt2Meteo(G0d, lat = lat, source = 'aguiar')
64   },
65   bdI = {
66     if (!is.list(dataRad) || is.data.frame(dataRad)){
67       dataRad <- list(file = dataRad)
68     }
69     switch(class(dataRad$file)[1],
70       character = {
71         bdI.default <- list(file = '', lat = lat)
72         bdI <- modifyList(bdI.default, dataRad)
73         res <- do.call('readBDi', bdI)
74         res
75       },
76       data.table = ,
77       data.frame = {
78         bdI.default <- list(file = '', lat = lat)
79         bdI <- modifyList(bdI.default, dataRad)
80         res <- do.call('dt2Meteo', bdI)
81         res
82       },
83       zoo = {
84         bdI.default <- list(file = '', lat = lat,
85                           source = '')
86         bdI <- modifyList(bdI.default, dataRad)
87         res <- do.call('zoo2Meteo', bdI)
88         res
89       },
90       stop('dataRad$file should be a character,
91 a data.table, a data.frame or a zoo.')
92   })
93 )
94 }

95

96 if (modeRad == 'bdI') {
97   sol <- calcSol(lat, sample = sample,
98                 BTi = indexD(BD),
99                 keep.night = keep.night, method = sunGeometry)
100  compI <- fCompI(sol = sol, GOI = BD, corr = corr, f = f, ...)
101  compD <- compI[, lapply(.SD, P2E, sol@sample),
102                .SDcols = c('GO', 'DO', 'BO'),
103                by = truncDay(Dates)]
104  names(compD)[1] <- 'Dates'
105  names(compD)[-1] <- paste(names(compD)[-1], 'd', sep = '')
106  compD$Fd <- compD$D0d/compD$G0d
107  compD$Kt <- compD$G0d/as.data.tableD(sol)$Bo0d
108 } else {
109   sol <- calcSol(lat, indexD(BD), sample = sample,
110                 keep.night = keep.night, method = sunGeometry)
111   compD <- fCompD(sol = sol, G0d = BD, corr = corr, f, ...)
112   compI <- fCompI(sol = sol, compD = compD, ...)
113 }
114

115 Ta = switch(modeRad,
116   bd = {
117     if (all(c("TempMax", "TempMin") %in% names(BD@data))) {

```

B. CÓDIGO COMPLETO

```
119      fTemp(sol, BD)
120    } else {
121      if ("Ta" %in% names(getData(BD))) {
122        data.table(Dates = indexD(sol),
123                    Ta = getData(BD)$Ta)
124      } else {
125        warning('No temperature information available!')
126      }
127    },
128  bdI = {
129    if ("Ta" %in% names(getData(BD))) {
130      data.table(Dates = indexI(sol),
131                  Ta = getData(BD)$Ta)
132    } else {
133      warning('No temperature information available!')
134    }
135  },
136  prom = {
137    if ("Ta" %in% names(getData(BD))) {
138      data.table(Dates = indexD(sol),
139                  Ta = getData(BD)$Ta)
140    } else {
141      warning('No temperature information available!')
142    }
143  },
144  aguiar = {
145    Dates<-indexI(sol)
146    x <- as.Date(Dates)
147    ind.rep <- cumsum(c(1, diff(x) != 0))
148    data.table(Dates = Dates,
149                Ta = getData(BD)$Ta[ind.rep])
150  }
151  )
152)

153 nms <- c('G0d', 'D0d', 'B0d')
154 G0dm <- compD[, lapply(.SD/1000, mean, na.rm = TRUE),
155   .SDcols = nms,
156   by = .(month(Dates), year(Dates))]

157 if(modeRad == 'prom'){
158   G0dm[, DayOfMonth := DOM(G0dm)]
159   G0y <- G0dm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
160   .SDcols = nms,
161   by = .(Dates = year)]
162   G0dm[, DayOfMonth := NULL]
163 } else{
164   G0y <- compD[, lapply(.SD/1000, sum, na.rm = TRUE),
165   .SDcols = nms,
166   by = .(Dates = year(Dates))]
167 }

168 promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18, 13)
169 G0dm[, Dates := as.Date(paste(year, month,
170                           promDays[month], sep = '-'))]
171 G0dm[, c('month', 'year') := NULL]
172 setcolororder(G0dm, 'Dates')

173 result <- new(Class = 'G0',
```

```

177     BD,
178     sol,
179     G0D = compD,
180     G0dm = G0dm,
181     G0y = G0y,
182     G0I = compI,
183     Ta = Ta
184     )
185   return(result)
186 }
```

B.3. calcGef

```

1 calcGef<-function(lat,
2   modeTrk = 'fixed',
3   modeRad = 'prom',
4   dataRad,
5   sample = 'hour',
6   keep.night = TRUE,
7   sunGeometry = 'michalsky',
8   corr, f,
9   betaLim = 90, beta = abs(lat)-10, alpha = 0,
10  iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
11  modeShd = '',
12  struct = list(),
13  distances = data.table(),
14  ...){
15
16  stopifnot(is.list(struct), is.data.frame(distances))
17
18  if (('bt' nilinnil modeShd) & (modeTrk!='horiz')) {
19    modeShd[which(modeShd=='bt')] = 'area'
20    warning('backtracking is only implemented for modeTrk = horiz')}
21
22  if (modeRad != 'prev'){
23    radHoriz <- calcGO(lat = lat, modeRad = modeRad,
24      dataRad = dataRad,
25      sample = sample, keep.night = keep.night,
26      sunGeometry = sunGeometry,
27      corr = corr, f = f, ...)
28  } else {
29    radHoriz <- as(dataRad, 'GO')
30  }
31
32  BT = ("bt" nilinnil modeShd)
33  angGen <- fTheta(radHoriz, beta, alpha, modeTrk,
34    betaLim, BT, struct, distances)
35  inclin <- fInclin(radHoriz, angGen, iS, alb, horizBright, HCPV)
36
37  by <- radHoriz@sample
38  nms <- c('Bo', 'Bn', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
39  nmsd <- paste(nms, 'd', sep = '')
40
41
42  if(radHoriz@type == 'prom'){
43    Gefdm <- inclin[, lapply(.SD/1000, P2E, by),
44      .SDcols = nms,
```

B. CÓDIGO COMPLETO

```
45     by = .(month(Dates), year(Dates))]  
46 names(Gefdm)[-c(1,2)] <- nmsd  
47 GefD <- Gefdm[, .SD*1000,  
48   .SDcols = nmsd,  
49   by = .(Dates = indexD(radHoriz))]  
50  
51 Gefdm[, DayOfMonth := DOM(Gefdm)]  
52 Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),  
53   .SDcols = nmsd,  
54   by = .(Dates = year)]  
55 Gefdm[, DayOfMonth := NULL]  
56 } else{  
57   GefD <- inclin[, lapply(.SD, P2E, by),  
58     .SDcols = nms,  
59     by = .(Dates = truncDay(Dates))]  
60 names(GefD)[-1] <- nmsd  
61  
62 Gefdm <- GefD[, lapply(.SD/1000, mean, na.rm = TRUE),  
63   .SDcols = nmsd,  
64   by = .(month(indexD(radHoriz)), year(indexD(radHoriz)))]  
65 Gefy <- GefD[, lapply(.SD/1000, sum, na.rm = TRUE),  
66   .SDcols = nmsd,  
67   by = .(Dates = year(indexD(radHoriz)))]  
68 }  
69  
70 promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18, 13)  
71 Gefdm[, Dates := as.Date(paste(year, month,  
72                           promDays[month], sep = '-'))]  
73 Gefdm[, c('month', 'year') := NULL]  
74 setcolororder(Gefdm, 'Dates')  
75  
76 result0 = new('Gef',  
77   radHoriz,  
78   Theta = angGen,  
79   GefD = GefD,  
80   Gefdm = Gefdm,  
81   Gefy = Gefy,  
82   GefI = inclin,  
83   iS = iS,  
84   alb = alb,  
85   modeTrk = modeTrk,  
86   modeShd = modeShd,  
87   angGen = list(alpha = alpha, beta = beta, betaLim = betaLim),  
88   struct = struct,  
89   distances = distances  
90 )  
91  
92 if (isTRUE(modeShd == "") ||  
93   ('bt' nilinnil modeShd)) {  
94   return(result0)  
95 } else {  
96   result <- calcShd(result0, modeShd, struct, distances)  
97   return(result)  
98 }  
99 }
```

B.4. prodGCPV

```

1 prodGCPV<-function(lat,
2                         modeTrk = 'fixed',
3                         modeRad = 'prom',
4                         dataRad,
5                         sample = 'hour',
6                         keep.night = TRUE,
7                         sunGeometry = 'michalsky',
8                         corr, f,
9                         betaLim = 90, beta = abs(lat)-10, alpha = 0,
10                        iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
11                        module = list(),
12                        generator = list(),
13                        inverter = list(),
14                        effSys = list(),
15                        modeShd = '',
16                        struct = list(),
17                        distances = data.table(),
18                        ...){
19
20
21  stopifnot(is.list(module),
22            is.list(generator),
23            is.list(inverter),
24            is.list(effSys),
25            is.list(struct),
26            is.data.table(distances))
27
27  if (('bt' nilinnil modeShd) & (modeTrk!='horiz')) {
28    modeShd[which(modeShd=='bt')] = 'area'
29    warning('backtracking is only implemented for modeTrk = horiz')}
30
31  if (modeRad!='prev'){
32
33    radEf <- calcGef(lat = lat, modeTrk = modeTrk, modeRad = modeRad,
34                         dataRad = dataRad,
35                         sample = sample, keep.night = keep.night,
36                         sunGeometry = sunGeometry,
37                         corr = corr, f = f,
38                         betaLim = betaLim, beta = beta, alpha = alpha,
39                         iS = iS, alb = alb, horizBright = horizBright, HCPV = HCPV,
40                         modeShd = modeShd, struct = struct,
41                         distances = distances, ...)
42
43 } else {
44
45   stopifnot(class(dataRad) nilinnil c('GO', 'Gef', 'ProdGCPV'))
46   radEf <- switch(class(dataRad),
47                     GO = calcGef(lat = lat,
48                               modeTrk = modeTrk, modeRad = 'prev',
49                               dataRad = dataRad,
50                               betaLim = betaLim, beta = beta, alpha = alpha,
51                               iS = iS, alb = alb, horizBright = horizBright,
52                               HCPV = HCPV, modeShd = modeShd, struct =
53                               struct,
54                               distances = distances, ...),
55                               Gef = dataRad,
56                               ProdGCPV = as(dataRad, 'Gef')
56 )

```

B. CÓDIGO COMPLETO

```
57 }
58
59
60
61 prodI <- fProd(radEf,module,generator,inverter,effSys)
62 module <- attr(prodI, 'module')
63 generator <- attr(prodI, 'generator')
64 inverter <- attr(prodI, 'inverter')
65 effSys <- attr(prodI, 'effSys')
66
67
68 Pg <- generator$Pg
69
70 by <- radEf@sample
71 nms1 <- c('Pac', 'Pdc')
72 nms2 <- c('Eac', 'Edc', 'Yf')
73
74
75 if(radEf@type == 'prom'){
76   prodDm <- prodI[, lapply(.SD/1000, P2E, by),
77     .SDcols = nms1,
78     by = .(month(Dates), year(Dates))]
79   names(prodDm)[-c(1,2)] <- nms2[-3]
80   prodDm[, Yf := Eac/(Pg/1000)]
81   prodD <- prodDm[, .SD*1000,
82     .SDcols = nms2,
83     by = .(Dates = indexD(radEf))]
84   prodD[, Yf := Yf/1000]
85
86   prodDm[, DayOfMonth := DOM(prodDm)]
87   prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
88     .SDcols = nms2,
89     by = .(Dates = year)]
90   prodDm[, DayOfMonth := NULL]
91 } else {
92   prodD <- prodI[, lapply(.SD, P2E, by),
93     .SDcols = nms1,
94     by = .(Dates = truncDay(Dates))]
95   names(prodD)[-1] <- nms2[-3]
96   prodD[, Yf := Eac/Pg]
97
98   prodDm <- prodD[, lapply(.SD/1000, mean, na.rm = TRUE),
99     .SDcols = nms2,
100     by = .(month(Dates), year(Dates))]
101   prodDm[, Yf := Yf * 1000]
102   prody <- prodD[, lapply(.SD/1000, sum, na.rm = TRUE),
103     .SDcols = nms2,
104     by = .(Dates = year(Dates))]
105   prody[, Yf := Yf * 1000]
106 }
107
108 promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18, 13)
109 prodDm[, Dates := as.Date(paste(year, month,
110                               promDays[month], sep = '-'))]
111 prodDm[, c('month', 'year') := NULL]
112 setcolororder(prodDm, 'Dates')
113
114 result <- new('ProdGCPV',
```

```

115     radEf,
116     prodD = prodD,
117     prodDm = prodDm,
118     prody = prody,
119     prodI = prodI,
120     module = module,
121     generator = generator,
122     inverter = inverter,
123     effSys = effSys
124   )
125 }
```

B.5. prodPVPS

```

1 prodPVPS<-function(lat,
2                         modeTrk = 'fixed',
3                         modeRad = 'prom',
4                         dataRad,
5                         sample = 'hour',
6                         keep.night = TRUE,
7                         sunGeometry = 'michalsky',
8                         corr, f,
9                         betaLim = 90, beta = abs(lat)-10, alpha = 0,
10                        iS = 2, alb = 0.2, horizBright = TRUE, HCPV = FALSE,
11                        pump , H,
12                        Pg, converter = list(),
13                        effSys = list(),
14                        ...){
15
16   stopifnot(is.list(converter),
17             is.list(effSys))
18
19   if (modeRad != 'prev'){
20
21     radEf <- calcGef(lat = lat, modeTrk = modeTrk, modeRad = modeRad,
22                         dataRad = dataRad,
23                         sample = sample, keep.night = keep.night,
24                         sunGeometry = sunGeometry,
25                         corr = corr, f = f,
26                         betaLim = betaLim, beta = beta, alpha = alpha,
27                         iS = iS, alb = alb, horizBright = horizBright, HCPV = HCPV,
28                         modeShd = '', ...)
29
30   } else {
31     stopifnot(class(dataRad) %in% c('GO', 'Gef', 'ProdPVPS'))
32     radEf <- switch(class(dataRad),
33                         GO = calcGef(lat = lat,
34                                     modeTrk = modeTrk, modeRad = 'prev',
35                                     dataRad = dataRad,
36                                     betaLim = betaLim, beta = beta, alpha = alpha,
37                                     iS = iS, alb = alb, horizBright = horizBright,
38                                     HCPV = HCPV, modeShd = '', ...),
39                         Gef = dataRad,
40                         ProdPVPS = as(dataRad, 'Gef')
41                         )
42   }
43 }
```

B. CÓDIGO COMPLETO

```
44 converter.default <- list(Ki = c(0.01,0.025,0.05), Pnom = Pg)
45 converter <- modifyList(converter.default, converter)
46
47 effSys.default <- list(ModQual = 3,ModDisp = 2,OhmDC = 1.5,OhmAC = 1.5,MPP = 1
48 ,TrafoMT = 1,Disp = 0.5)
49 effSys <- modifyList(effSys.default, effSys)
50
51 TONC <- 47
52 Ct <- (TONC-20)/800
53 lambda <- 0.0045
54 Gef <- radEf@GefI$Gef
55 night <- radEf@soli$night
56 Ta <- radEf@Ta$Ta
57
58 Tc <- Ta+Ct*Gef
59 Pdc <- Pg*Gef/1000*(1-lambda*(Tc-25))
60 Pdc[is.na(Pdc)] <- 0
61 PdcN <- with(effSys,
62                 Pdc/converter$Pnom*(1-ModQual/100)*(1-ModDisp/100)*(1-OhmDC/100)
63 )
64 PacN <- with(converter,{
65     A <- Ki[3]
66     B <- Ki[2]+1
67     C <- Ki[1]-(PdcN)
68
69     result <- (-B+sqrt(B^2-4*A*C))/(2*A)
70 })
71 PacN[PacN<0]<-0
72
73 Pac <- with(converter,
74                 PacN*Pnom*(1-effSys$OhmAC/100))
75 Pdc <- PdcN*converter$Pnom*(Pac>0)
76
77
78 fun<-fPump(pump = pump, H = H)
79
80 rango <- with(fun,Pac>=lim[1] & Pac<=lim[2])
81 Pac[!rango]<-0
82 Pdc[!rango]<-0
83 prodI <- data.table(Pac = Pac,Pdc = Pdc,Q = 0,Pb = 0,Ph = 0,f = 0)
84 prodI <- within(prodI,{
85     Q[rango]<-fun$fQ(Pac[rango])
86     Pb[rango]<-fun$fPb(Pac[rango])
87     Ph[rango]<-fun$fPh(Pac[rango])
88     f[rango]<-fun$fFreq(Pac[rango])
89     etam <- Pb/Pac
90     etab <- Ph/Pb
91 })
92
93 prodI[night,]<-NA
94 prodI[, Dates := indexI(radEf)]
95 setcolorder(prodI, c('Dates', names(prodI)[-length(prodI)]))
96
97
98 by <- radEf@sample
99
100
```

```

101 if(radEf@type == 'prom'){
102   prodDm <- prodI[, .(Eac = P2E(Pac, by)/1000,
103                     Qd = P2E(Q, by)),
104                     by = .(month(Dates), year(Dates)))]
105   prodDm[, Yf := Eac/(Pg/1000)]
106
107   prodD <- prodDm[, .(Eac = Eac*1000,
108                     Qd,
109                     Yf),
110                     by = .(Dates = indexD(radEf))]
111
112   prodDm[, DayOfMonth := DOM(prodDm)]
113
114   prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
115                     .SDcols = c('Eac', 'Qd', 'Yf'),
116                     by = .(Dates = year)]
117   prodDm[, DayOfMonth := NULL]
118 } else {
119   prodD <- prodI[, .(Eac = P2E(Pac, by)/1000,
120                     Qd = P2E(Q, by)),
121                     by = .(Dates = truncDay(Dates))]
122   prodD[, Yf := Eac/Pg*1000]
123
124   prodDm <- prodD[, lapply(.SD, mean, na.rm = TRUE),
125                     .SDcols = c('Eac','Qd', 'Yf'),
126                     by = .(month(Dates), year(Dates)))]
127   prody <- prodD[, lapply(.SD, sum, na.rm = TRUE),
128                     .SDcols = c('Eac', 'Qd', 'Yf'),
129                     by = .(Dates = year(Dates))]
130 }
131
132
133 promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18, 13)
134 prodDm[, Dates := as.Date(paste(year, month,
135                               promDays[month], sep = '-'))]
136 prodDm[, c('month', 'year') := NULL]
137 setcolororder(prodDm, 'Dates')
138
139 result <- new('ProdPVPS',
140                 radEf,
141                 prodD = prodD,
142                 prodDm = prodDm,
143                 prody = prody,
144                 prodI = prodI,
145                 pump = pump,
146                 H = H,
147                 Pg = Pg,
148                 converter = converter,
149                 effSys = effSys
150               )
151 }
```

B.6. calcShd

```

1 calcShd<-function(radEf,
2   modeShd = 'prom',
3   struct = list(),
```

B. CÓDIGO COMPLETO

```
4         distances = data.table()
5     )
6 {
7     stopifnot(is.list(struct), is.data.frame(distances))
8
9
10    prom = ("prom" nilinnil modeShd)
11    prev <- as.data.table(radEf, complete = TRUE)
12
13    modeTrk <- radEf@modeTrk
14    sol <- data.table(AzS = prev$AzS,
15                         Als = prev$Als)
16    theta <- radEf@Theta
17    AngGen <- data.table(theta, sol)
18    FS <- fSombra(AngGen, distances, struct, modeTrk, prom)
19
20    gef0 <- radEf@GefI
21    Bef0 <- gef0$Bef
22    Dcef0 <- gef0$Dcef
23    Gef0 <- gef0$Gef
24    Dief0 <- gef0$Dief
25    Ref0 <- gef0$Ref
26
27    Bef <- Bef0*(1-FS)
28    Dcef <- Dcef0*(1-FS)
29    Def <- Dief0+Dcef
30    Gef <- Dief0+Ref0+Bef+Dcef
31
32    nms <- c('Gef', 'Def', 'Dcef', 'Bef')
33    nmsIndex <- which(names(gef0) nilinnil nms)
34    names(gef0)[nmsIndex] <- paste(names(gef0)[nmsIndex], '0', sep = '')
35    GefShd <- gef0
36    GefShd[, c(nms, 'FS')] := .(Gef, Def, Dcef, Bef, FS)
37
38
39    by <- radEf@sample
40    nms <- c('Gef0', 'Def0', 'Bef0', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
41    nmsd <- paste(nms, 'd', sep = '')
42
43    Gefdm <- GefShd[, lapply(.SD/1000, P2E, by),
44                           by = .(month(truncDay(Dates)), year(truncDay(Dates))),
45                           .SDcols = nms]
46    names(Gefdm)[-c(1, 2)] <- nmsd
47
48    if(radEf@type == 'prom'){
49        GefD <- Gefdm[, .SD[, -c(1, 2)] * 1000,
50                           .SDcols = nmsd,
51                           by = .(Dates = indexD(radEf))]
52
53        Gefdm[, DayOfMonth := DOM(Gefdm)]
54
55        Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
56                           .SDcols = nmsd,
57                           by = .(Dates = year)]
58        Gefdm[, DayOfMonth := NULL]
59    } else{
60        GefD <- GefShd[, lapply(.SD/1000, P2E, by),
61                           .SDcols = nms,
```

```

62     by = .(Dates = truncDay(Dates))]
63 names(GefD)[-1] <- nmsd
64
65 Gefy <- GefD[, lapply(.SD[, -1], sum, na.rm = TRUE),
66             .SDcols = nmsd,
67             by = .(Dates = year(Dates))]
68 }
69
70 promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18, 13)
71 Gefdm[, Dates := as.Date(paste(year, month,
72                               promDays[month], sep = '-'))]
73 Gefdm[, c('month', 'year') := NULL]
74 setcolororder(Gefdm, c('Dates', names(Gefdm)[-length(Gefdm)]))
75
76 radEf@modeShd <- modeShd
77 radEf@GefI <- GefShd
78 radEf@GefD <- GefD
79 radEf@Gefdm <- Gefdm
80 radEf@Gefy <- Gefy
81 return(radEf)
82 }
```

B.7. optimShd

```

1 optimShd<-function(lat,
2                         modeTrk = 'fixed',
3                         modeRad = 'prom',
4                         dataRad,
5                         sample = 'hour',
6                         keep.night = TRUE,
7                         sunGeometry = 'michalsky',
8                         betaLim = 90, beta = abs(lat)-10, alpha = 0,
9                         iS = 2, alb = 0.2, HCPV = FALSE,
10                        module = list(),
11                        generator = list(),
12                        inverter = list(),
13                        effSys = list(),
14                        modeShd = '',
15                        struct = list(),
16                        distances = data.table(),
17                        res = 2,
18                        prog = TRUE){
19
20 if (('bt' nilinnil modeShd) & (modeTrk != 'horiz')) {
21   modeShd[which(modeShd == 'bt')] = 'area'
22   warning('backtracking is only implemented for modeTrk = horiz')}
23
24 listArgs <- list(lat = lat, modeTrk = modeTrk, modeRad = modeRad,
25                   dataRad = dataRad,
26                   sample = sample, keep.night = keep.night,
27                   sunGeometry = sunGeometry,
28                   betaLim = betaLim, beta = beta, alpha = alpha,
29                   iS = iS, alb = alb, HCPV = HCPV,
30                   module = module, generator = generator,
31                   inverter = inverter, effSys = effSys,
32                   modeShd = modeShd, struct = struct,
33                   distances = data.table(Lew = NA, Lns = NA, D = NA))
```

B. CÓDIGO COMPLETO

```
34 Red <- switch(modeTrk,
35   horiz = with(distances,
36     data.table(Lew = seq(Lew[1],Lew[2],by = res),
37       H = 0)),
38   two = with(distances,
39     data.table(
40       expand.grid(Lew = seq(Lew[1],Lew[2],by = res),
41         Lns = seq(Lns[1],Lns[2],by = res),
42         H = 0))),
43   fixed = with(distances,
44     data.table(D = seq(D[1],D[2],by = res),
45       H = 0))
46   )
47 )
48
49 casos <- dim(Red)[1]
50
51 if (prog) {pb <- txtProgressBar(min = 0, max = casos+1, style = 3)
52   setTxtProgressBar(pb, 0)}
53
54
55 listArgs0 <- modifyList(listArgs,
56   list(modeShd = '', struct = NULL, distances = NULL) )
57 Prod0 <- do.call(prodGCPV, listArgs0)
58 YfAnual0 <- mean(Prod0@prody$Yf)
59 if (prog) {setTxtProgressBar(pb, 1)}
60
61 YfAnual <- numeric(casos)
62
63 BT <- ('bt' nilinnil modeShd)
64 if (BT) {
65   RadBT <- as(Prod0, 'G0')
66   for (i in seq_len(casos)){
67     listArgsBT <- modifyList(listArgs,
68       list(modeRad = 'prev', dataRad = RadBT,
69         distances = Red[i,]))
70     prod.i <- do.call(prodGCPV, listArgsBT)
71     YfAnual[i] <- mean(prod.i@prody$Yf)
72     if (prog) {setTxtProgressBar(pb, i+1)}
73   }
74 } else {
75   prom <- ('prom' nilinnil modeShd)
76   for (i in seq_len(casos)){
77     Gef0 <- as(Prod0, 'Gef')
78     GefShd <- calcShd(Gef0, modeShd = modeShd,
79       struct = struct, distances = Red[i,])
80     listArgsShd <- modifyList(listArgs,
81       list(modeRad = 'prev', dataRad = GefShd)
82     )
83     prod.i <- do.call(prodGCPV, listArgsShd)
84     YfAnual[i] <- mean(prod.i@prody$Yf)
85     if (prog) {setTxtProgressBar(pb, i+1)}
86   }
87 }
88 if (prog) {close(pb)}
89
90
91 FS <- 1-YfAnual/YfAnual0
```

```

92 GRR <- switch(modeTrk,
93   two = with(Red,Lew*Lns)/with(struct,L*W),
94   fixed = Red$D/struct$L,
95   horiz = Red$Lew/struct$L)
96 SombraDF <- data.table(Red,GRR,FS,Yf = YfAnual)
97 FS.loess <- switch(modeTrk,
98   two = loess(FS~Lew*Lns,data = SombraDF),
99   horiz = loess(FS~Lew,data = SombraDF),
100  fixed = loess(FS~D,data = SombraDF))
101 Yf.loess <- switch(modeTrk,
102  two = loess(Yf~Lew*Lns,data = SombraDF),
103  horiz = loess(Yf~Lew,data = SombraDF),
104  fixed = loess(Yf~D,data = SombraDF))
105 result <- new('Shade',
106   Prod0,
107   FS = FS,
108   GRR = GRR,
109   Yf = YfAnual,
110   FS.loess = FS.loess,
111   Yf.loess = Yf.loess,
112   modeShd = modeShd,
113   struct = struct,
114   distances = Red,
115   res = res
116 )
117 result
118 }

```

B.8. Meteo2Meteo

- Meteo2Meteod

```

1 Meteo2Meteod <- function(G0i)
2 {
3   lat <- G0i@latm
4   source <- G0i@source
5
6   dt0 <- getData(G0i)
7   dt <- dt0[, lapply(.SD, sum, na.rm = TRUE),
8             .SDcols = 'GO',
9             by = .(Dates = as.IDate(Dates))]
10  if('Ta' %in% names(dt0)){
11    Ta <- dt0[, .(Ta = mean(Ta),
12               TempMin = min(Ta),
13               TempMax = max(Ta)),
14               by = .(Dates = as.IDate(Dates))]
15    if(all(Ta$Ta == c(Ta$TempMin, Ta$TempMax)))
16      Ta[, c('TempMin', 'TempMax') := NULL]
17    dt <- merge(dt, Ta)
18  }
19  if('GO' %in% names(dt)){
20    names(dt)[names(dt) == 'GO'] <- 'G0d'
21  }
22  if('DO' %in% names(dt)){
23    names(dt)[names(dt) == 'DO'] <- 'D0d'
24  }

```

B. CÓDIGO COMPLETO

```
25 if('B0' nilinnil names(dt)){
26   names(dt)[names(dt) == 'B0'] <- 'B0d'
27 }
28 G0d <- dt2Meteo(dt, lat, source, type = 'bd')
29 return(G0d)
30 }
```

■ Meteo2Meteom

```
1 Meteo2Meteom <- function(G0d)
2 {
3   lat <- G0d@latm
4   source <- G0d@source
5
6   dt <- getData(G0d)
7   nms <- names(dt)[-1]
8   dt <- dt[, lapply(.SD, mean),
9             .SDcols = nms,
10            by = .(month(Dates), year(Dates))]
11  dt[, Dates := fBTd()]
12  dt <- dt[, c('month', 'year') := NULL]
13
14  setcolororder(dt, 'Dates')
15
16  G0m <- dt2Meteo(dt, lat, source, type = 'prom')
17  return(G0m)
18 }
```

B.9. readBD

■ readBDd

```
1 readBDd <- function(file, lat,
2                     format = "nild/nilm/nily", header = TRUE,
3                     fill = TRUE, dec = '.', sep = ';',
4                     dates.col = 'Dates', ta.col = 'Ta',
5                     g0.col = 'G0', keep.cols = FALSE, ...)
6 {
7   stopifnot(is.character(dates.col) || is.numeric(dates.col))
8   stopifnot(is.character(ta.col) || is.numeric(ta.col))
9   stopifnot(is.character(g0.col) || is.numeric(g0.col))
10
11  bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep, ...)
12
13  if(dates.col == ''){
14    names(bd)[1] <- 'Dates'
15    dates.col <- 'Dates'
16  }
17
18  if(!(dates.col nilinnil names(bd))) stop(paste('The column',
19                                                dates.col,
20                                                'is not in the file'))
21  if(!(g0.col nilinnil names(bd))) stop(paste('The column',
22                                                g0.col,
```

```

23                                     'is not in the file'))
24 if(!(ta.col %in% names(bd))) stop(paste('The column',
25                                         ta.col,
26                                         'is not in the file'))
27
28 Dates <- bd[[dates.col]]
29 bd[, (dates.col) := NULL]
30 bd[, Dates := as.IDate(Dates, format = format)]
31
32 G0 <- bd[[g0.col]]
33 bd[, (g0.col) := NULL]
34 bd[, G0 := as.numeric(G0)]
35
36 Ta <- bd[[ta.col]]
37 bd[, (ta.col) := NULL]
38 bd[, Ta := as.numeric(Ta)]
39
40 names0 <- NULL
41 if(all(c('D0', 'B0') %in% names(bd))){
42   names0 <- c(names0, 'D0', 'B0')
43 }
44
45 names0 <- c(names0, 'Ta')
46
47 if(all(c('TempMin', 'TempMax') %in% names(bd))){
48   names0 <- c(names0, 'TempMin', 'TempMax')
49 }
50 if(keep.cols)
51 {
52   setcolororder(bd, c('Dates', 'G0', names0))
53 }
54 else
55 {
56   cols <- c('Dates', 'G0', names0)
57   bd <- bd[, .cols]
58 }
59
60 setkey(bd, 'Dates')
61 result <- new(Class = 'Meteo',
62                 latm = lat,
63                 data = bd,
64                 type = 'bd',
65                 source = file)
66 }
```

■ readBDi

```

1 readBDi <- function(file, lat,
2                     format = "nild/nslm/nsly nsH:nsM:nsS",
3                     header = TRUE, fill = TRUE, dec = '.',
4                     sep = ';', dates.col = 'Dates', times.col,
5                     ta.col = 'Ta', g0.col = 'G0', keep.cols = FALSE, ...)
6 {
7   stopifnot(is.character(dates.col) || is.numeric(dates.col))
8   stopifnot(is.character(ta.col) || is.numeric(ta.col))
9   stopifnot(is.character(g0.col) || is.numeric(g0.col))
```

B. CÓDIGO COMPLETO

```
10 bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep, ...)
11
12 if(dates.col == ''){
13   names(bd)[1] <- 'Dates'
14   dates.col <- 'Dates'
15 }
16
17
18 if(!(dates.col %in% names(bd))) stop(paste('The column', dates.col, 'is
19   not in the file'))
20 if(!(g0.col %in% names(bd))) stop(paste('The column', g0.col, 'is not in
21   the file'))
22 if(!(ta.col %in% names(bd))) stop(paste('The column', ta.col, 'is not in
23   the file'))
24
25 if(!missing(times.col)){
26   stopifnot(is.character(times.col) || is.numeric(times.col))
27   if(!(times.col %in% names(bd))) stop(paste('The column', times.col, 'is
28     not in the file'))
29
30   format <- strsplit(format, ' ')
31   dd <- as.IDate(bd[[dates.col]], format = format[[1]][1])
32   tt <- as.ITime(bd[[times.col]], format = format[[1]][2])
33   bd[, (dates.col) := NULL]
34   bd[, (times.col) := NULL]
35   bd[, Dates := as.POSIXct(dd, tt, tz = 'UTC')]
36 }
37
38 else
39 {
40   dd <- as.POSIXct(bd[[dates.col]], format = format, tz = 'UTC')
41   bd[, (dates.col) := NULL]
42   bd[, Dates := dd]
43 }
44
45 G0 <- bd[[g0.col]]
46 bd[, (g0.col) := NULL]
47 bd[, G0 := as.numeric(G0)]
48
49 Ta <- bd[[ta.col]]
50 bd[, (ta.col) := NULL]
51 bd[, Ta := as.numeric(Ta)]
52
53 names0 <- NULL
54 if(all(c('DO', 'BO') %in% names(bd))){
55   names0 <- c(names0, 'DO', 'BO')
56 }
57
58 names0 <- c(names0, 'Ta')
59
60 if(keep.cols){
61   setcolorder(bd, c('Dates', 'G0', names0))
62 }
63 else{
64   cols <- c('Dates', 'G0', names0)
65   bd <- bd[, ..cols]
```

```

64 }
65
66 setkey(bd, 'Dates')
67 result <- new(Class = 'Meteo',
68                 latm = lat,
69                 data = bd,
70                 type = 'bdI',
71                 source = file)
72 }
```

■ dt2Meteo

```

1 dt2Meteo <- function(file, lat, source = '', type){
2   if(missing(lat)) stop('lat is missing')
3
4   if(source == '') source <- class(file)[1]
5
6   bd <- data.table(file)
7
8   bd[, Dates := as.POSIXct(Dates, tz = 'UTC')]
9
10  if(missing(type)){
11    sample <- median(diff(bd$Dates))
12    IsDaily <- as.numeric(sample, units = 'days')
13    if(is.na(IsDaily)) IsDaily <- ifelse('G0d' %in% names(bd),
14                                             1, 0)
15    if(IsDaily >= 30) type <- 'prom'
16    else{
17      type <- ifelse(IsDaily >= 1, 'bd', 'bdI')
18    }
19
20  }
21  nms0 <- switch(type,
22                   bd = ,
23                   prom = {
24                     nms0 <- 'G0d'
25                     if(all(c('D0d', 'B0d') %in% names(bd))){
26                       nms0 <- c(nms0, 'D0d', 'B0d')
27                     }
28                     if('Ta' %in% names(bd)) nms0 <- c(nms0, 'Ta')
29                     if(all(c('TempMin', 'TempMax') %in% names(bd))){
30                       nms0 <- c(nms0, 'TempMin', 'TempMax')
31                     }
32                     nms0
33                   },
34                   bdI = {
35                     nms0 <- 'G0'
36                     if(all(c('D0', 'B0') %in% names(bd))){
37                       nms0 <- c(nms0, 'D0', 'B0')
38                     }
39                     if('Ta' %in% names(bd)) nms0 <- c(nms0, 'Ta')
40                     nms0
41                   })
42  setcolorder(bd, c('Dates', nms0))
43  setkey(bd, 'Dates')
44  result <- new(Class = 'Meteo',
```

B. CÓDIGO COMPLETO

```
45     latm = lat,
46     data = bd,
47     type = type,
48     source = source)
49
50 if(!('Ta' %in% names(bd))){
51   if(all(c('TempMin', 'TempMax') %in% names(bd))){
52     sol <- calcSol(lat = lat, BTi = indexD(result))
53     bd[, Ta := fTemp(sol, result)$Ta]
54   }
55   else bd[, Ta := 25]
56   result@data <- bd
57 }
58 return(result)
59 }
```

■ **zoo2Meteo**

```
1 zoo2Meteo <- function(file, lat, source = '')
2 {
3   if(source == ''){
4     name <- deparse(substitute(file))
5     cl <- class(file)
6     source <- paste(cl, name, sep = '-')
7   }
8   bd <- data.table(file)
9   sample <- median(diff(index(file)))
10  IsDaily <- as.numeric(sample, units = 'days')>=1
11  type <- ifelse(IsDaily, 'bd', 'bdI')
12  result <- new(Class = 'Meteo',
13    latm = lat,
14    data = bd,
15    type = type,
16    source = source)
17 }
```

B.10. **readG0dm**

```
1 readG0dm <- function(G0dm, Ta = 25, lat = 0,
2                       year = as.POSIXlt(Sys.Date())$year + 1900,
3                       promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18,
4                       13),
5                       source = '')
6 {
7   if(missing(lat)){lat <- 0}
8   Dates <- as.IDate(paste(year[1], 1:12, promDays, sep = '-'), tz = 'UTC')
9   if (length(year)>1){
10     for (i in year[-1]){
11       x <- as.IDate(paste(i, 1:12, promDays, sep = '-'), tz = 'UTC')
12       Dates <- c(Dates, x)
13     }
14   }
15   G0dm.dt <- data.table(Dates = Dates,
16                         G0d = G0dm,
17                         Ta = Ta)
```

```

17 setkey(G0dm.dt, 'Dates')
18 results <- new(Class = 'Meteo',
19                 latm = lat,
20                 data = G0dm.dt,
21                 type = 'prom',
22                 source = source)
23 }

```

B.11. readSIAR

```

1 readSIAR <- function(Lon = 0, Lat = 0,
2                         inicio = paste(year(Sys.Date())-1, '01-01', sep = '-'),
3                         final = paste(year(Sys.Date())-1, '12-31', sep = '-'),
4                         tipo = 'Mensuales', n_est = 3){
5   inicio <- as.Date(inicio)
6   final <- as.Date(final)
7
8   n_reg <- switch(tipo,
9     Horarios = {
10       tt <- difftime(final, inicio, units = 'days')
11       tt <- (as.numeric(tt)+1)*48
12       tt <- tt*n_est
13       tt
14     },
15     Diarios = {
16       tt <- difftime(final, inicio, units = 'days')
17       tt <- as.numeric(tt)+1
18       tt <- tt*n_est
19       tt
20     },
21     Semanales = {
22       tt <- difftime(final, inicio, units = 'weeks')
23       tt <- as.numeric(tt)
24       tt <- tt*n_est
25       tt
26     },
27     Mensuales = {
28       tt <- difftime(final, inicio, units = 'weeks')
29       tt <- as.numeric(tt)/4.34524
30       tt <- ceiling(tt)
31       tt <- tt*n_est
32       tt
33     })
34   if(n_reg > 100) stop(paste('Number of requested records (', n_reg,
35                           ') exceeds the maximum allowed (100)', sep = ''))
```

36 siar <- est_SIAR[
37 Fecha_Instalacion <= final & (is.na(Fecha_Baja) | Fecha_Baja >= inicio)
38]
39
40 siar[, dist := haversine(Latitud, Longitud, Lat, Lon)]
41 siar <- siar[order(dist)][1:n_est]
42 siar[, peso := 1/dist]
43 siar[, peso := peso/sum(peso)]
44 siar <- siar[, .(Estacion, Código, dist, peso)]
45
46 siar_list <- list()
47 for(código in siar\$Código){

B. CÓDIGO COMPLETO

```
48     siar_list[[codigo]] <- siarGET(id = codigo,
49                               inicio = as.character(inicio),
50                               final = as.character(final),
51                               tipo = tipo)
52   siar_list[[codigo]]$peso <- siar[Codigo == codigo, peso]
53 }
54
55 s_comb <- rbindlist(siar_list, use.names = TRUE, fill = TRUE)
56
57 nms <- names(s_comb)
58 nms <- nms[-c(1, length(nms))]
59
60 res <- s_comb[, lapply(.SD * peso, sum, na.rm = TRUE),
61               .SDcols = nms,
62               by = Dates]
63
64 mainURL <- "https://servicio.mapama.gob.es"
65 Estaciones <- siar[, paste(Estacion, '(', Código, ')', sep = '')]
66 Estaciones <- paste(Estaciones, collapse = ', ')
67 source <- paste(mainURL, '\n -Estaciones:', Estaciones, sep = ' ')
68
69 res <- switch(tipo,
70                 Horarios = {dt2Meteo(res, lat = Lat,
71                                         source = mainURL, type = 'bdI')},
72                 Diarios = {dt2Meteo(res, lat = Lat,
73                                         source = mainURL, type = 'bd')},
74                 Semanales = {res},
75                 Mensuales = {dt2Meteo(res, lat = Lat,
76                                         source = source, type = 'prom')})
77
78 return(res)
79 }
```

B.12. Clase Meteo

```
1 setClass(
2   Class = 'Meteo',
3   slots = c(
4     latm = 'numeric',
5     data = 'data.table',
6     type = 'character',
7     source = 'character'
8   ),
9   validity = function(object) {return(TRUE)}
10 )
```

B.13. Clase Sol

```
1 setClass(
2   Class = 'Sol', ##Solar angles
3   slots = c(
4     lat = 'numeric',      #latitud in degrees, >0 if North
5     solD = 'data.table', #daily angles
6     solI = 'data.table', #intradaily angles
7     sample = 'character', #sample of time
8     method = 'character' #method used for geometry calculations
9   ),
```

```

10   validity = function(object) {return(TRUE)}
11 )

```

B.14. Clase G0

```

1 setClass(
2   Class = 'G0',
3   slots = c(
4     GOD = 'data.table',
5     G0dm = 'data.table',
6     G0y = 'data.table',
7     GOI = 'data.table',
8     Ta = 'data.table'
9   ),
10  contains = c('Sol', 'Meteo'),
11  validity = function(object) {return(TRUE)}
12 )

```

B.15. Clase Gef

```

1 setClass(
2   Class = 'Gef',
3   slots = c(
4     GefD = 'data.table',
5     Gefdm = 'data.table',
6     Gefy = 'data.table',
7     GefI = 'data.table',
8     Theta = 'data.table',
9     iS = 'numeric',
10    alb = 'numeric',
11    modeTrk = 'character',
12    modeShd = 'character',
13    angGen = 'list',
14    struct = 'list',
15    distances = 'data.frame'
16   ),
17   contains = 'GO',
18   validity = function(object) {return(TRUE)}
19 )

```

B.16. Clase ProdGCPV

```

1 setClass(
2   Class = 'ProdGCPV',
3   slots = c(
4     prodD = 'data.table',
5     prodDm = 'data.table',
6     prody = 'data.table',
7     prodI = 'data.table',
8     module = 'list',
9     generator = 'list',
10    inverter = 'list',
11    effSys = 'list'
12   ),
13   contains = 'Gef',

```

B. CÓDIGO COMPLETO

```
14     validity = function(object) {return(TRUE)}  
15 })
```

B.17. Clase ProdPVPS

```
1 setClass(  
2   Class = 'ProdPVPS',  
3   slots = c(  
4     prodD = 'data.table',  
5     prodDm = 'data.table',  
6     prody = 'data.table',  
7     prodI = 'data.table',  
8     Pg = 'numeric',  
9     H = 'numeric',  
10    pump = 'list',  
11    converter = 'list',  
12    effSys = 'list'  
13  ),  
14  contains = 'Gef',  
15  validity = function(object) {return(TRUE)}  
16 )
```

B.18. Clase Shade

```
1 setClass(  
2   Class = 'Shade',  
3   slots = c(  
4     FS = 'numeric',  
5     GRR = 'numeric',  
6     Yf = 'numeric',  
7     FS.loess = 'loess',  
8     Yf.loess = 'loess',  
9     modeShd = 'character',  
10    struct = 'list',  
11    distances = 'data.frame',  
12    res = 'numeric'  
13  ),  
14  contains = 'ProdGCPV',  
15  validity = function(object) {return(TRUE)}  
16 )
```

B.19. corrFdKt

- Ktm

```
1 Ktm <- function(sol, G0dm){  
2   solf <- sol@sold[, .(Dates, Bo0d)]  
3   solf[, c('month', 'year') := .(month(Dates), year(Dates))]  
4   solf[,Bo0m := mean(Bo0d), by = .(month, year)]  
5   G0df <- G0dm@data[, .(Dates, G0d)]  
6   G0df[, c('month', 'year') := .(month(Dates), year(Dates))]  
7   G0df[, G0d := mean(G0d), by = .(month, year)]  
8   Ktm <- G0df$G0d/solf$Bo0m  
9   return(Ktm)
```

```
10 }
```

■ FdKtPage

```
1 FdKtPage <- function(sol, G0dm){
2   Kt <- Ktm(sol, G0dm)
3   Fd = 1-1.13*Kt
4   return(data.table(Fd, Kt))
5 }
```

■ FdKtLJ

```
1 FdKtLJ <- function(sol, G0dm){
2   Kt <- Ktm(sol, G0dm)
3   Fd = (Kt<0.3)*0.595774 +
4     (Kt>=0.3 & Kt<=0.7)*(1.39-4.027*Kt+5.531*Kt^2-3.108*Kt^3) +
5     (Kt>0.7)*0.215246
6   return(data.table(Fd, Kt))
7 }
```

■ Ktd

```
1 Ktd <- function(sol, G0d){
2   Bo0d <- sol@solD$Bo0d
3   G0d <- getG0(G0d)
4   Ktd <- G0d/Bo0d
5   return(Ktd)
6 }
```

■ FdKtCPR

```
1 FdKtCPR <- function(sol, G0d){
2   Kt <- Ktd(sol, G0d)
3   Fd = (0.99*(Kt<=0.17))+(Kt>0.17 & Kt<0.8)*
4     (1.188-2.272*Kt+9.473*Kt^2-21.856*Kt^3+14.648*Kt^4) +
5     (Kt>=0.8)*0.2426688
6   return(data.table(Fd, Kt))
7 }
```

■ FdKtEKDd

```
1 FdKtEKDd <- function(sol, G0d){
2   ws <- sol@solD$ws
3   Kt <- Ktd(sol, G0d)
4
5   WS1 = (abs(ws)<1.4208)
6   Fd = WS1*((Kt<0.715)*(1-0.2727*Kt+2.4495*Kt^2-11.9514*Kt^3+9.3879*Kt^4) +
```

B. CÓDIGO COMPLETO

```
7   (Kt>=0.715)*(0.143))+  
8   !WS1*((Kt<0.722)*(1+0.2832*Kt-2.5557*Kt^2+0.8448*Kt^3)+  
9   (Kt>=0.722)*(0.175))  
10  return(data.table(Fd, Kt))  
11 }
```

■ FdKtCLIMEDd

```
1 FdKtCLIMEDd <- function(sol, G0d){  
2   Kt <- Ktd(sol, G0d)  
3   Fd = (Kt<=0.13)*(0.952)+  
4     (Kt>0.13 & Kt<=0.8)*(0.868+1.335*Kt-5.782*Kt^2+3.721*Kt^3)+  
5     (Kt>0.8)*0.141  
6   return(data.table(Fd, Kt))  
7 }
```

■ Kti

```
1 Kti <- function(sol, G0i){  
2   Bo0 <- sol@solI$Bo0  
3   G0i <- getG0(G0i)  
4   Kti <- G0i/Bo0  
5   return(Kti)  
6 }
```

■ FdKtEKDh

```
1 FdKtEKDh <- function(sol, G0i){  
2   Kt <- Kti(sol, G0i)  
3   Fd = (Kt<=0.22)*(1-0.09*Kt)+  
4     (Kt>0.22 & Kt<=0.8)*(0.9511-0.1604*Kt+4.388*Kt^2-16.638*Kt^3+12.336*Kt^4)+  
5     (Kt>0.8)*0.165  
6   return(data.table(Fd, Kt))  
7 }
```

■ FdKtCLIMEDh

```
1 FdKtCLIMEDh <- function(sol, G0i){  
2   Kt <- Kti(sol, G0i)  
3   Fd = (Kt<=0.21)*(0.995-0.081*Kt)+  
4     (Kt>0.21 & Kt<=0.76)*(0.724+2.738*Kt-8.32*Kt^2+4.967*Kt^3)+  
5     (Kt>0.76)*0.180  
6   return(data.table(Fd, Kt))  
7 }
```

■ FdKtBRL

```

1 FdKtBRL <- function(sol, Goi){
2   Kt <- Kti(sol, Goi)
3   sample <- sol@sample
4   ind <- indexI(sol)
5
6   solI <- as.data.tableI(sol, complete = TRUE)
7   w <- solI$w
8   night <- solI$night
9   Als <- solI$Als
10  Bo0 <- solI$Bo0
11
12  G0d <- data.table(ind,
13    GO = getGO(Goi),
14    Bo0 = Bo0)
15  G0d[, G0d := P2E(GO, sample), by = truncDay(ind)]
16  G0d[, Bo0d := P2E(Bo0, sample), by = truncDay(ind)]
17  ktd <- G0d[, ifelse(night, 0, G0d/Bo0d)]
18
19  pers <- persistence(sol, Kt)
20
21  Fd = (1+exp(-5.38+6.63*Kt+0.006*r2h(w)-
22    0.007*r2d(Als)+1.75*ktd+1.31*pers))^-1
23
24  return(data.table(Fd, Kt))
25 }

```

B.20. fBTd

```

1 fBTd<-function(mode = 'prom',
2   year = as.POSIXlt(Sys.Date())$year+1900,
3   start = paste('01-01-', year, sep = ''),
4   end = paste('31-12-', year, sep = ''),
5   format = 'nild-nilm-nily'){
6 promDays<-c(17,14,15,15,15,10,18,18,18,19,18,13)
7 BTd = switch(mode,
8   serie = {
9     start.<-as.POSIXct(start, format = format, tz = 'UTC')
10    end.<-as.POSIXct(end, format = format, tz = 'UTC')
11    res<-seq(start., end., by = "1 day")
12  },
13  prom = as.POSIXct(paste(year, 1:12, promDays, sep = '-'), tz =
14    'UTC')
15 )
16 BTd
}

```

B.21. fBTi

```

1 fBTi <- function(BTd, sample = 'hour'){
2   BTi <- lapply(BTd, intervalo, sample)
3   BTi <- do.call(c, BTi)
4   return(BTi)
5 }

```

B.22. fCompD

```

1 fCompD <- function(sol, G0d, corr = 'CPR', f)
2 {
3   if(!(corr %in% c('CPR', 'Page', 'LJ', 'EKDd', 'CLIMEDd', 'user', 'none'))){
4     warning('Wrong descriptor of correlation Fd-Ktd. Set CPR.')
5     corr <- 'CPR'
6   }
7   if(class(sol)[1] != 'Sol'){
8     sol <- sol[, calcSol(lat = unique(lat), BTi = Dates)]
9   }
10  if(class(G0d)[1] != 'Meteo'){
11    dt <- copy(data.table(G0d))
12    if(!('Dates' %in% names(dt))){
13      dt[, Dates := indexD(sol)]
14      setcolorder(dt, 'Dates')
15      setkey(dt, 'Dates')
16    }
17    if('lat' %in% names(dt)){
18      latg <- unique(dt$lat)
19      dt[, lat := NULL]
20    }else{latg <- getLat(sol)}
21    G0d <- dt2Meteo(dt, latg)
22  }
23
24  stopifnot(indexD(sol) == indexD(G0d))
25  Bo0d <- sol@sol$Bo0d
26  G0 <- getData(G0d)$G0
27
28  is.na(G0) <- (G0>Bo0d)
29
30  if(corr != 'none'){
31    Fd <- switch(corr,
32                  CPR = FdKtCPR(sol, G0d),
33                  Page = FdKtPage(sol, G0d),
34                  LJ = FdKtLJ(sol, G0d),
35                  EKDd = FdKtEKDd(sol, G0d),
36                  CLIMEDd = FdKtCLIMEDd(sol, G0d),
37                  user = f(sol, G0d))
38
39    Kt <- Fd$Kt
40    Fd <- Fd$Fd
41    D0d <- Fd * G0
42    B0d <- G0 - D0d
43  }
44
45  else {
46    G0 <- getData(G0d)$G0d
47    D0d <- getData(G0d)[['D0d']]
48    B0d <- getData(G0d)[['B0d']]
49    Fd <- D0d/G0
50    Kt <- G0/Bo0d
51  }
52
53  result <- data.table(Dates = indexD(sol), Fd, Kt, G0d = G0, D0d, B0d)
54  setkey(result, 'Dates')
55  result
56 }
```

B.23. fCompl

```

1  fCompl <- function(sol, compD, GOI,
2                      corr = 'none', f,
3                      filterGO = TRUE){
4    if(!(corr %in% c('EKDh', 'CLIMEDh', 'BRL', 'user', 'none'))){
5      warning('Wrong descriptor of correlation Fd-Ktd. Set EKDh.')
6      corr <- 'EKDh'
7    }
8
9    if(class(sol)[1] != 'Sol'){
10      sol <- sol[, calcSol(lat = unique(lat), BTi = Dates)]
11    }
12
13    lat <- sol@lat
14    sample <- sol@sample
15    night <- sol@solI$night
16    Bo0 <- sol@solI$Bo0
17    Dates <- indexI(sol)
18
19    if (missing(GOI)) {
20
21      GOI <- collper(sol, compD)
22      G0 <- GOI$G0
23      B0 <- GOI$B0
24      D0 <- GOI$D0
25
26      Fd <- D0/G0
27      Kt <- G0/Bo0
28
29    } else {
30
31      if(class(GOI)[1] != 'Meteo'){
32        dt <- copy(data.table(GOI))
33        if(!('Dates' %in% names(dt))){
34          if(length(dt) == 1) names(dt) <- 'G0'
35          dt[, Dates := indexI(sol)]
36          setcolorder(dt, 'Dates')
37          setkey(dt, 'Dates')
38        }
39        if('lat' %in% names(GOI)){latg <- unique(GOI$lat)}
40        else{latg <- lat}
41        GOI <- dt2Meteo(dt, latg)
42      }
43
44      if (corr != 'none'){
45        if (filterGO) {
46          G0 <- getGO(GOI)
47          is.na(G0) <- (G0 > Bo0)
48          GOI <- dt2Meteo(data.table(Dates = indexD(GOI),
49                                G0 = G0),
50                                lat = GOI@latm,
51                                source = GOI@source,
52                                type = GOI@type)
53        }
54
55        Fd <- switch(corr,
56                      EKDh = FdKtEKDh(sol, GOI),
57                      CLIMEDh = FdKtCLIMEDh(sol, GOI),

```

B. CÓDIGO COMPLETO

```
58     BRL = FdKtBRL(sol, GOI),  
59     user = f(sol, GOI))  
60  
61     Kt <- Fd$Kt  
62     Fd <- Fd$Fd  
63     D0 <- Fd * G0  
64     B0 <- G0 - D0  
65  
66 } else {  
67     GO <- getGO(GOI)  
68     D0 <- getData(GOI)[['D0']]  
69     B0 <- getData(GOI)[['B0']]  
70     if (isTRUE(filterGO)) is.na(GO) <- is.na(D0) <- is.na(B0) <- (GO > Bo0)  
71  
72     Fd <- D0/G0  
73     Kt <- GO/Bo0  
74 }  
75 }  
76 GO[night] <- D0[night] <- B0[night] <- Kt[night] <- Fd[night] <- 0  
77  
78 result <- data.table(Dates, Fd, Kt, GO, D0, B0)  
79 setkey(result, 'Dates')  
80 result  
81 }
```

B.24. fInclin

```
1 fInclin <- function(compl, angGen, iS = 2, alb = 0.2,  
2                         horizBright = TRUE, HCPV = FALSE){  
3     stopifnot(iS %in% 1:4)  
4     Beta <- angGen$Beta  
5     Alpha <- angGen$Alpha  
6     cosTheta <- angGen$cosTheta  
7  
8     comp <- as.data.tableI(compl, complete = TRUE)  
9     night <- comp$night  
10    B0 <- comp$B0  
11    Bo0 <- comp$Bo0  
12    D0 <- comp$D0  
13    GO <- comp$GO  
14    cosThzS <- comp$cosThzS  
15    is.na(cosThzS) <- night  
16  
17    Suc <- rbind(c(1, 0.17, -0.069),  
18                  c(0.98,.2,-0.054),  
19                  c(0.97,0.21,-0.049),  
20                  c(0.92,0.27,-0.023))  
21    FTb <- (exp(-cosTheta/Suc[iS,2]) -  
22               exp(-1/Suc[iS,2]))/(1 - exp(-1/Suc[iS,2]))  
23    FTd <- exp(-1/Suc[iS,2] *  
24                  (4/(3*pi) * (sin(Beta) +  
25                               (pi - Beta - sin(Beta))/  
26                               (1 + cos(Beta)))) +  
27                  Suc[iS,3] * (sin(Beta) +  
28                               (pi - Beta - sin(Beta))/  
29                               (1 + cos(Beta)))^2))  
30    FTr <- exp(-1/Suc[iS,2] * (4/(3*pi) * (sin(Beta) +
```

```

31                                     (Beta - sin(Beta))/
32                                     (1 - cos(Beta))) +
33 Suc[iS,3] * (sin(Beta) +
34                                     (Beta - sin(Beta))/
35                                     (1 - cos(Beta)))^2))

36
37 B <- B0 * cosTheta/cosThzS * (cosThzS>0.007)
38 k1 <- B0/(Bo0)
39 Di <- D0 * (1-k1) * (1+cos(Beta))/2
40 if (horizBright) Di <- Di * (1+sqrt(B0/G0) * sin(Beta/2)^3)
41 Dc <- D0 * k1 * cosTheta/cosThzS * (cosThzS>0.007)
42 R <- alb * G0 * (1-cos(Beta))/2
43 D <- (Di + Dc)
44 Bo <- Bo0 * cosTheta/cosThzS * (cosThzS>0.007)
45 Bn <- B0/cosThzS
46 G <- B + D + R
47 Ref <- R * Suc[iS,1] * (1-FTr) * (!HCPV)
48 Ref[is.nan(FTr)] <- 0
49 Dief <- Di * Suc[iS,1] * (1 - FTd) * (!HCPV)
50 Dcef <- Dc * Suc[iS,1] * (1 - FTb) * (!HCPV)
51 Def <- Dief + Dcef
52 Bef <- B * Suc[iS,1] * (1 - FTb)
53 Gef <- Bef + Def + Ref
54
55 result <- data.table(Bo, Bn,
56                         G, D, Di, Dc, B, R,
57                         FTb, FTd, FTr,
58                         Dief, Dcef, Gef, Def, Bef, Ref)
59
60 result[night] <- 0
61 result[, Dates := indexI(compI)]
62 result[, .SD, by = Dates]
63 setcolorder(result, c('Dates', names(result)[-length(result)]))
64 result
65 }

```

B.25. fProd

```

1 fProd <- function(inclin,
2                     module = list(),
3                     generator = list(),
4                     inverter = list(),
5                     effSys = list()
6                     )
7 {
8
9   stopifnot(is.list(module),
10            is.list(generator),
11            is.list(inverter),
12            is.list(effSys)
13            )
14   if (class(inclin)[1]=='Gef') {
15     indInclin <- indexI(inclin)
16     gefI <- as.data.tableI(inclin, complete = TRUE)
17     Gef <- gefI$Gef
18     Ta <- gefI$Ta
19   } else {

```

B. CÓDIGO COMPLETO

```
20     Gef <- inclin$Gef
21     Ta <- inclin$Ta
22 }
23
24 module.default <- list(Vocn = 51.91,
25                         Iscn = 14.07,
26                         Vmn = 43.76,
27                         Imn = 13.03,
28                         Ncs = 24,
29                         Ncp = 6,
30                         CoefVT = 0.0049,
31                         TONC = 45)
32 module <- modifyList(module.default, module)
33 ## Make these parameters visible because they will be used often.
34 Ncs <- module$Ncs
35 Ncp <- module$Ncp
36
37 generator.default <- list(Nms = 22,
38                             Nmp = 130)
39 generator <- modifyList(generator.default, generator)
40 generator$Pg <- (module$Vmn * generator$Nms) *
41   (module$Imn * generator$Nmp)
42 Nms <- generator$Nms
43 Nmp <- generator$Nmp
44
45 inverter.default <- list(Ki = c(0.002, 0.005, 0.008),
46                           Pinv = 1.5e6,
47                           Vmin = 822,
48                           Vmax = 1300,
49                           Gumb = 20)
50 inverter <- modifyList(inverter.default, inverter)
51 Pinv <- inverter$Pinv
52
53 effSys.default <- list(ModQual = 3,
54                           ModDisp = 2,
55                           OhmDC = 1.5,
56                           OhmAC = 1.5,
57                           MPP = 1,
58                           TrafoMT = 1,
59                           Disp = 0.5)
60 effSys <- modifyList(effSys.default, effSys)
61
62 vocn <- with(module, Vocn / Ncs)
63 iscn <- with(module, Iscn/ Ncp)
64 vmn <- with(module, Vmn / Ncs)
65 imn <- with(module, Imn / Ncp)
66 vmin <- with(inverter, Vmin / (Ncs * Nms))
67 vmax <- with(inverter, Vmax / (Ncs * Nms))
68
69 cell <- iv(vocn, iscn,
70             vmn, imn,
71             module$TONC, module$CoefVT,
72             Ta, Gef,
73             vmin, vmax)
74
75 Idc <- Nmp * Ncp * cell$idc
76 Isc <- Nmp * Ncp * cell$isc
77 Imp <- Nmp * Ncp * cell$impp
```

```

78 Vdc <- Nms * Ncs * cell$vdc
79 Voc <- Nms * Ncs * cell$voc
80 Vmpp <- Nms * Ncs * cell$vmpp
81
82 PdcN <- with(effSys, (Idc * Vdc) / Pinv *
83           (1 - ModQual / 100) *
84           (1 - ModDisp / 100) *
85           (1 - MPP / 100) *
86           (1 - OhmDC / 100)
87       )
88
89 Ki <- inverter$Ki
90 if (is.matrix(Ki)) {
91   VP <- cbind(Vdc, PdcN)
92   PacN <- apply(VP, 1, solvePac, Ki)
93 } else {
94   A <- Ki[3]
95   B <- Ki[2] + 1
96   C <- Ki[1] - (PdcN)
97   PacN <- (-B + sqrt(B^2 - 4 * A * C))/(2 * A)
98 }
99 EffI <- PacN / PdcN
100 pacNeg <- PacN <= 0
101 PacN[pacNeg] <- PdcN[pacNeg] <- EffI[pacNeg] <- 0
102
103
104 Pac <- with(effSys, PacN * Pinv *
105           (Gef > inverter$Gumb) *
106           (1 - OhmAC / 100) *
107           (1 - TrafOMT / 100) *
108           (1 - Disp / 100))
109 Pdc <- PdcN * Pinv * (Pac > 0)
110
111 resProd <- data.table(Tc = cell$Tc,
112                         Voc, Isc,
113                         Vmpp, Impp,
114                         Vdc, Idc,
115                         Pac, Pdc,
116                         EffI)
117 if (class(inclin)[1] == 'Gef'){
118   result <- resProd[, .SD,
119                     by = .(Dates = indInclin)]
120   attr(result, 'generator') <- generator
121   attr(result, 'module') <- module
122   attr(result, 'inverter') <- inverter
123   attr(result, 'effSys') <- effSys
124   return(result)
125 } else {
126   result <- cbind(inclin, resProd)
127   return(result)
128 }
129 }
```

B.26. fPump

```

1 fPump <- function(pump, H){
2
```

```

3   w1 <- 3000
4   wm <- 2870
5   s <- (w1-wm)/w1
6   fen <- 50
7   fmin <- sqrt(H/pump$a)
8   fmax <- with(pump, (-b*Qmax+sqrt(b^2*Qmax^2-4*a*(c*Qmax^2-H)))/(2*a))
9   fb <- seq(fmin,min(60,fmax),length = 1000)
10  fe <- fb/(1-s)
11
12  Q <- with(pump, (-b*fb-sqrt(b^2*fb^2-4*c*(a*fb^2-H)))/(2*c))
13  Qmin <- 0.1*pump$Qn*fb/50
14  Q <- Q+(Qmin-Q)*(Q<Qmin)
15
16  Ph <- 2.725*Q*H
17
18  Q50 <- 50*Q/fb
19  H50 <- H*(50/fb)^2
20  etab <- with(pump, j*Q50^2+k*Q50+l)
21  Pb50 <- 2.725*H50*Q50/etab
22  Pb <- Pb50*(fb/50)^3
23
24  Pbc <- Pb*50/fe
25  etam <- with(pump, g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
26  Pmc <- Pbc/etam
27  Pm <- Pmc*fe/50
28  Pac <- Pm
29
30  fQ <- splinefun(Pac,Q)
31  fFreq <- splinefun(Pac,fe)
32  fPb <- splinefun(Pac,Pb)
33  fPh <- splinefun(Pac,Ph)
34  lim <- c(min(Pac),max(Pac))
35  result <- list(lim = lim,
36          fQ = fQ,
37          fPb = fPb,
38          fPh = fPh,
39          fFreq = fFreq)
40 }

```

B.27. fSolD

```

1 fSolD <- function(lat, BTd, method = 'michalsky'){
2   if (abs(lat) > 90){
3     lat <- sign(lat) * 90
4     warning(paste('Latitude outside acceptable values. Set to', lat))
5   }
6   sun <- data.table(Dates = unique(as.IDate(BTd)),
7                      lat = lat)
8
9   sun[, decl := declination(Dates, method = method)]
10  sun[, eo := eccentricity(Dates, method = method)]
11  sun[, EoT := eot(Dates)]
12  sun[, ws := sunrise(Dates, lat, method = method,
13                      decl = decl)]
14  sun[, Bo0d := bo0d(Dates, lat, method = method,
15                      decl = decl,
16                      eo = eo,

```

```

17         ws = ws
18     )]
19   setkey(sun, Dates)
20   return(sun)
21 }

```

B.28. fSolI

```

1 fSolI <- function(solD, sample = 'hour', BTi,
2                     EoT = TRUE, keep.night = TRUE, method = 'michalsky')
3 {
4   Bo <- 1367
5
6   if(missing(BTi)){
7     BTd <- solD$Dates
8     BTi <- fBTi(BTd, sample)
9   }
10  sun <- data.table(Dates = as.IDate(BTi),
11                      Times = as.ITime(BTi))
12  sun <- merge(solD, sun, by = 'Dates')
13  sun[, eqtime := EoT]
14  sun[, EoT := NULL]
15
16  sun[, w := sunHour(Dates, BTi, EoT = EoT, method = method, eqtime = eqtime)]
17
18  sun[, night := abs(w) >= abs(ws)]
19
20  sun[, cosThzS := zenith(Dates, lat, BTi,
21                           method = method,
22                           decl = decl,
23                           w = w
24                         )]
25
26  sun[, Als := asin(cosThzS)]
27
28  sun[, AzS := azimuth(Dates, lat, BTi, sample,
29                        method = method,
30                        decl = decl,
31                        w = w,
32                        cosThzS = cosThzS)]
33
34  sun[, Bo0 := Bo * eo * cosThzS]
35  sun[night == TRUE, Bo0 := 0]
36  sun[, decl := NULL]
37  sun[, eo := NULL]
38  sun[, eqtime := NULL]
39  sun[, ws := NULL]
40  sun[, Bo0d := NULL]
41  sun[, Dates := as.POSIXct(Dates, Times, tz = 'UTC')]
42  sun[, Times := NULL]
43
44  if(!keep.night){
45    sun <- sun[night == FALSE]
46  }
47
48  return(sun)
49

```

50 }

B.29. fSombra

```
1 fSombra<-function(angGen, distances, struct, modeTrk='fixed', prom=TRUE){  
2  
3   stopifnot(modeTrk %in% c('two', 'horiz', 'fixed'))  
4   res <- switch(modeTrk,  
5     two = {fSombra6(angGen, distances, struct, prom)},  
6     horiz = {fSombraHoriz(angGen, distances, struct)},  
7     fixed = {fSombraEst(angGen, distances, struct)}  
8   )  
9   return(res)  
10 }
```

■ fSombra6

```
1 fSombra6<-function(angGen, distances, struct, prom=TRUE)  
2 {  
3   stopifnot(is.list(struct),  
4             is.data.frame(distances))  
5   if (dim(distances)[1] == 1){  
6     Red <- distances[, .(Lew = c(-Lew, 0, Lew, -Lew, Lew),  
7                         Lns = c(Lns, Lns, Lns, 0, 0),  
8                         H=H)]  
9   } else {  
10     Red<-distances[1:5,]  
11  
12     SombraGrupo <- matrix(ncol=5,nrow=dim(angGen)[1]) ###VECTORIZE  
13     for (i in 1:5) {SombraGrupo[,i]<-fSombra2X(angGen,Red[i,],struct)}  
14     distrib <- with(struct,c(1,Ncol-2,1,Nrow-1,(Ncol-2)*(Nrow-1),Nrow-1))  
15     vProm <- c(sum(distrib[c(5,6)]),  
16                 sum(distrib[c(4,5,6)]),  
17                 sum(distrib[c(4,5)]),  
18                 sum(distrib[c(2,3,5,6)]),  
19                 sum(distrib[c(1,2,4,5)]))  
20     Nseg <- sum(distrib) ##Total number of followers  
21  
22     if (prom == TRUE){  
23       FS <- rowSums(sweep(SombraGrupo,2,vProm,'*'))/Nseg  
24       FS[FS>1] <- 1  
25     } else {  
26       FS <- rowSums(SombraGrupo)  
27       FS[FS>1] <-1  
28     }  
29     return(FS)  
30 }
```

■ fSombra2x

```
1 fSombra2X<-function(angGen,distances,struct)  
2 {  
3   stopifnot(is.list(struct),is.data.frame(distances))
```

```

4 P <- with(struct,distances/W)
5 b <- with(struct,L/W)
6 AzS <- angGen$AzS
7 Beta <- angGen$Beta
8 Als <- angGen$Als
9
10 d1 <- abs(P$Lew*cos(AzS)-P$Lns*sin(AzS))
11 d2 <- abs(P$Lew*sin(AzS)+P$Lns*cos(AzS))
12 FC <- sin(Als)/sin(Beta+Als)
13 s <- b*cos(Beta)+(b*sin(Beta)+P$H)/tan(Als)
14 FS1 <- 1-d1
15 FS2 <- s-d2
16 SombraCond <- (FS1>0)*(FS2>0)*(P$Lew*AzS>=0)
17 SombraCond[is.na(SombraCond)] <- FALSE
18 FS <- SombraCond*(FS1*FS2*FC)/b
19 FS[FS>1] <- 1
20 return(FS)
21 }
```

■ fSombraHoriz

```

1 fSombraHoriz<-function(angGen, distances, struct)
2 {
3   stopifnot(is.list(struct), is.data.frame(distances))
4   d <- with(struct, distances/L)
5   AzS <- angGen$AzS
6   Als <- angGen$Als
7   Beta <- angGen$Beta
8   lew <- d$Lew
9   Beta0 <- atan(abs(sin(AzS)/tan(Als)))
10  FS <- 1-lew*cos(Beta0)/cos(Beta-Beta0)
11  SombraCond <- (FS>0)
12  FS <- FS*SombraCond
13  FS[FS>1] <- 1
14  return(FS)
15 }
```

■ fSombraEst

```

1 fSombraEst<-function(angGen, distances, struct)
2 {
3   stopifnot(is.list(struct),is.data.frame(distances))
4   dist <- with(struct, distances/L)
5   Alpha <- angGen$Alpha
6   Beta <- angGen$Beta
7   Als <- angGen$Als
8   AzS <- angGen$AzS
9   cosTheta <- angGen$cosTheta
10  h <- dist$H
11  if(is.null(h)) h <- 0
12  d <- dist$D
13  s <- cos(Beta)+cos(Alpha-AzS)*(sin(Beta)+h)/tan(Als)
14  FC <- sin(Als)/sin(Beta+Als)
15  SombraCond <- (s-d>0)
```

B. CÓDIGO COMPLETO

```
16 FS <- (s-d)*SombraCond*FC*(cosTheta>0)
17 FS <- FS*(FS>0)
18 FS[FS>1] <- 1
19 return(FS)
20 }
```

B.30. fTemp

```
1 fTemp<-function(sol, BD)
2 {
3   stopifnot(class(sol) == 'Sol')
4   stopifnot(class(BD) == 'Meteo')
5
6   checkIndexD(indexD(sol), indexD(BD))
7
8   Dates <- indexI(sol)
9   x <- as.Date(Dates)
10  ind.rep <- cumsum(c(1, diff(x) != 0))
11
12  TempMax <- BD@data$TempMax[ind.rep]
13  TempMin <- BD@data$TempMin[ind.rep]
14  ws <- sol@solD$ws[ind.rep]
15  w <- sol@solI$w
16
17  Tm <- (TempMin+TempMax)/2
18  Tr <- (TempMax-TempMin)/2
19
20  wp <- pi/4
21
22  a1 <- pi*12*(ws-w)/(21*pi+12*ws)
23  a2 <- pi*(3*pi-12*w)/(3*pi-12*ws)
24  a3 <- pi*(24*pi+12*(ws-w))/(21*pi+12*ws)
25
26  T1 <- Tm-Tr*cos(a1)
27  T2 <- Tm+Tr*cos(a2)
28  T3 <- Tm-Tr*cos(a3)
29
30  Ta <- T1*(w<=ws)+T2*(w>ws&w<=wp)+T3*(w>wp)
31
32  result <- data.table(Dates, Ta)
33 }
```

B.31. fTheta

```
1 fTheta<-function(sol, beta, alpha = 0, modeTrk = 'fixed', betaLim = 90,
2                         BT = FALSE, struct, dist)
3 {
4   stopifnot(modeTrk %in% c('two', 'horiz', 'fixed'))
5   if (!missing(struct)) {stopifnot(is.list(struct))}
6   if (!missing(dist)) {stopifnot(is.data.frame(dist))}
7
8   betaLim <- d2r(betaLim)
9   lat <- getLat(sol, 'rad')
10  signLat <- ifelse(sign(lat) == 0, 1, sign(lat))
11
12  solI <- as.data.tableI(sol, complete = TRUE, day = TRUE)
```

```

13  A1S <- solI$A1S
14  AzS <- solI$AzS
15  decl <- solI$decl
16  w <- solI$w
17
18  night <- solI$night
19
20  Beta <- switch(modeTrk,
21      two = {Beta2x = pi/2-A1S
22          Beta = Beta2x+(betaLim-Beta2x)*(Beta2x>betaLim)},
23          fixed = rep(d2r(beta), length(w)),
24          horiz = {BetaHoriz0 = atan(abs(sin(AzS)/tan(A1S)))
25              if (BT){lew = dist$Lew/struct$L
26                  Longitud = lew*cos(BetaHoriz0)
27                  Cond = (Longitud>=1)
28                  Longitud[Cond] = 1
29                  BetaHoriz = BetaHoriz0+asin(Longitud)-pi/2
30              } else {
31                  BetaHoriz = BetaHoriz0
32                  rm(BetaHoriz0)}
33                  Beta = ifelse(BetaHoriz>betaLim,betaLim,BetaHoriz)}
34          )
35  is.na(Beta) <- night
36
37  Alpha<-switch(modeTrk,
38      two = AzS,
39          fixed = rep(d2r(alpha), length(w)),
40          horiz=pi/2*sign(AzS))
41  is.na(Alpha) <- night
42
43  cosTheta<-switch(modeTrk,
44      two = cos(Beta-(pi/2-A1S)),
45      horiz = {
46          t1 = sin(decl)*sin(lat)*cos(Beta)
47          t2 = cos(decl)*cos(w)*cos(lat)*cos(Beta)
48          t3 = cos(decl)*abs(sin(w))*sin(Beta)
49          cosTheta = t1+t2+t3
50          rm(t1,t2,t3)
51          cosTheta
52      },
53      fixed = {
54          t1 = sin(decl)*sin(lat)*cos(Beta)
55          t2 = -signLat*sin(decl)*cos(lat)*sin(Beta)*cos(Alpha)
56          t3 = cos(decl)*cos(w)*cos(lat)*cos(Beta)
57          t4 = signLat*cos(decl)*cos(w)*sin(lat)*sin(Beta)*cos(Alpha)
58          t5 = cos(decl)*sin(w)*sin(Alpha)*sin(Beta)
59          cosTheta = t1+t2+t3+t4+t5
60          rm(t1,t2,t3,t4,t5)
61          cosTheta
62      }
63  )
64  is.na(cosTheta) <- night
65  cosTheta = cosTheta*(cosTheta>0)
66
67  result <- data.table(Dates = indexI(sol),
68                         Beta, Alpha, cosTheta)
69  return(result)
70 }
```

B.32. HQCurve

```

1 HQCurve<-function(pump){
2   w1 <- 3000
3   wm <- 2870
4   s <- (w1-wm)/w1
5   fen <- 50
6
7   f <- seq(35, 50, by = 5)
8   Hn <- with(pump, a*50^2+b*50*Qn+c*Qn^2)
9
10  kiso <- Hn/pump$Qn^2
11  Qiso <- with(pump,seq(0.1*Qn,Qmax,l=10))
12  Hiso <- kiso*Qiso^2
13
14  Curva <- expand.grid(fb=f,Q=Qiso)
15
16  Curva <- within(Curva,{
17    fe = fb/(1-s)
18    H = with(pump,a*fb^2+b*fb*Q+c*Q^2)
19
20    is.na(H) <- (H<0)
21    Q50 <- 50*Q/fb
22    H50 <- H*(50/fb)^2
23    etab <- with(pump,j*Q50^2+k*Q50+l)
24    Pb50 <- 2.725*H50*Q50/etab
25    Pb <- Pb50*(fb/50)^3
26
27    Pbc <- Pb*50/fe
28    etam <- with(pump,g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
29    Pmc <- Pbc/etam
30    Pm <- Pmc*fe/50
31
32    etac <- 0.95
33    cab <- 0.05
34    Pdc <- Pm/(etac*(1-cab))
35    rm(etac,cab,Pmc,Pbc,Pb50,Q50,H50)
36  })
37
38  lattice.disp <- ("lattice" nilinnil .packages())
39  latticeExtra.disp <- ("latticeExtra" nilinnil .packages())
40  if (lattice.disp && latticeExtra.disp) {
41    p <- xyplot(H~Q, groups = factor(fb), data = Curva,
42                 type = 'l', par.settings = custom.theme.2(),
43                 panel = function(x, y, groups, ...){
44                   panel.superpose(x, y, groups, ...)
45                   panel.xyplot(Qiso, Hiso, col='black', ...)
46                   panel.text(Qiso[1], Hiso[1], 'ISO', pos = 3)
47                 })
48    p=p+glayer(panel.text(x[1], y[1], group.value, pos=3))
49    print(p)
50    result <- list(result = Curva, plot = p)
51  } else {
52    warning('lattice and/or latticeExtra packages are not available. Thus, the
53    plot could not be created')
54    result <- Curva}
}
```

B.33. local2Solar

```

1 local2Solar <- function(x, lon=NULL){
2   tz=attr(x, 'tzone')
3   if (tz == '' || is.null(tz)) {tz='UTC'}
4   ##Daylight savings time
5   A0 <- 3600*dst(x)
6   A0neg <- (A0<0)
7   if (any(A0neg)) {
8     A0[A0neg] <- 0
9     warning('Some Daylight Savings Time unknown. Set to zero.')
10  }
11  ##Difference between local longitude and time zone longitude LH
12  LH <- lonHH(tz)
13  if (is.null(lon))
14  {deltaL <- 0
15  } else
16  {deltaL <- d2r(lon)-LH
17  }
18  ##Local time corrected to UTC
19  tt <- format(x, tz=tz)
20  result <- as.POSIXct(tt, tz='UTC')-A0+r2sec(deltaL)
21  result
22 }
```

B.34. NmgPVPS

```

1 NmgPVPS <- function(pump, Pg, H, Gd, Ta = 30,
2                       lambda = 0.0045, TONC = 47,
3                       eta = 0.95, Gmax = 1200, t0 = 6, Nm = 6,
4                       title = '^', theme = custom.theme.2()){
5
6   t <- seq(-t0, t0, l = 2*t0*Nm);
7   d <- Gd/(Gmax*2*t0)
8   s <- (d*pi/2-1)/(1-pi/4)
9   G <- Gmax*cos(t/t0*pi/2)*(1+s*(1-cos(t/t0*pi/2)))
10  G[G<0] <- 0
11  G <- G/(sum(G,na.rm = 1)/Nm)*Gd
12  Red <- expand.grid(G = G,Pnom = Pg,H = H,Ta = Ta)
13  Red <- within(Red,{Tcm<-Ta+G*(TONC-20)/800
14    Pdc = Pnom*G/1000*(1-lambda*(Tcm-25))
15    Pac = Pdc*eta})
16
17  res <- data.table(Red,Q = 0)
18
19  for (i in seq_along(H)){
20    fun <- fPump(pump, H[i])
21    Cond <- res$H == H[i]
22    x <- res$Pac[Cond]
23    z <- res$Pdc[Cond]
24    rango <- with(fun, (x >= lim[1] & x <= lim[2]))
25    x[!rango] <- 0
26    z[!rango] <- 0
27    y <- res$Q[Cond]
28    y[rango] <- fun$fQ(x[rango])
29    res$Q[Cond] <- y
30    res$Pac[Cond] <- x
31    res$Pdc[Cond] <- z
32 }
```

B. CÓDIGO COMPLETO

```
32 }
33
34 resumen <- res[, lapply(.SD, function(x)sum(x, na.rm = 1)/Nm),
35   by = .(Pnom, H)]
36 param <- list(pump = pump, Pg = Pg, H = H, Gd = Gd, Ta = Ta,
37   lambda = lambda, TONC = TONC, eta = eta,
38   Gmax = Gmax, t0 = t0, Nm = Nm)
39
40
41 lattice.disp <- ("lattice" nilinnil .packages())
42 latticeExtra.disp <- ("latticeExtra" nilinnil .packages())
43 if (lattice.disp && latticeExtra.disp){
44   tema <- theme
45   tema1 <- modifyList(tema,
46     list(layout.width = list(panel = 1,
47       ylab = 2,
48       axis.left = 1.0,
49       left.padding = 1,
50       ylab.axis.padding = 1,
51       axis.panel = 1)))
52   tema2 <- modifyList(tema, list(layout.width = list(panel = 1,
53     ylab = 2,
54     axis.left = 1.0,
55     left.padding = 1,
56     ylab.axis.padding = 1,
57     axis.panel = 1)))
58   temaT <- modifyList(tema, list(layout.heights = list(panel = c(1, 1))))
59   p1 <- xyplot(Q~Pdc, groups = H, data = resumen,
60     ylab = "Qd (m\u00b3/d)", type = c('l', 'g'),
61     par.settings = tema1)
62
63   p1lab <- p1+glayer(panel.text(x[1], y[1], group.value, pos = 2, cex = 0.7))
64
65   p2 <- xyplot(Pnom~Pdc, groups = H, data = resumen,
66     ylab = "Pg", type = c('l', 'g'),
67     par.settings = tema2)
68   p2lab <- p2+glayer(panel.text(x[1], y[1], group.value, pos = 2, cex = 0.7))
69
70   p <- update(c(p1lab, p2lab, x.same = TRUE),
71     main = paste(title, '\nSP', pump$Qn, 'A', pump$stages, ' ',
72       'Gd ', Gd/1000, " kWh/m\u00b3", sep = ''),
73     layout = c(1, 2),
74     scales = list(x = list(draw = FALSE)),
75     xlab = '',
76     ylab = list(c("Qd (m\u00b3/d)", "Pg (Wp)"), y = c(1/4, 3/4)),
77     par.settings = temaT
78   )
79   print(p)
80   result <- list(I = res, D = resumen, plot = p, param = param)
81 } else {
82   warning('lattice, latticeExtra packages are not all available. Thus, the
83   plot could not be created')
84   result <- list(I = res, D = resumen, param = param)
85 }
```

B.35. sample2Diff

- diff2Hours

```

1 diff2Hours <- function(by){
2   if (!inherits(by, 'difftime')) {
3     stop('This function is only useful for difftime objects.')
4   } else {
5     return(as.numeric(by, units='hours'))
6   }
7 }
```

- char2diff

```

1 char2diff <- function(by){
2   if (!is.character(by)) {
3     stop('This function is only useful for character strings.')
4   } else {
5     ##Adapted from seq.POSIXt
6     by2 <- strsplit(by, " ", fixed = TRUE)[[1L]]
7     if (length(by2) > 2L || length(by2) < 1L)
8       stop("invalid 'by' string")
9     units <- c("secs", "mins", "hours")
10    valid <- pmatch(by2[length(by2)], units)
11    if (is.na(valid)) {
12      stop("invalid string for 'by'")
13    } else {
14      unitValid <- units[valid]
15      if (length(by2)==1) {
16        by2=1
17      } else {
18        by2=as.numeric(by2[1])
19      }
20      result <- as.difftime(by2,units=unitValid)
21      return(result)
22    }
23  }
24 }
```

- sample2Hours

```

1 sample2Hours <- function(by){
2   if (is.character(by)) {
3     y <- char2diff(by)
4     return(diff2Hours(y))
5   } else if (inherits(by, 'difftime')) {
6     return(diff2Hours(by))
7   } else {stop('by must be a character or difftime.')}
8 }
```

- P2E

```
1 P2E <- function(x, by){  
2   Nm <- 1/sample2Hours(by)  
3   sum(x, na.rm = 1)/Nm  
4 }
```

B.36. solarAngles

▪ declination

```
1 declination <- function(d, method = 'michalsky')  
2 {  
3   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){  
4     warning("'method' must be: michalsky, cooper, strous or spencer.  
5 Set michalsky")  
6     method = 'michalsky'  
7   }  
8  
9   d <- as.IDate(d)  
10  dn <- yday(d)  
11  origin <- as.IDate('2000-01-01')  
12  jd <- as.numeric(d - origin)  
13  X <- 2 * pi * (dn - 1) / 365  
14  
15  switch(method,  
16    michalsky = {  
17      meanLong <- (280.460 + 0.9856474 * jd)nilnil360  
18      meanAnomaly <- (357.528 + 0.9856003 * jd)nilnil360  
19      eclipLong <- (meanLong +1.915 * sin(d2r(meanAnomaly)) +  
20                          0.02 * sin(d2r(2 * meanAnomaly)))nilnil360  
21      excen <- 23.439 - 0.0000004 * jd  
22      sinEclip <- sin(d2r(eclipLong))  
23      sinExcen <- sin(d2r(excen))  
24      asin(sinEclip * sinExcen)  
25    },  
26    cooper = {  
27      d2r(23.45) * sin(2 * pi * (dn +284) / 365)  
28    },  
29    strous = {  
30      meanAnomaly <- (357.5291 + 0.98560028 * jd)nilnil360  
31      coefC <- c(1.9148, 0.02, 0.0003)  
32      sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))  
33      C <- colSums(coefC * sinC)  
34      trueAnomaly <- (meanAnomaly + C)nilnil360  
35      eclipLong <- (trueAnomaly + 282.9372)nilnil360  
36      excen <- 23.435  
37      sinEclip <- sin(d2r(eclipLong))  
38      sinExcen <- sin(d2r(excen))  
39      asin(sinEclip * sinExcen)  
40    },  
41    spencer = {  
42      0.006918 - 0.399912 * cos(X) + 0.070257 * sin(X) -  
43      0.006758 * cos(2 * X) + 0.000907 * sin(2 * X) -  
44      0.002697 * cos(3 * X) + 0.001480 * sin(3 * X)  
45    })  
46 }
```

▪ eccentricity

```

1 eccentricity <- function(d, method = 'michalsky')
2 {
3   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
4     warning("'method' must be: michalsky, cooper, strous or spencer. Set
5     michalsky")
6     method = 'michalsky'
7   }
8
9   d <- as.IDate(d)
10  dn <- yday(d)
11  X <- 2 * pi * (dn-1)/365
12
13  switch(method,
14    cooper = 1 + 0.033*cos(2*pi*dn/365),
15    spencer = ,
16    michalsky = ,
17    strous = 1.000110 + 0.034221*cos(X) +
18      0.001280*sin(X) + 0.000719*cos(2*X) +
19      0.000077*sin(2*X)
20  )
}

```

▪ eot

```

1 eot <- function(d)
2 {
3   d <- as.IDate(d)
4   dn <- yday(d)
5   M <- 2 * pi/365.24 * dn
6   EoT <- 229.18 * (-0.0334 * sin(M) +
7     0.04184 * sin(2 * M + 3.5884))
8   EoT <- h2r(EoT/60)
9   return(EoT)
10 }

```

▪ sunrise

```

1 sunrise <- function(d, lat, method = 'michalsky',
2                           decl = declination(d, method))
3 {
4   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
5     warning("'method' must be: michalsky, cooper, strous or spencer.
6     Set michalsky")
6     method = 'michalsky'
7   }
8
9   cosWs <- -tan(d2r(lat)) * tan(decl)
10  ws <- -acos(cosWs)
11  polar <- which(is.nan(ws))
12  ws[polar] <- -pi * (cosWs[polar] < -1) + 0 * (cosWs[polar] > 1)
13  return(ws)
14 }

```

B. CÓDIGO COMPLETO

■ bo0d

```
1 bo0d <- function(d, lat, method = 'michalsky',
2                     decl = declination(d, method = method),
3                     eo = eccentricity(d, method = method),
4                     ws = sunrise(d, lat, method = method))
5 {
6   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
7     warning("'method' must be: michalsky, cooper, strous or spencer.
8 Set michalsky")
9   method = 'michalsky'
10 }
11
12 Bo <- 1367
13 latr <- d2r(lat)
14 Bo0d <- -24/pi * Bo * eo * (ws * sin(latr) * sin(decl) +
15                               cos(latr) * cos(decl) * sin(ws))
16 return(Bo0d)
17 }
```

■ sunHour

```
1 sunHour <- function(d, BTi, sample = 'hour', EoT = TRUE,
2                      method = 'michalsky',
3                      eqtime = eot(d))
4 {
5   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
6     warning("'method' must be: michalsky, cooper, strous or spencer.
7 Set michalsky")
8   method = 'michalsky'
9 }
10
11 if(missing(BTi)){
12   BTi <- fBTi(BTd = d, sample = sample)
13 }else {
14   if (inherits(BTi, 'data.table')) {
15     Times <- as.ITime(BTi$Times)
16     Dates <- as.IDate(BTi$Dates)
17     BTi <- as.POSIXct(Dates, Times, tz = 'UTC')
18   }
19   else {
20     BTi <- as.POSIXct(BTi, tz = 'UTC')
21   }
22 }
23 rep <- cumsum(c(1, diff(as.Date(BTi)) != 0))
24 if(EoT)
25 {
26   EoT <- eqtime
27   if(length(EoT) != length(BTi)){EoT <- EoT[rep]}
28 }else{EoT <- 0}
29
30 jd <- as.numeric(julian(BTi, origin = '2000-01-01 12:00:00 UTC'))
31 T0 <- hms(BTi)
32
33 w=switch(method,
34           cooper = h2r(T0-12)+EoT,
```

```

35     spencer = h2r(T0-12)+EoT,
36     michalsky = {
37       meanLong <- (280.460+0.9856474*jd)nilnil360
38       meanAnomaly <- (357.528+0.9856003*jd)nilnil360
39       eclipLong <- (meanLong +1.915*sin(d2r(meanAnomaly))+
40                         0.02*sin(d2r(2*meanAnomaly)))nilnil360
41       excen <- 23.439-0.0000004*jd
42
43       sinEclip <- sin(d2r(eclipLong))
44       cosEclip <- cos(d2r(eclipLong))
45       cosExcen <- cos(d2r(excen))
46
47       ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))nilnil360
48
49       lmst <- (h2d(6.697375 + 0.0657098242*jd + T0))nilnil360
50       w <- (lmst-ascension)
51       w <- d2r(w + 360*(w < -180) - 360*(w > 180))
52     },
53     strous = {
54       meanAnomaly <- (357.5291 + 0.98560028*jd)nilnil360
55       coefC <- c(1.9148, 0.02, 0.0003)
56       sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))
57       C <- colSums(coefC*sinC)
58       trueAnomaly <- (meanAnomaly + C)nilnil360
59       eclipLong <- (trueAnomaly + 282.9372)nilnil360
60       excen <- 23.435
61
62       sinEclip <- sin(d2r(eclipLong))
63       cosEclip <- cos(d2r(eclipLong))
64       cosExcen <- cos(d2r(excen))
65
66       ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))nilnil360
67
68       lmst <- (280.1600+360.9856235*jd)nilnil360
69       w <- (lmst-ascension)
70       w <- d2r(w + 360*(w< -180) - 360*(w>180))
71     }
72   )
73   return(w)
74 }
```

■ zenith

```

1 zenith <- function(d, lat, BTi, sample = 'hour', method = 'michalsky',
2                           decl = declination(d, method = method),
3                           w = sunHour(d, BTi, sample, method = method))
4 {
5   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
6     warning("'method' must be: michalsky, cooper, strous or spencer.
7 Set michalsky")
8     method = 'michalsky'
9   }
10
11   if(missing(BTi)){BTi <- fBTi(d, sample)}
12   x <- as.Date(BTi)
13   rep <- cumsum(c(1, diff(x) != 0))
```

B. CÓDIGO COMPLETO

```
14 latr <- d2r(lat)
15 if(length(decl) == length(BTi)){decl <- decl}
16 else{decl <- decl[rep]}
17 zenith <- sin(decl) * sin(latr) +
18   cos(decl) * cos(w) * cos(latr)
19 zenith <- ifelse(zenith > 1, 1, zenith)
20 return(zenith)
21 }
```

■ azimuth

```
azimuth <- function(d, lat, BTi, sample = 'hour', method = 'michalsky',
1 decl = declination(d, method = method),
2   w = sunHour(d, BTi, sample, method = method),
3   cosThzS = zenith(d, lat, BTi, sample,
4     method = method,
5     decl = decl,
6     w = w))
7 {
8   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
9     warning("'method' must be: michalsky, cooper, strous or spencer.
10 Set michalsky")
11   method = 'michalsky'
12 }
13
14
15 signLat <- ifelse(sign(lat) == 0, 1, sign(lat))
16 if(missing(BTi)){BTi <- fBTi(d, sample)}
17 x <- as.Date(BTi)
18 rep <- cumsum(c(1, diff(x) != 0))
19 latr <- d2r(lat)
20 if(length(decl) != length(BTi)){decl <- decl[rep]}
21 Als <- asin(cosThzS)
22 cosazimuth <- signLat * (cos(decl) * cos(w) * sin(latr) -
23   cos(latr) * sin(decl)) / cos(Als)
24 cosazimuth <- ifelse(abs(cosazimuth)>1, sign(cosazimuth), cosazimuth)
25 azimuth <- sign(w)*acos(cosazimuth)
26 return(azimuth)
27 }
```

B.37. utils-angle

■ d2r

```
1 d2r <- function(x){x*pi/180}
```

■ r2d

```
1 r2d <- function(x){x*180/pi}
```

■ h2r

```
1 h2r <- function(x){x*pi/12}
```

- **h2d**

```
1 h2d <- function(x){x*180/12}
```

- **r2h**

```
1 r2h <- function(x){x*12/pi}
```

- **d2h**

```
1 d2h <- function(x){x*12/180}
```

- **r2sec**

```
1 r2sec <- function(x){x*12/pi*3600}
```

B.38. utils-time

- **hms**

```
1 hms <- function(x)
2 {
3   hour(x)+minute(x)/60+second(x)/3600
4 }
```

- **doy**

```
1 doy <- function(x){
2   as.numeric(format(x, 'nilj'))
3 }
```

- **dom**

```
1 dom <- function(x){
2   as.numeric(format(x, 'nild'))
3 }
```

- **dst**

```
1 dst <- function(x) #Daylight Savings Time
2 {
3   as.POSIXlt(x)$isdst
4 }
```

■ truncDay

```
1 truncDay <- function(x){as.POSIXct(trunc(x, units='days'))}
```

B.39. as.data.tableD

```
1 setGeneric('as.data.tableD', function(object, complete = FALSE, day = FALSE){
2   standardGeneric('as.data.tableD')})
3
4 setMethod('as.data.tableD',
5   signature = (object = 'Sol'),
6   definition = function(object, complete = FALSE, day = FALSE){
7     sol <- copy(object)
8     sold <- sol@sold
9     data <- sold
10    if(day){
11      ind <- indexD(sol)
12    }
13    return(data)
14  }
15 )
16
17 setMethod('as.data.tableD',
18   signature = (object = 'GO'),
19   definition = function(object, complete = FALSE, day = FALSE){
20     g0 <- copy(object)
21     GOD <- g0@GOD
22     sold <- g0@sold
23     if(complete){
24       data <- data.table(GOD, sold[, Dates := NULL])
25     } else {
26       GOD[, Fd := NULL]
27       GOD[, Kt := NULL]
28       data <- GOD
29     }
30     if(day){
31       ind <- indexD(object)
32       data$day <- doy(ind)
33       data$month <- month(ind)
34       data$year <- year(ind)
35     }
36     return(data)
37   }
38
39 setMethod('as.data.tableD',
40   signature = (object = 'Gef'),
41   definition = function(object, complete = FALSE, day = FALSE){
42     gef <- copy(object)
```

```

43 GefD <- gef@GefD
44 GOD <- gef@GOD
45 sold <- gef@sold
46 if(complete){
47   data <- data.table(GefD,
48                     GOD[, Dates := NULL],
49                     sold[, Dates := NULL])
50 } else {data <- GefD[, c('Dates', 'Gefd',
51                         'Defd', 'Befd')]}

52 if(day){
53   ind <- indexD(object)
54   data$day <- doy(ind)
55   data$month <- month(ind)
56   data$year <- year(ind)
57 }
58 return(data)
59 }
60 )
61
62 setMethod('as.data.tableD',
63           signature = (object = 'ProdGCPV'),
64           definition = function(object, complete = FALSE, day = FALSE){
65             prodgcpv <- copy(object)
66             prodD <- prodgcpv@prodD
67             GefD <- prodgcpv@GefD
68             GOD <- prodgcpv@GOD
69             sold <- prodgcpv@sold
70             if(complete){
71               data <- data.table(prodD,
72                     GefD[, Dates := NULL],
73                     GOD[, Dates := NULL],
74                     sold[, Dates := NULL]
75                   )
76             } else { data <- prodD[, c('Dates', 'Eac',
77                           'Edc', 'Yf')]}

78             if(day){
79               ind <- indexD(object)
80               data$day <- doy(ind)
81               data$month <- month(ind)
82               data$year <- year(ind)
83             }
84             return(data)
85           }
86         )
87
88 setMethod('as.data.tableD',
89           signature = (object = 'ProdPVPS'),
90           definition = function(object, complete = FALSE, day = FALSE){
91             prodpvps <- copy(object)
92             prodD <- prodpvps@prodD
93             GefD <- prodpvps@GefD
94             GOD <- prodpvps@GOD
95             sold <- prodpvps@sold
96             if(complete){
97               data <- data.table(prodD,
98                     GefD[, Dates := NULL],
99                     GOD[, Dates := NULL],
100                    sold[, Dates := NULL]
100

```

```
101          )
102      } else { data <- prodD[, c('Dates', 'Eac',
103                                'Qd', 'Yf')]}
104      if(day){
105        ind <- indexD(object)
106        data$day <- doy(ind)
107        data$month <- month(ind)
108        data$year <- year(ind)
109      }
110      return(data)
111    }
112  )
```

B.40. as.data.tableI

```
1 setGeneric('as.data.tableI',
2   function(object, complete = FALSE, day = FALSE){standardGeneric('
3     as.data.tableI'))}
4
5 setMethod('as.data.tableI',
6   signature = (object = 'Sol'),
7   definition = function(object, complete = FALSE, day = FALSE){
8     sol <- copy(object)
9     Dates <- indexI(sol)
10    x <- truncDay(Dates)
11    ind.rep <- cumsum(c(1, diff(x) != 0))
12    solI <- sol@solI
13    sold <- sol@sold[ind.rep]
14    if(complete){
15      data <- data.table(solI, sold[, Dates := NULL])
16    } else{data <- solI}
17    if(day){
18      ind <- indexI(sol)
19      data$day <- doy(ind)
20      data$month <- month(ind)
21      data$year <- year(ind)
22    }
23    return(data)
24  }
25
26 setMethod('as.data.tableI',
27   signature = (object = 'GO'),
28   definition = function(object, complete = FALSE, day = FALSE){
29     g0 <- copy(object)
30     Dates <- indexI(g0)
31     x <- truncDay(Dates)
32     ind.rep <- cumsum(c(1, diff(x) != 0))
33     GOI <- g0@GOI
34     GOD <- g0@GOD[ind.rep]
35     solI <- as.data.tableI(as(g0, 'Sol'), complete = TRUE)
36     Ta <- g0@Ta
37     if(nrow(Ta) != nrow(GOI)) Ta <- Ta[ind.rep]
38     if(complete){
39       data <- data.table(solI,
40                           GOI[, Dates := NULL],
41                           GOD[, Dates := NULL],
```

```

42                                     Ta[, Dates := NULL])
43     } else{
44         GOI[, Kt := NULL]
45         GOI[, Fd := NULL]
46         data <- GOI
47     }
48     if(day){
49         ind <- indexI(object)
50         data$day <- doy(ind)
51         data$month <- month(ind)
52         data$year <- year(ind)
53     }
54     return(data)
55 }
56 )
57
58 setMethod('as.data.tableI',
59     signature = (object = 'Gef'),
60     definition = function(object, complete = FALSE, day = FALSE){
61         gef <- copy(object)
62         Dates <- indexI(gef)
63         x <- truncDay(Dates)
64         ind.rep <- cumsum(c(1, diff(x) != 0))
65         GefI <- gef@GefI
66         GefD <- gef@GefD[ind.rep]
67         GOI <- as.data.tableI(as(gef, 'GO'), complete = TRUE)
68         if(complete){
69             data <- data.table(GOI,
70                             GefI[, Dates := NULL],
71                             GefD[, Dates := NULL])
72         } else {
73             data <- GefI[, c('Dates', 'Gef',
74                               'Bef', 'Def')]
75         }
76         if(day){
77             ind <- indexI(object)
78             data$day <- doy(ind)
79             data$month <- month(ind)
80             data$year <- year(ind)
81         }
82         return(data)
83     }
84 )
85
86 setMethod('as.data.tableI',
87     signature = (object = 'ProdGCPV'),
88     definition = function(object, complete = FALSE, day = FALSE){
89         prodgcpv <- copy(object)
90         Dates <- indexI(prodgcpv)
91         x <- truncDay(Dates)
92         ind.rep <- cumsum(c(1, diff(x) != 0))
93         prodI <- prodgcpv@prodI
94         prodD <- prodgcpv@prodD[ind.rep]
95         Theta <- prodgcpv@Theta
96         GefI <- as.data.tableI(as(prodgcpv, 'Gef'), complete = TRUE)
97         if(complete){
98             data <- data.table(GefI,
99                             prodI[, Dates := NULL],

```

B. CÓDIGO COMPLETO

```
100                                Theta[, Dates := NULL])
101      } else {
102        data <- prodI[, c('Dates', 'Pac', 'Pdc')]
103      }
104      if(day){
105        ind <- indexI(object)
106        data$day <- doy(ind)
107        data$month <- month(ind)
108        data$year <- year(ind)
109      }
110      return(data)
111    }
112  )
113
114 setMethod('as.data.tableI',
115   signature = (object = 'ProdPVPS'),
116   definition = function(object, complete = FALSE, day = FALSE){
117     prodpvps <- copy(object)
118     Dates <- indexI(prodpvps)
119     x <- truncDay(Dates)
120     ind.rep <- cumsum(c(1, diff(x) != 0))
121     prodI <- prodpvps@prodI
122     prodD <- prodpvps@prodD[ind.rep]
123     Theta <- prodpvps@Theta
124     GefI <- as.data.tableI(as(prodpvps, 'Gef'), complete = TRUE)
125     if(complete){
126       data <- data.table(GefI,
127                           prodI[, Dates := NULL],
128                           prodD[, Dates := NULL],
129                           Theta[, Dates := NULL])
130     } else {
131       data <- prodI[, c('Dates', 'Pac', 'Pdc')]
132     }
133     if(day){
134       ind <- indexI(object)
135       data$day <- doy(ind)
136       data$month <- month(ind)
137       data$year <- year(ind)
138     }
139     return(data)
140   }
141 )
```

B.41. as.data.tableM

```
1 setGeneric('as.data.tableM', function(object, complete = FALSE, day = FALSE){
2   standardGeneric('as.data.tableM')})
3
4 setMethod('as.data.tableM',
5   signature = (object = 'GO'),
6   definition = function(object, complete = FALSE, day = FALSE){
7     g0 <- copy(object)
8     G0dm <- g0@G0dm
9     data <- G0dm
10    if(day){
11      ind <- indexD(object)
12      data$month <- month(ind)
```

```

12     data$year <- year(ind)
13   }
14   return(data)
15 }
16 )
17
18 setMethod('as.data.tableM',
19   signature = (object = 'Gef'),
20   definition = function(object, complete = FALSE, day = FALSE){
21     gef <- copy(object)
22     Gefdm <- gef@Gefdm
23     G0dm <- gef@G0dm
24     if(complete){
25       data <- data.table(Gefdm, G0dm[, Dates := NULL])
26     } else {data <- Gefdm}
27     if(day){
28       ind <- indexD(object)
29       data$month <- month(ind)
30       data$year <- year(ind)
31     }
32     return(data)
33   }
34 )
35
36 setMethod('as.data.tableM',
37   signature = (object = 'ProdGCPV'),
38   definition = function(object, complete = FALSE, day = FALSE){
39     prodgcpv <- copy(object)
40     prodDm <- prodgcpv@prodDm
41     Gefdm <- prodgcpv@Gefdm
42     G0dm <- prodgcpv@G0dm
43     if(complete){
44       data <- data.table(prodDm,
45                           Gefdm[, Dates := NULL],
46                           G0dm[, Dates := NULL])
47     } else {data <- prodDm}
48     if(day){
49       ind <- indexD(object)
50       data$month <- month(ind)
51       data$year <- year(ind)
52     }
53     return(data)
54   }
55 )
56
57 setMethod('as.data.tableM',
58   signature = (object = 'ProdPVPS'),
59   definition = function(object, complete = FALSE, day = FALSE){
60     prodpvps <- copy(object)
61     prodDm <- prodpvps@prodDm
62     Gefdm <- prodpvps@Gefdm
63     G0dm <- prodpvps@G0dm
64     if(complete){
65       data <- data.table(prodDm,
66                           Gefdm[, Dates := NULL],
67                           G0dm[, Dates := NULL])
68     } else {data <- prodDm}
69     if(day){

```

```
70     ind <- indexD(object)
71     data$month <- month(ind)
72     data$year <- year(ind)
73   }
74   return(data)
75 }
76 )
```

B.42. as.data.tableY

```
1 setGeneric('as.data.tableY', function(object, complete = FALSE, day = FALSE){
2   standardGeneric('as.data.tableY')})
3
4 setMethod('as.data.tableY',
5   signature = (object = 'G0'),
6   definition = function(object, complete = FALSE, day = FALSE){
7     g0 <- copy(object)
8     G0y <- g0@G0y
9     data <- G0y
10    if(day){data$year <- data$Dates}
11    return(data)
12  }
13 )
14
15 setMethod('as.data.tableY',
16   signature = (object = 'Gef'),
17   definition = function(object, complete = FALSE, day = FALSE){
18     gef <- copy(object)
19     Gefy <- gef@Gefy
20     G0y <- gef@G0y
21     if(complete){
22       data <- data.table(Gefy, G0y[, Dates := NULL])
23     } else {data <- Gefy}
24     if(day){data[, year := Dates]}
25     return(data)
26   }
27 )
28
29 setMethod('as.data.tableY',
30   signature = (object = 'ProdGCPV'),
31   definition = function(object, complete = FALSE, day = FALSE){
32     prodgcpv <- copy(object)
33     prody <- prodgcpv@prody
34     Gefy <- prodgcpv@Gefy
35     G0y <- prodgcpv@G0y
36     if(complete){
37       data <- data.table(prody,
38                           Gefy[, Dates := NULL],
39                           G0y[, Dates := NULL])
40     } else {data <- prody}
41     if(day){data$year <- data$Dates}
42     return(data)
43   }
44 )
45 setMethod('as.data.tableY',
46   signature = (object = 'ProdPVPS'),
```

```

47     definition = function(object, complete = FALSE, day = FALSE){
48       prodpvps <- copy(object)
49       prody <- prodpvps@prody
50       Gefy <- prodpvps@Gefy
51       G0y <- prodpvps@G0y
52       if(complete){
53         data <- data.table(prody,
54                               Gefy[, Dates := NULL],
55                               G0y[, Dates := NULL])
56       } else {data <- prody}
57       if(day){data$year <- data$Dates}
58       return(data)
59     }
60   )

```

B.43. compare

```

1 setMethod('compare',
2           signature = 'G0',
3           definition = function(...){
4             vars <- c('D0d', 'B0d', 'G0d')
5             res <- compareFunction(..., vars = vars)
6             return(res)
7           }
8         )
9
10 setMethod('compare',
11           signature = 'Gef',
12           definition = function(...){
13             vars <- c('Defd', 'Befd', 'Gefd')
14             res <- compareFunction(..., vars = vars)
15             return(res)
16           }
17         )
18
19 setMethod('compare',
20           signature = 'ProdGCPV',
21           definition = function(...){
22             vars <- c('G0d', 'Gefd', 'Yf')
23             res <- compareFunction(..., vars = vars)
24             return(res)
25           }
26         )

```

B.44. getData

```

1 setGeneric('getData', function(object){standardGeneric('getData')})
2
3 setMethod('getData',
4           signature = (object = 'Meteo'),
5           definition = function(object){
6             result <- object@data
7             return(result)
8           })

```

B.45. getG0

```
1 setGeneric('getG0', function(object){standardGeneric('getG0')})  
2  
3 setMethod('getG0',  
4   signature = (object = 'Meteo'),  
5   definition = function(object){  
6     result <- getData(object)  
7     return(result$G0)  
8   })
```

B.46. getLat

```
1 setGeneric('getLat', function(object, units = 'rad')  
2 {standardGeneric('getLat')})  
3  
4 setMethod('getLat',  
5   signature = (object = 'Sol'),  
6   definition = function(object, units = 'rad'){  
7     stopifnot(units %in% c('deg', 'rad'))  
8     result = switch(units,  
9       rad = d2r(object@lat),  
10      deg = object@lat)  
11      return(result)  
12    })  
13  
14 setMethod('getLat',  
15   signature = (object = 'Meteo'),  
16   definition = function(object, units = 'rad'){  
17     stopifnot(units %in% c('deg', 'rad'))  
18     result = switch(units,  
19       rad = d2r(object@latm),  
20       deg = object@latm)  
21     return(result)  
22   })
```

B.47. indexD

```
1 setGeneric('indexD', function(object){standardGeneric('indexD')})  
2  
3 setMethod('indexD',  
4   signature = (object = 'Sol'),  
5   definition = function(object){as.POSIXct(object@solD$Dates)}  
6 )  
7  
8 setMethod('indexD',  
9   signature = (object = 'Meteo'),  
10  definition = function(object){as.POSIXct(getData(object)$Dates)})
```

B.48. indexI

```
1 setGeneric('indexI', function(object){standardGeneric('indexI')})  
2  
3 setMethod('indexI',  
4   signature = (object = 'Sol'),
```

```

5     definition = function(object){as.POSIXct(object@soli$Dates)
6   })

```

B.49. levelplot

```

1 setGeneric('levelplot')
2
3 setMethod('levelplot',
4   signature = c(x = 'formula', data = 'Meteo'),
5   definition = function(x, data,
6     par.settings = solaR.theme,
7     panel = panel.levelplot.raster, interpolate =
8     TRUE,
9     xscale.components = xscale.solar,
10    yscale.components = yscale.solar,
11    ...){
12      data0 <- getData(data)
13      ind <- data0$Dates
14      data0$day <- doy(ind)
15      data0$month <- month(ind)
16      data0$year <- year(ind)
17      data0$w <- h2r(hms(ind)-12)
18      levelplot(x, data0,
19        par.settings = par.settings,
20        xscale.components = xscale.components,
21        yscale.components = yscale.components,
22        panel = panel, interpolate = interpolate,
23        ...)
24    }
25  )
26
27 setMethod('levelplot',
28   signature = c(x = 'formula', data = 'Sol'),
29   definition = function(x, data,
30     par.settings = solaR.theme,
31     panel = panel.levelplot.raster, interpolate =
32     TRUE,
33     xscale.components = xscale.solar,
34     yscale.components = yscale.solar,
35     ...){
36       data0 <- as.data.tableI(data, complete=TRUE, day=TRUE)
37       ind <- data0$Dates
38       data0$day <- doy(ind)
39       data0$month <- month(ind)
40       data0$year <- year(ind)
41       levelplot(x, data0,
42         par.settings = par.settings,
43         xscale.components = xscale.components,
44         yscale.components = yscale.components,
45         panel = panel, interpolate = interpolate,
46         ...)
47   }
48
49 setMethod('levelplot',
50   signature = c(x = 'formula', data = 'GO'),
51   definition = function(x, data,

```

```
51         par.settings = solaR.theme,
52         panel = panel.levelplot.raster, interpolate =
53         TRUE,
54         xscale.components = xscale.solar,
55         yscale.components = yscale.solar,
56         ...){
57     data0 <- as.data.tableI(data, complete=TRUE, day=TRUE)
58     ind <- data0$Dates
59     data0$day <- doy(ind)
60     data0$month <- month(ind)
61     data0$year <- year(ind)
62     levelplot(x, data0,
63             par.settings = par.settings,
64             xscale.components = xscale.components,
65             yscale.components = yscale.components,
66             panel = panel, interpolate = interpolate,
67             ... )
68 }
```

B.50. Losses

- **compareLosses**

```
1 setGeneric('losses', function(object){standardGeneric('losses')})
2
3 setMethod('losses',
4           signature = (object = 'Gef'),
5           definition=function(object){
6               dat <- as.data.tableY(object, complete = TRUE)
7               isShd <- ('Gef0d' %in% names(dat)) ##is there shadows?
8               if (isShd) {
9                   shd <- with(dat, mean(1-Gefd/Gef0d))
10                  eff <- with(dat, mean(1-Gef0d/Gd))
11              } else {
12                  shd <- 0
13                  eff <- with(dat, mean(1-Gefd/Gd))
14              }
15              result <- data.table(Shadows = shd, AoI = eff)
16              result <- melt(result, measure.vars = names(result),
17                             variable.name = 'id')
18          }
19      )
20
21 setMethod('losses',
22           signature = (object = 'ProdGCPV'),
23           definition = function(object){
24               datY <- as.data.tableY(object, complete = TRUE)
25               module0 <- object@module
26               module0$CoefVT <- 0 ##No losses with temperature
27               Pg <- object@generator$Pg
28               datI <- as.data.tableI(object, complete = TRUE)
29               if (object@type == 'prom'){
30                   YfDC0 <- datI[, P2E(Vmpp*Impp, object@sample),
31                               by = .(month(Dates), year(Dates))]
32                   YfDC0 <- YfDC0[, V1 := V1/Pg*D0M(YfDC0)]
```

```

33     YfDC0 <- sum(YfDC0$V1)
34     YfAC0 <- datI[, P2E(Pdc*EffI, object@sample),
35                     by = .(month(Dates), year(Dates))]
36     YfAC0 <- YfAC0[, V1 := V1/Pg*DOM(YfAC0)]
37     YfAC0 <- sum(YfAC0$V1)
38 } else {
39     YfDC0 <- datI[, P2E(Vmpp*Impp, object@sample),
40                     by = year(Dates)]
41     YfDC0 <- YfDC0[, V1 := V1/Pg]
42     YfDC0 <- sum(YfDC0$V1)
43     YfAC0 <- datI[, P2E(Pdc*EffI, object@sample),
44                     by = year(Dates)]
45     YfAC0 <- YfAC0[, V1 := V1/Pg]
46     YfAC0 <- sum(YfAC0$V1)
47 }
48 gen <- mean(1-YfDC0/datY$Gefd)
49 YfDC <- datY$Edc/Pg*1000
50 DC <- mean(1-YfDC/YfDC0)
51 inv <- mean(1-YfAC0/YfDC)
52 AC <- mean(1-datY$Yf/YfAC0)
53 result0 <- losses(as(object, 'Gef'))
54 result1 <- data.table(Generator = gen,
55                         DC = DC,
56                         Inverter = inv,
57                         AC = AC)
58 result1 <- melt(result1, measure.vars = names(result1),
59                   variable.name = 'id')
60 result <- rbind(result0, result1)
61 result
62 }
63 )
64

```

■ losses

```

1 setGeneric('compareLosses', signature = '...', function(...){standardGeneric('
2   compareLosses')})
3
4 setMethod('compareLosses', 'ProdGCPV',
5           definition=function(...){
6             dots <- list(...)
7             nms0 <- substitute(list(...))
8             if (!is.null(names(nms0))){ ##do.call
9               nms <- names(nms0[-1])
10            } else {
11              nms <- as.character(nms0[-1])
12            }
13            foo <- function(object, label){
14              yY <- losses(object)
15              yY <- cbind(yY, name=label)
16              yY
17            }
18            cdata <- mapply(FUN=foo, dots, nms, SIMPLIFY=FALSE)
19            z <- do.call(rbind, cdata)
20            z$id <- ordered(z$id, levels = c('Shadows', 'AoI', 'Generator',
21                                         'DC', 'Inverter', 'AC'))}

```

```
21     p <- dotplot(id~value*100, groups = name, data = z,
22                     par.settings = solaR.theme, type = 'b',
23                     auto.key = list(corner = c(0.95,0.2), cex = 0.7),
24                     xlab = 'Losses (nil)')
25     print(p)
26     return(z)
27   }
28 }
```

B.51. mergesolaR

```
1 setGeneric('mergesolaR', signature = '...', function(...){standardGeneric('
2   mergesolaR')))
3
4 setMethod('mergesolaR',
5           signature = 'Meteo',
6           definition = function(...){
7             res <- mergeFunction(..., foo = fooMeteo, var = 'GO')
8             res
9           }
10 )
11
12 setMethod('mergesolaR',
13           signature = 'GO',
14           definition = function(...){
15             res <- mergeFunction(..., foo = fooGO, var = 'G0d')
16             res
17           }
18 )
19
20 setMethod('mergesolaR',
21           signature = 'Gef',
22           definition = function(...){
23             res <- mergeFunction(..., foo = fooGO, var = 'Gefd')
24             res
25           }
26 )
27
28 setMethod('mergesolaR',
29           signature = 'ProdGCPV',
30           definition = function(...){
31             res <- mergeFunction(..., foo = fooGO, var = 'Yf')
32             res
33           }
34 )
35
36 setMethod('mergesolaR',
37           signature = 'ProdPVPS',
38           definition = function(...){
39             res <- mergeFunction(..., foo = fooGO, var = 'Yf')
40             res
41           }
42 )
```

B.52. shadeplot

```

1 setGeneric('shadeplot', function(x, ...)standardGeneric('shadeplot'))
2
3 setMethod('shadeplot', signature(x = 'Shade'),
4           function(x,
5                     main = '',
6                     xlab = expression(L[ew]),
7                     ylab = expression(L[ns]),
8                     n = 9, ...){
9             red <- x@distances
10            FS.loess <- x@FS.loess
11            Yf.loess <- x@Yf.loess
12            struct <- x@struct
13            mode <- x@modeTrk
14            if (mode=='two'){
15                Lew <- seq(min(red$Lew),max(red$Lew),length = 100)
16                Lns <- seq(min(red$Lns),max(red$Lns),length = 100)
17                Red <- expand.grid(Lew = Lew,Lns = Lns)
18                FS <- predict(FS.loess,Red)
19                Red$FS <- as.numeric(FS)
20                AreaG <- with(struct,L*W)
21                GRR <- Red$Lew*Red$Lns/AreaG
22                Red$GRR <- GRR
23                FS.m <- matrix(1-FS,
24                               nrow = length(Lew),
25                               ncol = length(Lns))
26                GRR.m <- matrix(GRR,
27                               nrow = length(Lew),
28                               ncol = length(Lns))
29                niveles <- signif(seq(min(FS.m),max(FS.m),1 = n+1),3)
30                pruebaCB <- ("RColorBrewer" nillinnil .packages())
31                if (pruebaCB) {
32                    paleta <- rev(brewer.pal(n, 'YlOrRd'))
33                } else {
34                    paleta <- rev(heat.colors(n))}
35                par(mar = c(4.1,4.1,2.1,2.1))
36                filled.contour(x = Lew,y = Lns,z = FS.m,
37                               col = paleta,
38                               nlevels = n,
39                               plot.title = title(xlab = xlab,
40                                                 ylab = ylab, main = main),
41                               plot.axes = {
42                                   axis(1); axis(2);
43                                   contour(Lew, Lns, FS.m,
44                                         nlevels = n,
45                                         col = "black",
46                                         labcex = .8, add = TRUE)
47                                   contour(Lew, Lns, GRR.m,
48                                         col = "black", lty = 3,
49                                         labcex = .8, add = TRUE)
50                                   grid(col = "white",lty = 3)},
51                               key.title = title("1-FS",cex.main = .8))
52                }
53                if (mode=='horiz') {
54                    Lew <- seq(min(red$Lew),max(red$Lew),length = 100)
55                    FS <- predict(FS.loess,Lew)
56                    GRR <- Lew/struct$L
57                    plot(GRR,1-FS,main = main,type = 'l',...)
58                }
59            }
60        }
61    }
62}

```

```
58     grid()    }
59   if (mode=='fixed'){
60     D <- seq(min(red$D),max(red$D),length = 100)
61     FS <- predict(FS.loess,D)
62     GRR <- D/struct$L
63     plot(GRR,1-FS,main = main,type = 'l',...)
64     grid()    }
65   )
66 )
```

B.53. window

```
1 setMethod('[',
2   signature='Meteo',
3   definition=function(x, i, j,...){
4     if (!missing(i)) {
5       i <- truncDay(i)
6     } else {
7       i <- indexD(x)[1]
8     }
9     if (!missing(j)) {
10       j <- truncDay(j)+86400-1
11     } else {
12       nDays <- length(indexD(x))
13       j <- indexD(x)[nDays]+86400-1
14     }
15     stopifnot(j>i)
16     if (!is.null(i)) i <- truncDay(i)
17     if (!is.null(j)) j <- truncDay(j)+86400-1
18     d <- indexD(x)
19     x@data <- x@data[(d >= i & d <= j)]
20     x
21   }
22 )
23
24
25 setMethod('[',
26   signature='Sol',
27   definition=function(x, i, j, ...){
28     if (!missing(i)) {
29       i <- truncDay(i)
30     } else {
31       i <- indexD(x)[1]
32     }
33     if (!missing(j)) {
34       j <- truncDay(j)+86400-1
35     } else {
36       nDays <- length(indexD(x))
37       j <- indexD(x)[nDays]+86400-1
38     }
39     stopifnot(j>i)
40     if(!is.null(i)) i <- truncDay(i)
41     if(!is.null(j)) j <- truncDay(j)
42     d1 <- indexD(x)
43     d2 <- indexI(x)
44     x@sold <- x@sold[(d1 >= i & d1 <= j)]
45     x@solI <- x@solI[(d2 >= i & d2 <= j)]
```

```

46         x
47     }
48 )
49
50 setMethod('[[',
51   signature='GO',
52   definition=function(x, i, j, ...){
53     sol <- as(x, 'Sol')[i=i, j=j, ...]
54     meteo <- as(x, 'Meteo')[i=i, j=j, ...]
55     i <- indexI(sol)[1]
56     j <- indexI(sol)[length(indexI(sol))]
57     d1 <- indexD(x)
58     d2 <- indexI(x)
59     GOIw <- x@GOI[(d2 >= i & d2 <= j)]
60     Taw <- x@Ta[(d2 >= i & d2 <= j)]
61     G0dw <- x@GOD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
62     G0dmw <- G0dw[, lapply(.SD/1000, mean, na.rm= TRUE),
63       .SDcols = c('G0d', 'D0d', 'B0d'),
64       by = .(month(Dates), year(Dates)) ]
65     if (x@type == 'prom'){
66       G0dmw[, DayOfMonth := DOM(G0dmw)]
67       G0yw <- G0dmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
68                     .SDcols = c('G0d', 'D0d', 'B0d'),
69                     by = .(Dates = year)]
70       G0dmw[, DayOfMonth := NULL]
71     } else {
72       G0yw <- G0dw[, lapply(.SD/1000, sum, na.rm = TRUE),
73                     .SDcols = c('G0d', 'D0d', 'B0d'),
74                     by = .(Dates = year(unique(truncDay(Dates))))]
75     }
76     G0dmw[, Dates := paste(month.abb[month], year, sep = ' . ')]
77     G0dmw[, c('month', 'year') := NULL]
78     setcolorder(G0dmw, 'Dates')
79     result <- new('GO',
80                   meteo,
81                   sol,
82                   GOD = G0dw,
83                   G0dm = G0dmw,
84                   G0y = G0yw,
85                   GOI = GOIw,
86                   Ta = Taw)
87     result
88   }
89 )
90
91
92 setMethod('[[',
93   signature = 'Gef',
94   definition = function(x, i, j, ...){
95     g0 <- as(x, 'GO')[i = i, j = j, ...]
96     i <- indexI(g0)[1]
97     j <- indexI(g0)[length(indexI(g0))]
98     d1 <- indexD(x)
99     d2 <- indexI(x)
100    GefIw <- x@GefI[(d2 >= i & d2 <= j)]
101    Thetaw <- x@Theta[(d2 >= i & d2 <= j)]
102    Gefdw <- x@GefD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
103    nms <- c('B0d', 'B0nd', 'G0d', 'D0d',

```

B. CÓDIGO COMPLETO

```
104          'Bd', 'Gefd', 'Defd', 'Befd')
105 Gefdmw <- Gefdw[, lapply(.SD/1000, mean, na.rm = TRUE),
106             .SDcols = nms,
107             by = .(month(Dates), year(Dates))]
108 if (x@type == 'prom'){
109   Gefdmw[, DayOfMonth:= DOM(Gefdmw)]
110   Gefyw <- Gefdmw[, lapply(.SD*DayOfMonth, sum),
111                 .SDcols = nms,
112                 by = .(Dates = year)]
113   Gefdmw[, DayOfMonth := NULL]
114 } else {
115   Gefyw <- Gefdw[, lapply(.SD/1000, sum, na.rm = TRUE),
116                 .SDcols = nms,
117                 by = .(Dates = year(Dates))]
118 }
119 Gefdmw[, Dates := paste(month.abb[month], year, sep = '. ')]
120 Gefdmw[, c('month', 'year') := NULL]
121 setcolorder(Gefdmw, 'Dates')
122 result <- new('Gef',
123                 g0,
124                 GefD = Gefdw,
125                 Gefdm = Gefdmw,
126                 Gefy = Gefyw,
127                 GefI = GefIw,
128                 Theta = Thetaw,
129                 iS = x@iS,
130                 alb = x@alb,
131                 modeTrk = x@modeTrk,
132                 modeShd = x@modeShd,
133                 angGen = x@angGen,
134                 struct = x@struct,
135                 distances = x@distances
136               )
137   result
138 }
139 )
140
141
142 setMethod('[',
143   signature = 'ProdGCPV',
144   definition = function(x, i, j, ...){
145     gef <- as(x, 'Gef')[i = i, j = j, ...]
146     i <- indexI(gef)[1]
147     j <- indexI(gef)[length(indexI(gef))]
148     d1 <- indexD(x)
149     d2 <- indexI(x)
150     prodIw <- x@prodI[(d2 >= i & d2 <= j)]
151     prodDw <- x@prodD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
152     prodDmw <- prodDw[, lapply(.SD/1000, mean, na.rm = TRUE),
153                   .SDcols = c('Eac', 'Edc'),
154                   by = .(month(Dates), year(Dates))]
155     prodDmw$Yf <- prodDw$Yf
156     if (x@type == 'prom'){
157       prodDmw[, DayOfMonth := DOM(prodDmw)]
158       prodyw <- prodDmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
159                     .SDcols = c('Eac', 'Edc', 'Yf'),
160                     by = .(Dates = year)]
161       prodDmw[, DayOfMonth := NULL]
```

```

162     } else {
163         prodyw <- prodDw[, lapply(.SD/1000, sum, na.rm = TRUE),
164                         .SDcols = c('Eac', 'Edc', 'Yf'),
165                         by = .(Dates = year(Dates))]
166     }
167     prodDmw[, Dates := paste(month.abb[month], year, sep = '.')]
168     prodDmw[, c('month', 'year') := NULL]
169     setcolororder(prodDmw, c('Dates', names(prodDmw)[-length(prodDmw)]))
170     result <- new('ProdGCPV',
171                   gef,
172                   prodD = prodDw,
173                   prodDm = prodDmw,
174                   prody = prodyw,
175                   prodI = prodIw,
176                   module = x@module,
177                   generator = x@generator,
178                   inverter = x@inverter,
179                   effSys = x@effSys
180                   )
181     result
182   }
183 )
184
185 setMethod('[',
186   signature = 'ProdPVPS',
187   definition = function(x, i, j, ...){
188     gef <- as(x, 'Gef')[i = i, j = j, ...]
189     i <- indexI(gef)[1]
190     j <- indexI(gef)[length(indexI(gef))]
191     d1 <- indexD(x)
192     d2 <- indexI(x)
193     prodIw <- x@prodI[(d2 >= i & d2 <= j)]
194     prodDw <- x@prodD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
195     prodDmw <- prodDw[, .(Eac = Eac/1000,
196                           Qd = Qd,
197                           Yf = Yf),
198                           by = .(month(Dates), year(Dates))]
199     if (x@type == 'prom'){
200       prodDmw[, DayOfMonth := DOM(prodDmw)]
201       prodyw <- prodDmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
202                           .SDcols = c('Eac', 'Qd', 'Yf'),
203                           by = .(Dates = year)]
204       prodDmw[, DayOfMonth := NULL]
205     } else {
206       prodyw <- prodDw[, .(Eac = sum(Eac, na.rm = TRUE)/1000,
207                             Qd = sum(Qd, na.rm = TRUE),
208                             Yf = sum(Yf, na.rm = TRUE)),
209                             by = .(Dates = year(Dates))]
210     }
211     prodDmw[, Dates := paste(month.abb[month], year, sep = '.')]
212     prodDmw[, c('month', 'year') := NULL]
213     setcolororder(prodDmw, c('Dates', names(prodDmw)[-length(prodDmw)]))
214     result <- new('ProdPVPS',
215                   gef,
216                   prodD = prodDw,
217                   prodDm = prodDmw,
218                   prody = prodyw,
219                   prodI = prodIw,

```

```
220     pump = x@pump,
221     H = x@H,
222     Pg = x@Pg,
223     converter = x@converter,
224     effSys = x@effSys
225   )
226   result
227 }
228 )
```

B.54. writeSolar

```
1 setGeneric('writeSolar', function(object, file,
2                           complete = FALSE, day = FALSE,
3                           timeScales = c('i', 'd', 'm', 'y'), sep = ',',
4                           ...){
5   standardGeneric('writeSolar')})
6
7 setMethod('writeSolar', signature = (object = 'Sol'),
8           definition = function(object, file, complete = FALSE,
9                                 day = FALSE,
10                                timeScales = c('i', 'd', 'm', 'y'),
11                                sep = ',', ...){
12   name <- strsplit(file, '\\.')[[1]][1]
13   ext <- strsplit(file, '\\.')[[1]][2]
14   timeScales <- match.arg(timeScales, several.ok = TRUE)
15   if ('i' %in% timeScales) {
16     zI <- as.data.tableI(object, complete = complete, day = day)
17     write.table(zI,
18                 file = file, sep = sep, row.names = FALSE, ...)
19   }
20   if ('d' %in% timeScales) {
21     zD <- as.data.tableD(object, complete = complete, day = day)
22     write.table(zD,
23                 file = paste(name, 'D', ext, sep = '.'), 
24                 sep = sep, row.names = FALSE, ...)
25   }
26   if ('m' %in% timeScales) {
27    zM <- as.data.tableM(object, complete = complete, day = day)
28     write.table(zM,
29                 file = paste(name, 'M', ext, sep = '.'), 
30                 sep = sep, row.names = FALSE, ...)
31   }
32   if ('y' %in% timeScales) {
33     zY <- as.data.tableY(object, complete = complete, day = day)
34     write.table(zY,
35                 file = paste(name, 'Y', ext, sep = '.'), 
36                 sep = sep, row.names = FALSE, ...)
37   }
38 })
```

B.55. xyplot

```
1 setGeneric('xyplot')
2
3 setMethod('xyplot',
```

```

4   signature = c(x = 'data.table', data = 'missing'),
5   definition = function(x, data,
6     par.settings = solaR.theme.2,
7     xscale.components = xscale.solar,
8     yscale.components = yscale.solar,
9     scales = list(y = 'free'),
10    ...){
11
12    N <- length(x)-1
13    x0 <- x[, lapply(.SD, as.numeric), by = Dates]
14    x0 <- melt(x0, id.vars = 'Dates')
15    x0$variable <- factor(x0$variable,
16      levels = rev(levels(factor(x0$variable))))
17    xyplot(value ~ Dates | variable, x0,
18      par.settings = par.settings,
19      xscale.components = xscale.components,
20      yscale.components = yscale.components,
21      scales = scales,
22      type = 'l', layout = c(1,N),
23      ...)
24  })
25
26 setMethod('xyplot',
27   signature = c(x = 'formula', data = 'Meteo'),
28   definition = function(x, data,
29     par.settings = solaR.theme,
30     xscale.components = xscale.solar,
31     yscale.components = yscale.solar,
32     ...){
33
34   data0 <- getData(data)
35   xyplot(x, data0,
36     par.settings = par.settings,
37     xscale.components = xscale.components,
38     yscale.components = yscale.components,
39     strip = strip.custom(strip.levels = c(TRUE, TRUE)), ...)
40 }
41
42 setMethod('xyplot',
43   signature = c(x = 'formula', data = 'Sol'),
44   definition = function(x, data,
45     par.settings = solaR.theme,
46     xscale.components = xscale.solar,
47     yscale.components = yscale.solar,
48     ...){
49
50   data0 <- as.data.tableI(data, complete = TRUE, day = TRUE)
51   data0[, w := h2r(hms(Dates)-12)]
52   xyplot(x, data0,
53     par.settings = par.settings,
54     xscale.components = xscale.components,
55     yscale.components = yscale.components,
56     strip = strip.custom(strip.levels = c(TRUE, TRUE)), ...)
57 }
58
59 setMethod('xyplot',
60   signature = c(x = 'formula', data = 'GO'),
61   definition = function(x, data,
62     par.settings = solaR.theme,

```

```
62         xscale.components = xscale.solar,
63         yscale.components = yscale.solar,
64         ...){
65     data0 <- as.data.tableI(data, complete = TRUE, day = TRUE)
66     xyplot(x, data0,
67             par.settings = par.settings,
68             xscale.components = xscale.components,
69             yscale.components = yscale.components,
70             strip = strip.custom(strip.levels = c(TRUE, TRUE)), ...)
71 }
72 )
73
74 setMethod('xyplot',
75           signature = c(x = 'formula', data = 'Shade'),
76           definition = function(x, data,
77                                 par.settings = solaR.theme,
78                                 xscale.components = xscale.solar,
79                                 yscale.components = yscale.solar,
80                                 ...){
81     data0 <- as.data.table(data)
82     xyplot(x, data0,
83             par.settings = par.settings,
84             xscale.components = xscale.components,
85             yscale.components = yscale.components,
86             strip = strip.custom(strip.levels = c(TRUE, TRUE)), ...)
87 }
88 )
89
90 setMethod('xyplot',
91           signature = c(x = 'Meteo', data = 'missing'),
92           definition = function(x, data,
93                                 ...){
94     x0 <- getData(x)
95     xyplot(x0,
96             scales = list(cex = 0.6, rot = 0, y = 'free'),
97             strip = FALSE, strip.left = TRUE,
98             par.strip.text = list(cex = 0.6),
99             ylab = '',
100            ...)
101 }
102 )
103
104 setMethod('xyplot',
105           signature = c(x = 'GO', data = 'missing'),
106           definition = function(x, data, ...){
107     x0 <- as.data.tableD(x, complete = FALSE)
108     x0 <- melt(x0, id.vars = 'Dates')
109     xyplot(value~Dates, x0, groups = variable,
110             par.settings = solaR.theme.2,
111             xscale.components = xscale.solar,
112             yscale.components = yscale.solar,
113             superpose = TRUE,
114             auto.key = list(space = 'right'),
115             ylab = 'Wh/m\u00b2',
116             type = 'l',
117             ...)
118 }
119 )
```

```
120
121 setMethod('xyplot',
122     signature = c(x = 'ProdGCPV', data = 'missing'),
123     definition = function(x, data, ...){
124         x0 <- as.data.tableD(x, complete = FALSE)
125         xyplot(x0,
126                 strip = FALSE, strip.left = TRUE,
127                 ylab = ' ', ...)
128     }
129 )
130
131 setMethod('xyplot',
132     signature = c(x = 'ProdPVPS', data = 'missing'),
133     definition = function(x, data, ...){
134         x0 <- as.data.tableD(x, complete = FALSE)
135         xyplot(x0,
136                 strip = FALSE, strip.left = TRUE,
137                 ylab = ' ', ...)
138     }
139 )
```