



UNIVERSIDAD  
POLITÉCNICA  
DE MADRID



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

Grado en Ingeniería Eléctrica

---

TRABAJO DE FIN DE GRADO

**Título**

Autor: Francisco Delgado López

---

Tutor: Oscar Perpiñán Lamigueiro

Departamento de Ingeniería Eléctrica,  
Electrónica, Automática y Física aplicada

Madrid, 3 de septiembre de 2024



# Agradecimientos

Agradezco a ...



# 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 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 intradía.

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` represents an evolution of the existing `solaR` package, featuring significant improvements in modularity and efficiency. Unlike `solaR`, which relied on the `zoo` package for time series management, `solaR2` uses `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` provides advanced tools for simulating the performance of grid-connected systems and solar-powered water pumping systems. It includes classes, methods, and functions for calculating solar geometry and incident solar radiation, as well as estimating the final productivity of these systems from global horizontal irradiation on a daily and intraday basis.

The modular design based on S4 classes facilitates the management of multivariate time series and provides advanced visualization methods for performance analysis in large-scale photovoltaic plants. The use of `data.table` enhances data handling efficiency, allowing for faster and more precise analyses. Key functionalities include calculating solar radiation on different planes, estimating the performance of photovoltaic and pumping systems, and evaluating shading.

Additionally, `solaR2` offers advanced statistical visualization tools, compatible with other R packages for time series manipulation and spatial analysis. This makes it a valuable tool for researchers and professionals involved in the design and optimization of photovoltaic systems, enabling 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 the profitability of solar installations.

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





# Índice general

<b>Índice general</b>	<b>IX</b>
<b>Índice de figuras</b>	<b>XI</b>
<b>Nomenclatura</b>	<b>XIII</b>
<b>1 Introducción</b>	<b>1</b>
1.1. Objetivos . . . . .	1
1.2. Análisis previo de soluciones . . . . .	3
1.3. Aspectos técnicos . . . . .	3
<b>2 Estado del arte</b>	<b>7</b>
2.1. Situación actual de la generación fotovoltaica . . . . .	7
2.2. Solución actual y sus carencias . . . . .	8
<b>3 Marco teórico</b>	<b>11</b>
3.1. Naturaleza de la radiación solar . . . . .	11
3.2. Radiación en superficies inclinadas . . . . .	14
3.3. Cálculo de la energía producida por el generador . . . . .	18
<b>4 Desarrollo del código</b>	<b>27</b>
4.1. Geometría solar . . . . .	27
4.2. Datos meteorológicos . . . . .	33
4.3. Radiación en el plano horizontal . . . . .	36
4.4. Radiación efectiva en el plano del generador . . . . .	45
4.5. Producción eléctrica de un SFCR . . . . .	55
4.6. Producción eléctrica de un SFB . . . . .	59
4.7. Optimización de distancias . . . . .	62
4.8. Métodos de visualización . . . . .	66
<b>5 Ejemplo práctico de aplicación</b>	<b>77</b>
5.1. <code>solaR</code> . . . . .	77
5.2. <code>PVsyst</code> . . . . .	81
5.3. <code>solaR2</code> . . . . .	81
5.4. Comparación y conclusiones . . . . .	83
<b>A Código completo</b>	<b>85</b>
A.1. Constructores . . . . .	85
A.2. Clases . . . . .	109
A.3. Funciones . . . . .	112
A.4. Métodos . . . . .	141
A.5. Conjunto de datos . . . . .	165

**Bibliografía**

**169**

# Índice de figuras

3.1. Procedimiento de cálculo . . . . .	12
3.2. Perfil de irradiancia difusa y global obtenido a partir del generador empírico de [CR79] para valores de irradiancia tomadas cada 10 minutos . . . . .	15
3.3. Ángulo de visión del cielo . . . . .	16
3.4. Pérdidas angulares de un módulo fotovoltaico para diferentes grados de suciedad en función del ángulo de incidencia. . . . .	17
3.5. Curvas corriente-tensión(línea discontinua) y potencia-tensión(línea continua) de una célula solar ( $T_a = 20^{\circ}C$ y $G = 800W/m^2$ ) . . . . .	19
3.6. Evolución de la eficiencia de células según la tecnología (según el National Renewable Energy Laboratory [Nat24] (EEUU)). . . . .	20
4.1. Proceso de cálculo de las funciones de <b>solaR2</b> . . . . .	28
4.2. Cálculo de la geometría solar mediante la función <b>calcSol</b> , la cual unifica las funciones <b>fSolD</b> y <b>fSolI</b> resultando en un objeto clase <b>Sol</b> el cual contiene toda la información geométrica necesaria para realizar las siguientes estimaciones. . . . .	28
4.3. Los datos meteorologicas se pueden leer mediante las funciones <b>readG0dm</b> , <b>readBD</b> , <b>dt2Meteo</b> , <b>zoo2Meteo</b> y <b>readSIAR</b> las cuales procesan estos datos y los almacenan en un objeto de clase <b>Meteo</b> . . . . .	33
4.4. Cálculo de la radiación incidente en el plano horizontal mediante la función <b>calcG0</b> , la cual procesa un objeto clase <b>Sol</b> y otro clase <b>Meteo</b> mediante las funciones <b>fCompD</b> y <b>fCompI</b> resultando en un objeto clase <b>G0</b> . : . . . . .	37
4.5. Cálculo de la radiación efectiva incidente en el plano del generador mediante la función <b>calcGef</b> , la cual emplea la función <b>fInclin</b> para el computo de las componentes efectivas, la función <b>fTheta</b> que provee a la función anterior los ángulos necesarios para su computo y la función <b>calcShd</b> que reprocesa el objeto de clase <b>Gef</b> resultante, añadiendole el efecto de las sombras producidas entres módulos. . . . .	45
4.6. Estimación de la producción eléctrica de un SFCR mediante la función <b>prodGCPV</b> , la cual emplea la función <b>fProd</b> para el computo de la potencia a la entrada ( $P_{DC}$ ), a la salida ( $P_{AC}$ ) y el rendimiento ( $\eta_{inv}$ ) del inversor. . . . .	55
4.7. Estimación de la producción eléctrica de un SFB mediante la función <b>prodPVPS</b> , la cual emplea la función <b>fPump</b> para el computo del rendimiento de las diferentes parte de una bomba centrífuga alimentada por un convertidor de frecuencia. . . . .	59



# Nomenclatura

$A_c$	Área de una célula
$\alpha$	Ángulo de orientación de un sistema fotovoltaico
$AM$	Masa de aire
$AO$	Adelanto oficial durante el horario de verano
$B_0(0)$	irradiancia extra-atmósferica o extra-terrestre en el plano horizontal
$B_0$	Constante solar o irradiancia solar incidente en un plano normal al vector solar en el límite superior de la atmósfera terrestre
$B$	Radiación directa
$\beta$	Ángulo de inclinación de un sistema fotovoltaico
$D$	Radiación difusa
$D^C$	Radiación difusa circunsolar
$\delta$	Declinación
$\Delta\lambda$	Diferencia entre la longitud local y la longitud del huso horario
$D^I$	Radiación difusa isotrópica
$d_n$	Día del año
$EoT$	Ecuación del tiempo
$\epsilon_0$	Corrección debida a la excentricidad de la elipse de la trayectoria terrestre alrededor del sol
$F_D$	Fracción de difusa
$FT_B$	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
$H_f$	Altura asociada a las pérdidas de fricción en una tubería
$H_T$	Altura total incluyendo las pérdidas de fricción de la tubería

$H_v$	Altura vertical aparente
$K_T$	Índice de claridad
MPP	Punto de máxima potencia de un dispositivo fotovoltaico
$\omega$	Hora solar o tiempo solar verdadero
$\omega_s$	Ángulo del amanecer
$P_f$	Pérdidas de fricción en la tubería de un sistema de bombeo
$P_H$	Potencia hidráulica necesaria en un sistema de bombeo de agua
$P_H$	Potencia hidráulica
$\phi$	Latitud
$Q$	Caudal de agua
$Q_t$	Caudal de ensayo de un pozo
$R$	Radiación del albedo
$r_D$	Relación entre la irradiancia y la irradiación difusa en el plano horizontal
$\rho$	Densidad del agua
$\rho$	Coefficiente de reflexión del terreno para la irradiancia de albedo
STC	Condiciones estándar de medida de un dispositivo fotovoltaico
$T_c^*$	Temperatura de célula en condiciones estándar de medida
$T_c$	Temperatura de célula
$\theta_s$	Ángulo de incidencia o ángulo entre el vector solar y el vector director de una superficie
TO	Hora oficial
TONC	Temperatura de operación nominal de célula

# 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, si fuera necesario, se hará referencia al paquete desarrollado en este proyecto con el nombre `solaR2` [CITAR SOLAR2].

El usuario puede colocar los datos que considere convenientes (desde una base de datos oficial, una base de datos propia... etc.) 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, tuvo 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` basó 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 optó 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 necesitas 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.

- **CRAN**: 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 (A.1.1).
- Tratamiento de datos meteorológicos (en especial de radiación), procedentes de datos ofrecidos del usuario y de la red de estaciones SIAR [Min23] (A.1.8).
- Una vez calculado lo anterior, se pueden hacer estimaciones de:
  - Los componentes de radiación horizontal (A.1.2).
  - Los componentes de radiación eficaz en el plano inclinado (A.1.3).
  - La producción de sistemas fotovoltaicos conectados a red (A.1.4) y sistemas fotovoltaicos de bombeo (A.1.5).

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. El paquete **solar2** utiliza este paquete para representar de forma visual los datos obtenidos en las estimaciones.
- Junto con el modo Org, se ha utilizado el preprocesador de textos L<sup>A</sup>T<sub>E</sub>X (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

Este proyecto, como ya se ha comentado, es el heredero del paquete **solaR** desarrollado por Oscar Perpiñán. La filosofía de ambos paquetes es la misma y los resultados que dan son muy similares. Sin embargo, lo que les diferencia (a parte de que **solaR2** es más modular, es decir, tiene muchas funciones autónomas que permiten realizar cálculos específicos, en especial de geometría y radiación) es el paquete sobre el que construyen sus datos.

Mientras que **solaR** basa sus clases y métodos en el paquete **zoo**, **solaR2** en el paquete **data.table**. Los dos paquetes pueden trabajar con series temporales, pero, mientras que **zoo** es más eficaz trabajando con series temporales, **data.table** es más eficiente a la hora de trabajar con una cantidad grande de datos, lo cual a la hora de realizar estimaciones muy precisas es beneficioso.

Por otro lado, existen otras soluciones fuera de R:

1. **PVsyst - Photovoltaic Software** [PVS24] Este software es probablemente el más conocido dentro del ámbito del estudio y la estimación de instalaciones fotovoltaicas. Permite una gran personalización de todos los componentes de la instalación.
2. **SISIFO** [Sis24] Herramienta web diseñada por el **Grupo de Sistemas Fotovoltaicos del Instituto de Energía Solar de la Universidad Politécnica de Madrid**.
3. **PVGIS** [PVG24] Aplicación web desarrollada por el **European Commission Joint Research Center** desde 2001.
4. **System Advisor Model** [SAM24] Desarrollado por el **Laboratorio Nacional de Energías Renovables**, perteneciente al Departamento de energía del gobierno de EE.UU.

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

## 1.3. Aspectos técnicos

Las fuentes de un paquete de **R** están contenidas en un directorio que contiene al menos:

- Los ficheros **DESCRIPTION** y **NAMESPACE**
- Los subdirectorios:
  - **R**: código en ficheros **.R**
  - **man**: páginas de ayuda de las funciones, métodos y clases contenidas en el paquete.

Esta estructura puede ser generada con **package.skeleton**

### 1.3.1. DESCRIPTION

El fichero **DESCRIPTION** contiene la información básica:

```
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 **Package**, **Version**, **License**, **Title**, **Autor** y **Maintainer** son obligatorios.
- Si usa métodos **S4** debe incluir **Depends: methods**.

### 1.3.2. NAMESPACE

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"	"ESSR"	"package:stats"	"package:graphics"
[5] "package:grDevices"	"package:utils"	"package:datasets"	"package:methods"
[9] "AutoLoads"	"package:base"		

## Manejo de variables

- Exportar variables:

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

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

```
1 import(pkgExt)
```

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

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

## Manejo de clases y métodos

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

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

- Para usar clases y métodos **S4**:

```
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, ...)
```

### 1.3.3. Documentación

Las páginas de ayuda de los objetos **R** se escriben usando el formato “R documentation” (Rd), un lenguaje similar a  $\text{\LaTeX}$ .

```
\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}
```

## Estado del arte

### 2.1. Situación actual de la generación fotovoltaica

Según el informe anual de 2023 de la UNEF<sup>1</sup> [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 alcanzó a superar 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 record 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 REPowerEU<sup>2</sup> [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.

---

<sup>1</sup>UNEF: Unión Española Fotovoltaica.

<sup>2</sup>Plan 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+<sup>3</sup> [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 agrovoltaica. 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 PERTE<sup>4</sup>, 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 PIB<sup>5</sup>, 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. Solución actual y sus carencias

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**. El paquete permite realizar investigaciones reproducibles sobre el rendimiento de sistemas fotovoltaicos y la radiación solar, proporcionando métodos para calcular la geometría solar, la radiación incidente sobre un

---

<sup>3</sup>Plan + 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.

<sup>4</sup>PERTE: Proyecto Estratégico para la Recuperación y Transformación Económica.

<sup>5</sup>PIB: Producto Interior Bruto.

generador fotovoltaico, y simular el rendimiento de sistemas fotovoltaicos tanto conectados a la red como de bombeo de agua.

Pese a ser un herramienta muy capaz, **solaR** presenta una serie de carencias relativas al paquete **zoo**:

- **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. Por otro lado, **data.table** está diseñado específicamente para manejar grandes volúmenes de datos de manera eficiente, ofreciendo un rendimiento superior en operaciones de lectura, escritura y manipulación masiva de datos.
- **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 paralelización. Sin embargo, **data.table** supera esta limitación al proporcionar una infraestructura altamente optimizada para operaciones en paralelo y manejo de grandes conjuntos de datos, permitiendo que las aplicaciones escalen mejor en entornos de datos intensivos.
- **Manipulación de datos:** **zoo** es adecuado para manejar series temporales básicas, pero carece de las capacidades avanzadas de manipulación de datos que ofrece **data.table**, como la indexación rápida, las uniones eficientes, y la capacidad de realizar operaciones complejas de agrupamiento y agregación. Estas características de **data.table** permiten un manejo de datos más flexible y potente, lo cual es esencial en análisis de datos complejo y en tiempo real.
- **Consumo de memoria:** **zoo** puede consumir más memoria en comparación con **data.table** cuando se trabaja con grandes conjuntos de datos. Por otro lado, **data.table** está optimizado para operaciones en memoria, lo que permite manejar datasets más grandes sin requerir un incremento proporcional en el uso de recursos, haciendo que las operaciones sean más sostenibles en términos de memoria.

Por lo tanto, al adoptar **data.table** en **solaR2**, se abordarían estas limitaciones, proporcionando un paquete más robusto y capaz de manejar los desafíos actuales en el análisis de datos de radiación solar y de producción de sistemas fotovoltaicos.





## Marco teórico

El paquete **solarR2** 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.
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. Se trasladan estos valores de irradiación a valores de irradiancia.
5. Integrando estos valores se pueden 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. Se transforma en potencia en corriente alterna mediante un inversor que tiene una eficiencia asociada.
8. Integrando esta potencia se puede obtener la energía que produce el generador en un tiempo determinado.

### 3.1. Naturaleza de la radiación solar

Para el cálculo de la radiación solar que incide en una superficie se deben distinguir tres componentes diferenciados:

- **Radiación Directa**, B: fracción de radiación que procede en línea recta desde el Sol.
- **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.

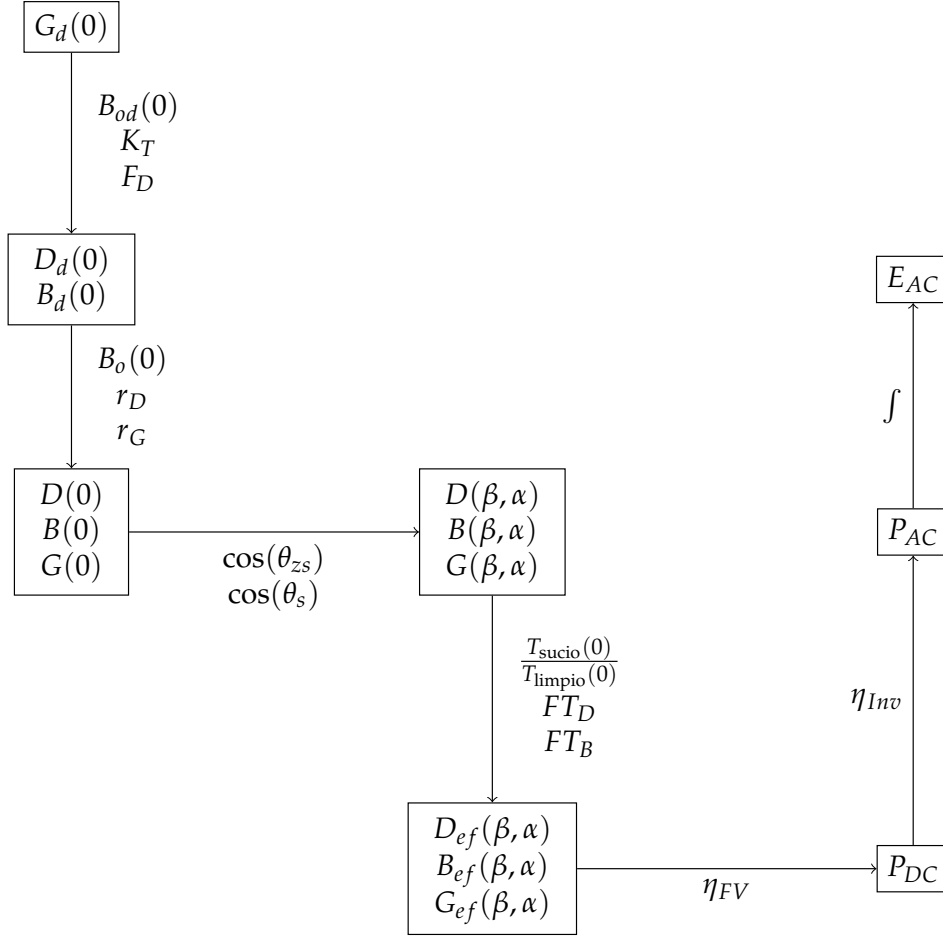


FIGURA 3.1: Procedimiento de cálculo

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

$$G = B + D + R \quad (3.1)$$

Tomando como base el libro antes mencionado [Per23], se describirá el proceso que se ha de seguir para obtener una estimación de las componenetes directa y difusa a partir del dato de radiación global, dado que es el que comúnmente se puede obtener de una localización determinada.

### 3.1.1. Radiación fuera de la atmósfera terrestre

Lo primero que se menciona en dicho proceso es la obtención de la irradiancia denominada extra-terrestre o extra-atmosférica, que es la radiación que llega a la atmósfera, directamente desde el Sol, que no sufre ninguna pérdida por interaccionar con algún medio. Como la relación entre el tamaño de nuestro planeta y la distancia entre el Sol y la Tierra es muy reducida, es posible asumir que el valor de dicha irradiancia es constante, siendo este valor  $B_0 = 1367 \frac{W}{m^2}$ , según varias mediciones. Como la órbita que describe la Tierra alrededor del Sol no es totalmente circular, sino que tiene forma de elipse, para calcular la irradiancia incidente en una superficie tangente a la atmosfera en ua latitud concreta, debemos aplicar un factor de corrección de la excentricidad de la elipse:

$$B_0(0) = B_0 \epsilon_0 \cos \theta_{zs} \quad (3.2)$$

Siendo cada componente:

- Constante solar:  $B_0 = 1367 \frac{W}{m^2}$
- Factor de corrección por excentricidad:  $\epsilon_0 = (\frac{r_0}{r})^2 = 1 + 0,033 \cdot \cos(\frac{2\pi d_n}{365})^1$
- Ángulo zenital solar:  $\cos(\theta_{zs}) = \cos(\delta)\cos(\omega)\cos(\phi) + \sin(\delta)\sin(\phi)$  {Ángulo cenital solar}

Donde:

- Declinación:  $\delta = 23,45^\circ \cdot \sin(\frac{2\pi \cdot (d_n + 284)}{365})$  donde  $d_n$  es el día del año.
- Latitud:  $\phi$
- Hora solar o tiempo solar verdadero:  $\omega = 15 \cdot (TO - AO - 12) + \Delta\lambda + \frac{EoT}{4}$

Donde:

- Hora oficial:  $TO$
- Adelanto oficial durante el horario de verano:  $AO$
- Diferencia entre la longitud local y la longitud del huso horario:  $\Delta\lambda$
- Ecuación del tiempo:  $EoT = 229,18 \cdot (-0,0334 \cdot \sin(\frac{2\pi}{365,24} \cdot dn) + 0,04184 \cdot \sin(2 \cdot \frac{2\pi}{365,24} \cdot dn + 3,5884))$

Esta irradiancia extra-terrestre solo tiene componentes geométricas. De modo que, si integramos la ecuación 3.2, se obtiene la irradiación diaria extra-terrestre:

$$B_{0d}(0) = -\frac{T}{\pi} B_0 \epsilon_0 (\omega_s \sin \phi \sin \delta + \cos \phi \cos \delta \sin \omega_s) \quad (3.3)$$

Siendo:

- Ángulo del amanecer:

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

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

<sup>1</sup>Para las ecuaciones de este apartado se va a optar por poner la ecuación más simple posible. Sin embargo, el paquete **solAR2** otorga la posibilidad de realizar los cálculos de utilizando las ecuaciones propuestas por 4 autores diferentes.

<sup>2</sup>Se van a utilizar las ecuaciones propuestas por P.I. Cooper [Coo69] por su simpleza.

### 3.1.2. 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: {Índice de claridad diario}

$$K_{Td} = \frac{G_d(0)}{B_{0d}(0)} \quad (3.4)$$

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

$$K_{Tm} = \frac{G_{d,m}(0)}{B_{0d,m}(0)} \quad (3.5)$$

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]:

$$F_{Dm} = 1 - 1,13 \cdot K_{Tm} \quad (3.6)$$

Donde:

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

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

$$D_d(0) = F_D \cdot G_d(0) \quad (3.7)$$

$$B_d(0) = G_d(0) - D_d(0) \quad (3.8)$$

## 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] (3.9):

$$r_D = \frac{D(0)}{D_d(0)} = \frac{B_0(0)}{B_{0d}(0)} \quad (3.9)$$

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 3.2 3.3:

$$\frac{B_0(0)}{B_{0d}(0)} = \frac{\pi}{T} \cdot \frac{\cos(\omega) - \cos(\omega_s)}{\omega_s \cdot \cos(\omega_s) - \sin(\omega_s)} = r_D \quad (3.10)$$

el mismo análisis mostró una relación entre la irradiancia e irradiación global asimilable a una función dependiente de la hora solar (3.11):

$$r_G = \frac{G(0)}{G_d(0)} = r_D \cdot (a + b \cdot \cos(w)) \quad (3.11)$$

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, y 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:

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

$$B(\beta, \alpha) = B(0) \cdot \frac{\max(0, \cos(\theta_s))}{\cos(\theta_{zs})} \quad (3.12)$$

donde:

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

$$D(\beta, \alpha) = D^I(\beta, \alpha) + D^C(\beta, \alpha) \quad (3.13)$$

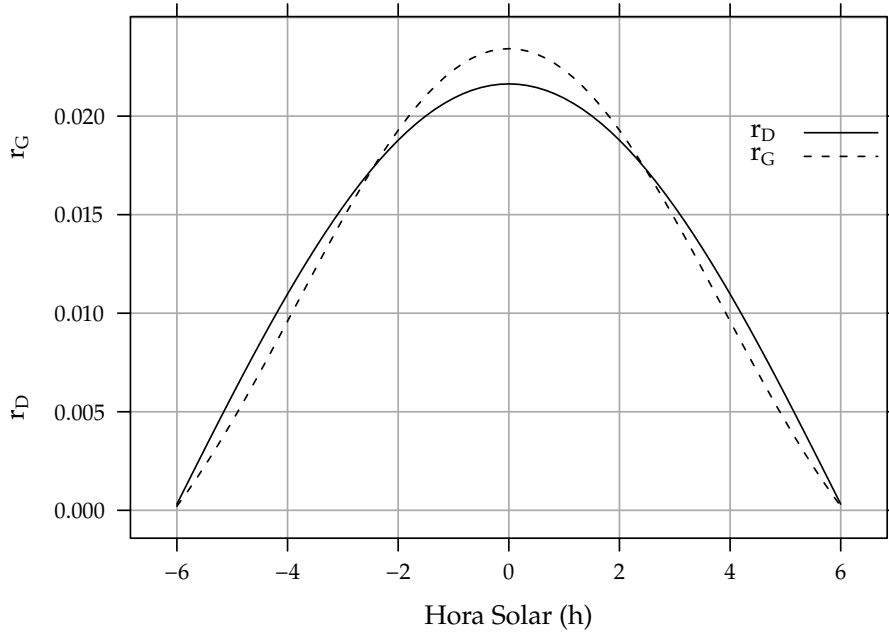


FIGURA 3.2: Perfil de irradiancia difusa y global obtenido a partir del generador empírico de [CR79] para valores de irradiancia tomadas cada 10 minutos

$$D^I(\beta, \alpha) = D(0)(1 - k_1) \cdot \frac{1 + \cos(\beta)}{2} \quad (3.14)$$

$$D^C(\beta, \alpha) = D(0) \cdot k_1 \cdot \frac{\max(0, \cos(\theta_s))}{\cos(\theta_{zs})} \quad (3.15)$$

Donde:

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

- **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,  $\rho$ , que depende del terreno. En la ecuación 3.16, se utiliza el factor  $\frac{1 - \cos(\beta)}{2}$ , complementario al factor de visión de la difusa isotrópica (figura 3.3)

$$R(\beta, \alpha) = \rho \cdot G(0) \cdot \frac{1 - \cos(\beta)}{2} \quad (3.16)$$

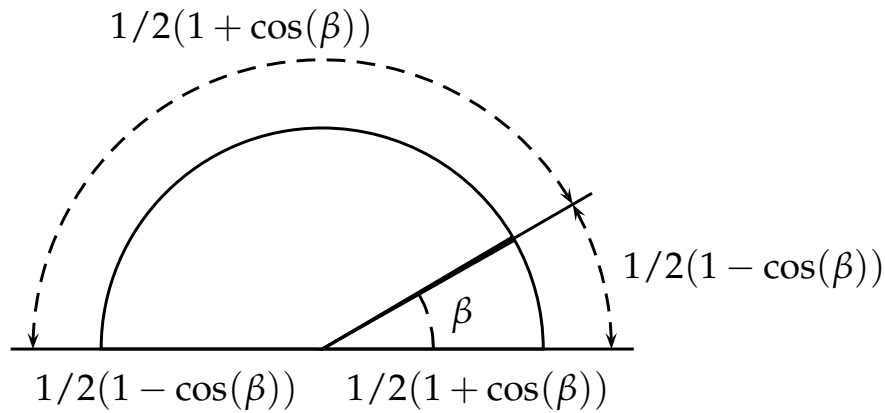
### 3.2.2. Ángulo de incidencia y suciedad

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  $\theta_s$ . 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 3.17:

$$B_{ef}(\beta, \alpha) = B(\beta, \alpha) \cdot \left[ \frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FTB(\theta_s)) \quad (3.17)$$

donde  $FTB(\theta_s)$  es el factor de pérdidas angulares, que se calcula con la ecuación 3.18:

$$FTB(\theta_s) = \frac{\exp(-\frac{\cos(\theta_s)}{a_r}) - \exp(-\frac{1}{a_r})}{1 - \exp(-\frac{1}{a_r})} \quad (3.18)$$




---

FIGURA 3.3: Ángulo de visión del cielo

Este factor depende el ángulo de incidencia  $\theta_s$  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  $\theta_s$  supera los  $60^\circ$ , y se acentúan con mayor suciedad.

Para calcular las componente de radiación difusa isotrópica y de albedo se utilizan las ecuaciones 3.19 y 3.2.2:

$$FTD(\beta) \approx \exp\left[-\frac{1}{a_r} \cdot \left(c_1 \cdot \left(\sin\beta + \frac{\pi - \beta - \sin\beta}{1 + \cos\beta}\right) + c_2 \cdot \left(\sin\beta + \frac{\pi - \beta - \sin\beta}{1 + \cos\beta}\right)^2\right)\right] \quad (3.19)$$

$$FTR(\beta) \approx \exp\left[-\frac{1}{a_r} \cdot \left(c_1 \cdot \left(\sin\beta + \frac{\beta - \sin\beta}{1 - \cos\beta}\right) + c_2 \cdot \left(\sin\beta + \frac{\beta - \sin\beta}{1 - \cos\beta}\right)^2\right)\right] \quad (3.20)$$

Donde:

- Ángulo de inclinación del generador (en radianes):  $\beta$
- 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 componenetes el cálculo de irradiancia efectiva es similar al de la irradiancia directa (ecuaciones 3.21 y 3.23). Para la componente difusa circunsolar emplearemos el factor de pérdidas angulares de la irradiancia efectiva (ecuacion 3.22):

$$D_{ef}^I(\beta, \alpha) = D^I(\beta, \alpha) \cdot \left[\frac{T_{sucio}(0)}{T_{limpio}(0)}\right] \cdot (1 - FTD(\beta)) \quad (3.21)$$

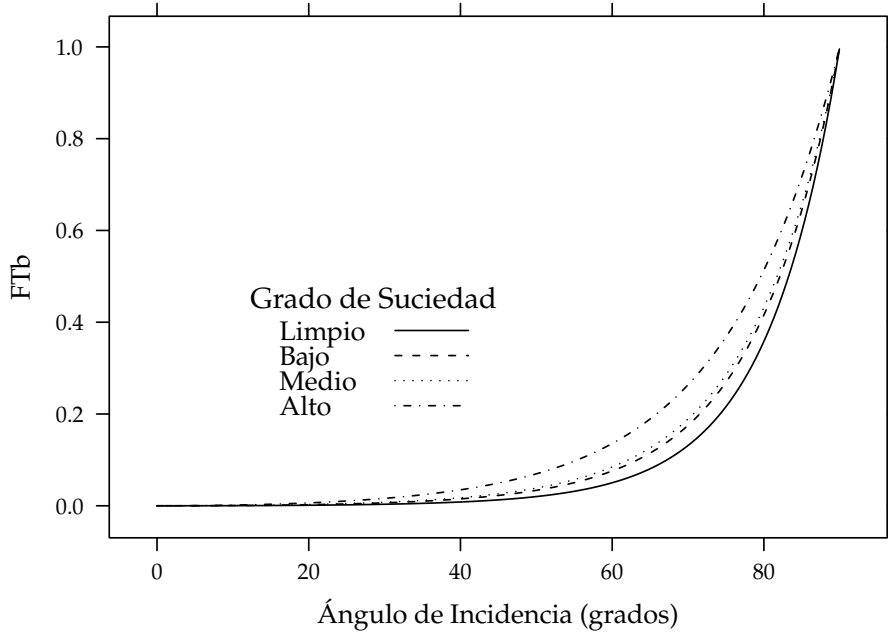


FIGURA 3.4: Pérdidas angulares de un módulo fotovoltaico para diferentes grados de suciedad en función del ángulo de incidencia.

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

$$D_{ef}^C(\beta, \alpha) = D^C(\beta, \alpha) \cdot \left[ \frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FT_B(\theta_s)) \quad (3.22)$$

$$R_{ef}(\beta, \alpha) = R(\beta, \alpha) \cdot \left[ \frac{T_{sucio}(0)}{T_{limpio}(0)} \right] \cdot (1 - FT_R(\beta)) \quad (3.23)$$

Siguiendo el esquema de la figura 3.1, a partir de estas irradiancias efectivas se puede calcular la irradiación global efectiva diaria, mensual y anual. 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 el generador

#### 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 solar y en los valores de tensión y corriente que se alcanzan en dichas condiciones.

Para definir una célula solar, se tomar 4 variables:

- La corriente de cortocircuito:  $I_{sc}$ {Corriente de cortocircuito de una célula}
- La tensión de circuito abierto:  $V_{oc}$ {Tensión de circuito abierto de una célula}
- La corriente en el punto de máxima potencia:  $I_{mpp}$ {Corriente de una célula en el punto de máxima potencia}
- La tensión en el punto de máxima potencia:  $V_{mpp}$ {Tensión de una célula en el punto de máxima potencia}

#### 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 es observable 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:

$$FF = \frac{I_{mpp} \cdot V_{mpp}}{I_{sc} \cdot V_{oc}} \quad (3.24)$$



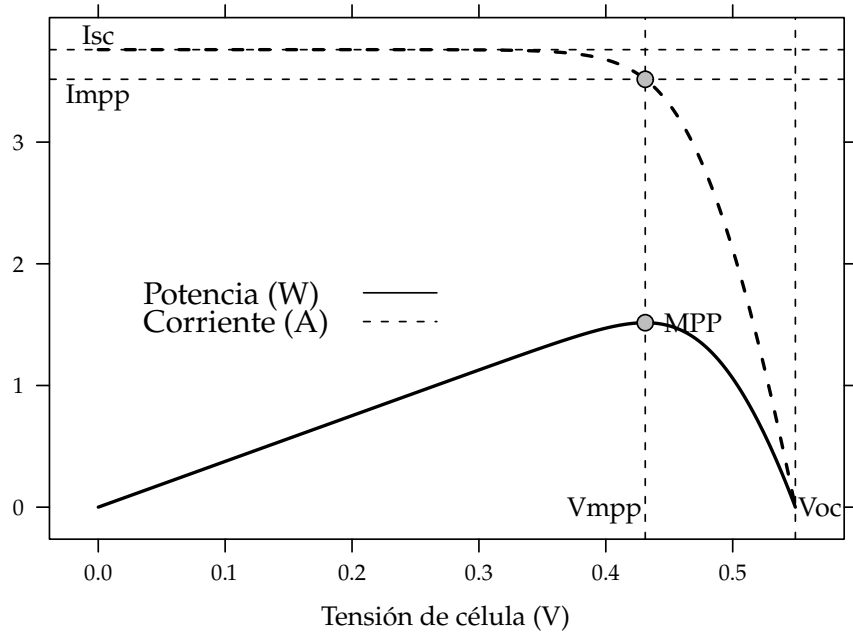


FIGURA 3.5: Curvas corriente-tensión(línea discontinua) y potencia-tensión(línea continua) de una célula solar ( $T_a = 20^\circ\text{C}$  y  $G = 800\text{W}/\text{m}^2$ )

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

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

$$\eta = \frac{I_{mpp} \cdot V_{mpp}}{P_L} \quad (3.25)$$

donde  $P_L = A_c \cdot G_{ef}$  representa la potencia luminosa que incide en la célula. Como es evidente de la ecuación 3.25, este valor de eficiencia se corresponde al caso en el que el acoplamiento entre la carga y la célula permite a ésta 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  $-2,3 \frac{\text{mV}}{^\circ\text{C}}$ . 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 fotocorriente 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:

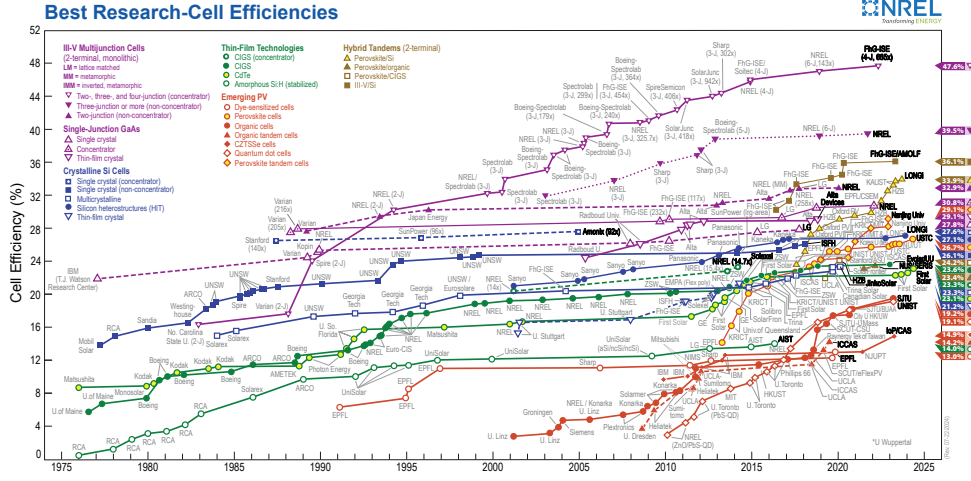


FIGURA 3.6: Evolución de la eficiencia de células según la tecnología (según el National Renewable Energy Laboratory [Nat24] (EEUU)).

- Irradiancia:  $G_{stc} = 1000W/m^2$  con incidencia normal. {Irradiancia incidente en condiciones estandar de medida}
- Temperatura de célula:  $T_c^* = 25^\circ C$ .
- Masa de aire:  $AM = 1,5$ .<sup>3</sup>

Frecuentemente los fabricantes informan de los valores de las tensiones  $V_{oc}^*$  y  $V_{mpp}^*$  y las corrientes  $I_{sc}^*$  y  $I_{mpp}^*$ .<sup>4</sup> 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 ue definen el comportamiento de un módulo fotovoltaico se establecen en lo que se conocen 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 ésta 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.

<sup>3</sup>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 ( $AM = 1/\cos\theta_{zs}$ ).

<sup>4</sup>Es de uso común añadir un asterisco como superíndice para denotar aquellos parámetros medidos en estas condiciones.

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

La ecuación 3.26 expresa una aproximación aceptable del comportamiento térmico de una célula integrada en un módulo en base a las consideraciones previas:

$$T_c = T_a + G_{ef} \cdot \frac{NOCT - 20}{800} \quad (3.26)$$

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  $T_a = 25^\circ \text{C}$  para simulaciones 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.

### Cálculo de $V_{oc}$ y $I_{sc}$

Conociendo ya los valores horarios de temperatura de la célula, se puede calcular  $V_{oc}$  utilizando la ecuación 3.27. Y, por último, mediante la ecuación 3.28 se puede calcular  $I_{sc}$ .

$$V_{oc}(T_c) = V_{oc}^* + (T_c - T_c^*) \cdot \frac{dV_{oc}}{dT_c} \cdot N_{cs} \quad (3.27)$$

$$I_{sc} = G_{ef} \cdot \frac{I_{sc}^*}{G^*} \quad (3.28)$$

### 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, y por tanto es un punto de interés para el cálculo.

Existen dos metodologías de cálculo de dicho punto, uno de ellos significativamente más sencillo que el otro. Éste consiste en suponer que el Factor de Forma, definido en la expresión 3.24 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

$$FF = FF^* \quad (3.29)$$

entonces

$$\frac{I_{mpp} \cdot V_{vmpp}}{I_{sc} \cdot V_{oc}} = \frac{I_{mpp}^* \cdot V_{vmpp}^*}{I_{sc}^* \cdot V_{oc}^*} \quad (3.30)$$

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

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

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

$$V_{tn} = \frac{V_t \cdot (273 + 25)}{300} \quad (3.31)$$

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

$$R_s^* = \frac{\frac{V_{oc}^*}{N_{cs}} - \frac{V_{mpp}^*}{N_{cs}} + m \cdot V_{tn} \cdot \ln(1 - \frac{I_{mpp}^*}{I_{sc}^*})}{\frac{I_{mpp}^*}{N_{cp}}} \quad (3.32)$$

- **Cálculo de  $r_s$ :** Utilizando los 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.

$$r_s = R_s^* \cdot \left( \frac{N_{cs}}{N_{cp}} \cdot \frac{I_{sc}}{V_{oc}} \right) \quad (3.33)$$

- **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:

$$k_{oc} = \frac{V_{oc}/N_{cs}}{m \cdot V_t \cdot \frac{T_c + 273}{300}} \quad (3.34)$$

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

$$i_{mpp} = 1 - \frac{D_M}{k_{oc}} \quad (3.35)$$

$$v_{mpp} = 1 - \frac{\ln(k_{oc}/D_M)}{k_{oc}} - r_s \cdot i_{mpp} \quad (3.36)$$

donde:

$$D_M = D_{M0} + 2 \cdot r_s \cdot D_{M0}^2 \quad (3.37)$$

$$D_{M0} = \frac{k_{oc} - 1}{k_{oc} - \ln k_{oc}} \quad (3.38)$$

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:

$$P_{mpp} = I_{mpp} \cdot V_{mpp} \quad (3.39)$$

### 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 sfer, se debe tener en cuenta su configuración de módulos en serie y en paralelo.

$$P_g^* = N_s \cdot N_p \cdot P_m^* \quad (3.40)$$

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  $P_{inv}$  {Potencia nominal de un inversor} la potencia nominal del inversor:

$$p_i = \frac{P_{DC}}{P_{inv}} \quad (3.41)$$

$$p_o = \frac{P_{AC}}{P_{inv}} \quad (3.42)$$

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

$$\eta_{inv} = \frac{p_o}{p_o + k_0 + k_1 p_o + k_2 p_o^2} \quad (3.43)$$

De las dos ecuaciones anteriores se puede deducir:

$$p_i = p_o + k_0 + k_1 p_o + k_2 p_o^2 \quad (3.44)$$

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

$$k_2 p_o^2 + (k_1 + 1)p_o + (k_0 - p_i) = 0 \quad (3.45)$$

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

$$P_{AC} = p_o \cdot P_{inv} \quad (3.46)$$

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

$$E_{AC} = \int_T P_{AC} dt \quad (3.47)$$

y la productividad:

$$Y_f = \frac{E_{ac}}{P_g^*} \quad (3.48)$$

### 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,  $Q$

$$P_H = g \cdot \rho \cdot Q \cdot H_V \quad (3.49)$$

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

$$P_H = 2,725 \cdot Q \cdot H_v \quad (3.50)$$

Asumiendo que el agua bombeado sale por el co ducto 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:

$$H_T = H_v + H_f \quad (3.51)$$

La potencia eléctrica a la entrada de la motobomba,  $P_{el}$ , {Potencia eléctrica necesaria en la entrada de una motobomba} es:

$$P_{el} = \frac{P_H + P_f}{\eta_{mp}} \quad (3.52)$$

donde  $\eta_{mp}$  es la eficiencia de la motobomba. {Eficiencia de una motobomba}

La potencia eléctrica requerida por la motobomba es entregada por un generador FV y acondicionador de potencia:

$$P_{el} = P_g^* \cdot \frac{G}{G_{stc}} \cdot \frac{\eta_g}{\eta_g^*} \cdot \eta_{inv} \quad (3.53)$$

siendo  $G$  la irradiancia en el plano del generador,  $\eta_{inv}$  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:

$$Q_d = \int_d \frac{G}{G^*} \cdot P_g^* \cdot \eta_g \cdot \frac{\eta_{ig}}{\eta_{ig}^*} \cdot \eta_{inv} \cdot \eta_{mp} dt \quad (3.54)$$

### Altura

Se puede definir una altura total equivalente,  $H_{TE}$  {Altura total equivalente de un sistema de bombeo}, 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.

$$Q_d = \frac{P_g^*}{2,725 \cdot G^* \cdot H_{TE}} \int \left( \frac{G}{\eta_{ig}^* \eta_m^* \eta_{inv} \eta_{mp}} \right) dt \quad (3.55)$$

Ahora el cálculo en la integral sólo 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 estático de un pozo}
  - Nivel dinámico,  $H_{dt}$ . {Nivel dinámico de un pozo}

- 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,  $Q_{max}$  ( $Q_t = Q_{max}$ ). {caudal máximo del pozo}

Con estas suposiciones se puede llegar a la expresión:

$$H_{TE} = H_{ot} + H_{st} + \left( \frac{H_{dt} - H_{st}}{Q_T} \right) Q_{AP} + H_f(Q_{AP}) \quad (3.56)$$

donde:

- $H_{OT}$ , es la altura desde la salida de agua hasta el suelo. {Diferencia de cotas entre la salida de agua y la entrada en el depósito}
- Nivel estático,  $H_{st}$ .
- Nivel dinámico,  $H_{dt}$ .
- Caudal aparente,  $Q_{AP} = \alpha \cdot Q_d$  ( $\alpha = 1/24 = 0,0416h^{-1}$ ).
- $H_f(Q_{AP})$ , 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:

$$P_g^* = \frac{10 \cdot H_{TE} \cdot Q_d}{\frac{G_d}{G_{stc}}} \quad (3.57)$$

### 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 ordenaor. 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:

$$H = a \cdot f^2 + b \cdot f \cdot Q + c \cdot Q^2 \quad (3.58)$$

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

2. Relaciones de semejanza para bombas centrífugas:

$$\frac{f_1}{f_2} = \frac{Q_1}{Q_2} = \left( \frac{H_1}{H_2} \right)^{1/2} = \left( \frac{P_1}{P_2} \right)^{1/3} \quad (3.59)$$

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

$$Q_{50} = \frac{50 \cdot Q}{f} \quad (3.60)$$

$$H_{50} = H \cdot \left( \frac{50}{f} \right)^2 \quad (3.61)$$

4. Ecuación de potencia hidráulica:

$$P_{h,50} = 2,725 \cdot Q_{50} \cdot H_{50} \quad (3.62)$$

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

$$P_{b,50} = \frac{P_{h,50}}{\eta_b} \quad (3.63)$$

6. Potencia mecánica a frecuencia  $f$ :

$$P_b = P_{b,50} \cdot \left(\frac{f}{50}\right)^3 \quad (3.64)$$

7. Potencia eléctrica demandada por el motor:

$$P_{bc} = P_b \cdot \frac{50}{f} \quad (3.65)$$

$$P_{e,50} = \frac{P_{bc}}{\eta_m} \quad (3.66)$$

$$P_e = P_{e,50} \cdot \frac{f}{50} \quad (3.67)$$

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

$$G = G_{max} \cdot \cos\left(\frac{t}{t_0} \cdot \frac{\pi}{2}\right) \cdot \left[1 + s \cdot \left(1 - \cos\left(\frac{t}{t_0} \cdot \frac{\pi}{2}\right)\right)\right] \quad (3.68)$$

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:

$$s = \frac{d \cdot \frac{\pi}{2} - 1}{1 - \frac{\pi}{4}} \quad (3.69)$$

siendo  $d$  el factor de conjunto de datos calculado con:

$$d = \frac{G_d}{G_{max} \cdot h} \quad (3.70)$$

### 3.3.5. Cálculo y optimización de sombras



## 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 programa sigue los siguientes procesos:

### 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** [A.1.1], la cual mediante las funciones **fSolD** [A.3.9] y **fSolI** [A.3.10] calcula todos los ángulos y componentes que caracterizan la geometría solar.

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

- **fSolD**: la cual, a partir de la latitud ( $\phi$ ), calcula la geometría a nivel diario, es decir, los ángulos y componentes que se pueden calcular en cada día independiente.

Estas son:

- Declinación ( $\delta$ ): calculada a partir de la función **declination**<sup>1</sup>.
- Excentricidad ( $\epsilon_o$ ): obtenida mediante la función **eccentricity**.
- Ecuación del tiempo ( $EoT$ ): obtenida mediante la función **eot**.
- Ángulo del amanecer ( $\omega_s$ ): 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
	<IDat>	<num>	<num>	<num>	<num>	<num>	<num>
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

<sup>1</sup>Todas las funciones mencionadas en este punto, se encuentran en el apartado A.3.19.

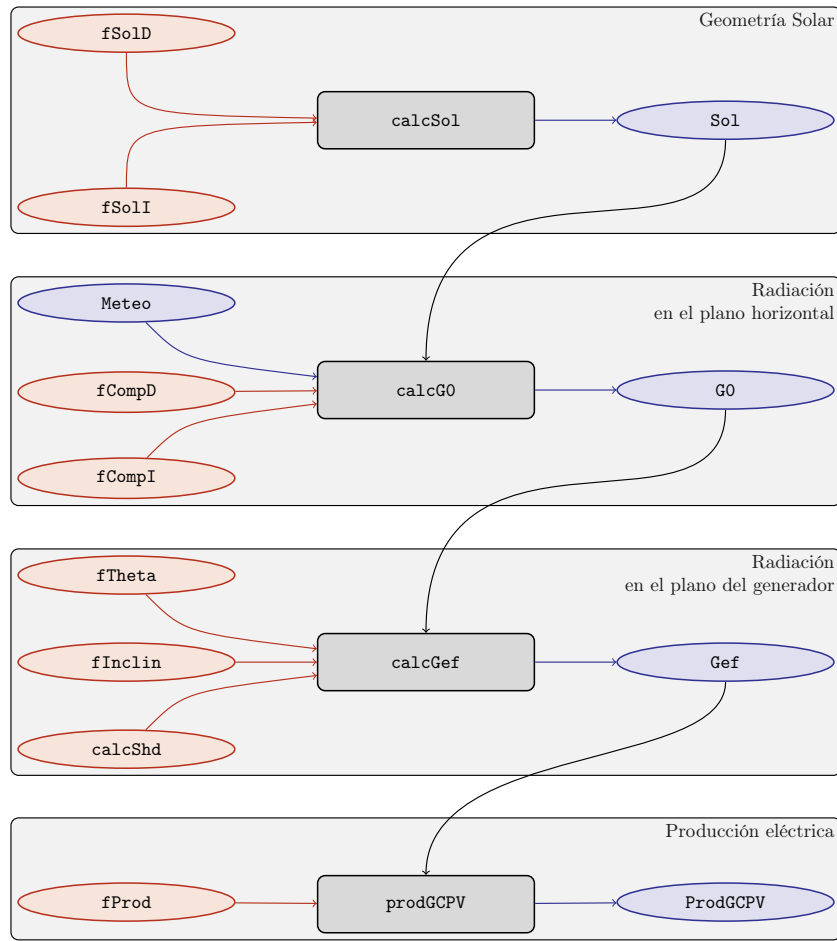


FIGURA 4.1: Proceso de cálculo de las funciones de **solar2**

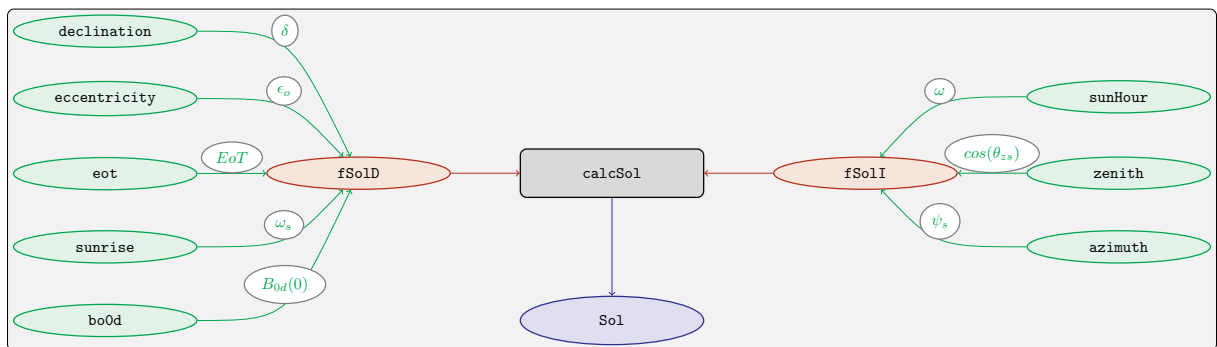


FIGURA 4.2: Cálculo de la geometría solar mediante la función **calcSol**, la cual unifica las funciones **fSolD** y **fSolI** resultando en un objeto clase **Sol** el cual contiene toda la información geométrica necesaria para realizar las siguientes estimaciones.

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

Además, **fSold** permite seleccionar el método de cálculo entre los propuestos por 4 autores diferentes (**cooper** [Coo69], **spencer** [Spe71], **strous** [Str11], **michalsky** [Mic88])(el valor por defecto es **michalsky**):

```
1 sold_cooper <- fSold(lat = lat, BTd = BTd, method = 'cooper')
2 show(sold_cooper)
```

Key: <Dates>							
	Dates	lat	decl	eo	EoT	ws	BoOd
	<IDat>	<num>	<num>	<num>	<num>	<num>	<num>
1:	2024-01-17	37.2	-0.36506987	1.0315970	-0.0455346238	-1.276457	4702.617
2:	2024-02-14	37.2	-0.23770977	1.0235842	-0.0614793356	-1.385835	6024.833
3:	2024-03-15	37.2	-0.04219743	1.0091112	-0.0368674274	-1.538742	7968.679
4:	2024-04-15	37.2	0.17074888	0.9917107	0.0017482721	-1.702053	9870.335
5:	2024-05-15	37.2	0.33214647	0.9770196	0.0143055938	-1.835696	11107.378
6:	2024-06-10	37.2	0.40292516	0.9690335	-0.0007378952	-1.900263	11575.213
7:	2024-07-18	37.2	0.36346384	0.9684861	-0.0263454380	-1.863677	11260.684
8:	2024-08-18	37.2	0.21721704	0.9778484	-0.0111761118	-1.739110	10144.635
9:	2024-09-18	37.2	0.01056696	0.9933706	0.0342189964	-1.578817	8367.014
10:	2024-10-19	37.2	-0.19902155	1.0107363	0.0689613044	-1.417100	6356.454
11:	2024-11-18	37.2	-0.34965673	1.0247443	0.0575423573	-1.290358	4835.353
12:	2024-12-13	37.2	-0.40651987	1.0315970	0.0158622941	-1.237915	4260.830

```
1 sold_spencer <- fSold(lat = lat, BTd = BTd, method = 'spencer')
2 show(sold_spencer)
```

Key: <Dates>							
	Dates	lat	decl	eo	EoT	ws	BoOd
	<IDat>	<num>	<num>	<num>	<num>	<num>	<num>
1:	2024-01-17	37.2	-0.36483670	1.0340422	-0.0455346238	-1.276669	4716.264
2:	2024-02-14	37.2	-0.23199205	1.0259717	-0.0614793356	-1.390501	6100.057
3:	2024-03-15	37.2	-0.03563921	1.0107943	-0.0368674274	-1.543730	8048.574
4:	2024-04-15	37.2	0.17171286	0.9926547	0.0017482721	-1.702813	9888.522
5:	2024-05-15	37.2	0.33007088	0.9775162	0.0143055938	-1.833871	11096.093
6:	2024-06-10	37.2	0.40208757	0.9691480	-0.0007378952	-1.899469	11570.124
7:	2024-07-18	37.2	0.36657157	0.9675489	-0.0263454380	-1.866501	11274.319
8:	2024-08-18	37.2	0.22748717	0.9758022	-0.0111761118	-1.747427	10212.886
9:	2024-09-18	37.2	0.03143967	0.9907919	0.0342189964	-1.594670	8548.821
10:	2024-10-19	37.2	-0.17549393	1.0088406	0.0689613044	-1.435795	6590.939
11:	2024-11-18	37.2	-0.33679169	1.0245012	0.0575423573	-1.301800	4971.285
12:	2024-12-13	37.2	-0.40419949	1.0328516	0.0158622941	-1.240121	4290.674

```
1 sold_strous <- fSold(lat = lat, BTd = BTd, method = 'cooper')
2 show(sold_strous)
```

Key: <Dates>							
	Dates	lat	decl	eo	EoT	ws	BoOd
	<IDat>	<num>	<num>	<num>	<num>	<num>	<num>
1:	2024-01-17	37.2	-0.36506987	1.0315970	-0.0455346238	-1.276457	4702.617
2:	2024-02-14	37.2	-0.23770977	1.0235842	-0.0614793356	-1.385835	6024.833
3:	2024-03-15	37.2	-0.04219743	1.0091112	-0.0368674274	-1.538742	7968.679
4:	2024-04-15	37.2	0.17074888	0.9917107	0.0017482721	-1.702053	9870.335
5:	2024-05-15	37.2	0.33214647	0.9770196	0.0143055938	-1.835696	11107.378
6:	2024-06-10	37.2	0.40292516	0.9690335	-0.0007378952	-1.900263	11575.213
7:	2024-07-18	37.2	0.36346384	0.9684861	-0.0263454380	-1.863677	11260.684
8:	2024-08-18	37.2	0.21721704	0.9778484	-0.0111761118	-1.739110	10144.635
9:	2024-09-18	37.2	0.01056696	0.9933706	0.0342189964	-1.578817	8367.014
10:	2024-10-19	37.2	-0.19902155	1.0107363	0.0689613044	-1.417100	6356.454
11:	2024-11-18	37.2	-0.34965673	1.0247443	0.0575423573	-1.290358	4835.353
12:	2024-12-13	37.2	-0.40651987	1.0315970	0.0158622941	-1.237915	4260.830

- **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:

- La hora solar o tiempo solar verdadero ( $\omega$ ): 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**)<sup>2</sup>.
- El coseno del ángulo cenital solar ( $\cos(\theta_{zs})$ ): obtenida a partir de la función **zenith**.
- La altura solar ( $\gamma_s$ ): obtenida a partir del resultado anterior<sup>3</sup>.
- El ángulo acimutal solar ( $\theta_{zs}$ ): 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 excentricidad (calculada en **fSold**) [ecuación 3.2].

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

Index: <night>								
	Dates	lat	w	night	cosThzS	AlS	AzS	BoO
	<POSs>	<num>	<num>	<lgcl>	<num>	<num>	<num>	<num>
1:	2024-01-17 00:00:00	37.2	3.09905026	TRUE	-0.958552332	-1.281876984	3.00157749	0.00000
2:	2024-01-17 01:00:00	37.2	-2.92239722	TRUE	-0.941407376	-1.226779122	-2.49462689	0.00000
3:	2024-01-17 02:00:00	37.2	-2.66065932	TRUE	-0.874749489	-1.064918604	-2.03862388	0.00000
4:	2024-01-17 03:00:00	37.2	-2.39892132	TRUE	-0.763119126	-0.868125900	-1.77932134	0.00000
5:	2024-01-17 04:00:00	37.2	-2.13718324	TRUE	-0.614120126	-0.661270606	-1.59701536	0.00000
6:	2024-01-17 05:00:00	37.2	-1.87544507	TRUE	-0.437901763	-0.453263434	-1.44469585	0.00000
7:	2024-01-17 06:00:00	37.2	-1.61370681	TRUE	-0.246467423	-0.249033534	-1.30093496	0.00000
8:	2024-01-17 07:00:00	37.2	-1.35196846	TRUE	-0.052856976	-0.052881619	-1.15283370	0.00000
9:	2024-01-17 08:00:00	37.2	-1.09023003	FALSE	0.129741461	0.130108233	-0.99014548	183.39419
10:	2024-01-17 09:00:00	37.2	-0.82849151	FALSE	0.288889848	0.293067041	-0.80329847	408.35612
11:	2024-01-17 10:00:00	37.2	-0.56675290	FALSE	0.413747472	0.426566560	-0.58400587	584.84684
12:	2024-01-17 11:00:00	37.2	-0.30501420	FALSE	0.495809380	0.518766586	-0.32921922	700.84427
13:	2024-01-17 12:00:00	37.2	-0.04327541	FALSE	0.529485721	0.557994217	-0.04769723	748.44699
14:	2024-01-17 13:00:00	37.2	0.21846346	FALSE	0.512482515	0.538073327	0.23821864	724.41235
15:	2024-01-17 14:00:00	37.2	0.48020243	FALSE	0.445957919	0.462244212	0.50355560	630.37745
16:	2024-01-17 15:00:00	37.2	0.74194148	FALSE	0.334443348	0.341014503	0.73469016	472.74762

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

<sup>3</sup> $\gamma_s = \arcsin(\cos(\theta_s))$ .

17:	2024-01-17 16:00:00	37.2	1.00368062	FALSE	0.185534810	0.186616094	0.93148844	262.26008
18:	2024-01-17 17:00:00	37.2	1.26541985	FALSE	0.009375501	0.009375638	1.10112996	13.25261
19:	2024-01-17 18:00:00	37.2	1.52715917	TRUE	-0.182035120	-0.183055757	1.25297092	0.00000
20:	2024-01-17 19:00:00	37.2	1.78889857	TRUE	-0.375658695	-0.385107424	1.39694027	0.00000
21:	2024-01-17 20:00:00	37.2	2.05063807	TRUE	-0.558306105	-0.592342658	1.54466726	0.00000
22:	2024-01-17 21:00:00	37.2	2.31237766	TRUE	-0.717535874	-0.800258081	1.71368519	0.00000
23:	2024-01-17 22:00:00	37.2	2.57411733	TRUE	-0.842501657	-1.001910427	1.93928567	0.00000
24:	2024-01-17 23:00:00	37.2	2.83585709	TRUE	-0.924691065	-1.180223341	2.30977400	0.00000
	Dates	lat	w	night	cosThzS	AlS	AzS	Bo0

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_nigth <- fSolI(sold = sold[1], sample = 'hour', keep.night = FALSE)
2 show(solI_nigth)
```

	Dates	lat	w	night	cosThzS	AlS	AzS	Bo0
	<POS>	<num>	<num>	<lgcl>	<num>	<num>	<num>	<num>
1:	2024-01-17 08:00:00	37.2	-1.09023003	FALSE	0.129741461	0.130108233	-0.99014548	183.39419
2:	2024-01-17 09:00:00	37.2	-0.82849151	FALSE	0.288889848	0.293067041	-0.80329847	408.35612
3:	2024-01-17 10:00:00	37.2	-0.56675290	FALSE	0.413747472	0.426566560	-0.58400587	584.84684
4:	2024-01-17 11:00:00	37.2	-0.30501420	FALSE	0.495809380	0.518766586	-0.32921922	700.84427
5:	2024-01-17 12:00:00	37.2	-0.04327541	FALSE	0.529485721	0.557994217	-0.04769723	748.44699
6:	2024-01-17 13:00:00	37.2	0.21846346	FALSE	0.512482515	0.538073327	0.23821864	724.41235
7:	2024-01-17 14:00:00	37.2	0.48020243	FALSE	0.445957919	0.462244212	0.50355560	630.37745
8:	2024-01-17 15:00:00	37.2	0.74194148	FALSE	0.334443348	0.341014503	0.73469016	472.74762
9:	2024-01-17 16:00:00	37.2	1.00368062	FALSE	0.185534810	0.186616094	0.93148844	262.26008
10:	2024-01-17 17:00:00	37.2	1.26541985	FALSE	0.009375501	0.009375638	1.10112996	13.25261

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

```
1 BTi <- fBTi(BTd, sample = 'hour')
2 solI_BTi <- fSolI(sold, BTi = BTi)
3 show(solI_BTi)
```

Index: <night>								
	Dates	lat	w	night	cosThzS	AlS	AzS	Bo0
	<POS>	<num>	<num>	<lgcl>	<num>	<num>	<num>	<num>
1:	2024-01-17 00:00:00	37.2	3.099050	TRUE	-0.9585523	-1.2818770	3.001577	0
2:	2024-01-17 01:00:00	37.2	-2.922397	TRUE	-0.9414074	-1.2267791	-2.494627	0
3:	2024-01-17 02:00:00	37.2	-2.660659	TRUE	-0.8747495	-1.0649186	-2.038624	0
4:	2024-01-17 03:00:00	37.2	-2.398921	TRUE	-0.7631191	-0.8681259	-1.779321	0
5:	2024-01-17 04:00:00	37.2	-2.137183	TRUE	-0.6141201	-0.6612706	-1.597015	0
---								
284:	2024-12-13 19:00:00	37.2	1.856445	TRUE	-0.4444110	-0.4605166	1.394524	0
285:	2024-12-13 20:00:00	37.2	2.118158	TRUE	-0.6191456	-0.6676542	1.539641	0
286:	2024-12-13 21:00:00	37.2	2.379871	TRUE	-0.7679298	-0.8756029	1.709361	0
287:	2024-12-13 22:00:00	37.2	2.641583	TRUE	-0.8806309	-1.0771921	1.946876	0
288:	2024-12-13 23:00:00	37.2	2.903296	TRUE	-0.9495736	-1.2518732	2.377338	0

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 show(solI_EoT)
```

Index: <night>

	Dates	lat	w	night	cosThzS	ALS	AzS	BoO
	<POS<	<num>	<num>	<lgcl>	<num>	<num>	<num>	<num>
1:	2024-01-17 00:00:00	37.2	3.099050	TRUE	-0.9585523	-1.2818770	3.001577	0
2:	2024-01-17 01:00:00	37.2	-2.922397	TRUE	-0.9414074	-1.2267791	-2.494627	0
3:	2024-01-17 02:00:00	37.2	-2.660659	TRUE	-0.8747495	-1.0649186	-2.038624	0
4:	2024-01-17 03:00:00	37.2	-2.398921	TRUE	-0.7631191	-0.8681259	-1.779321	0
5:	2024-01-17 04:00:00	37.2	-2.137183	TRUE	-0.6141201	-0.6612706	-1.597015	0
---								
284:	2024-12-13 19:00:00	37.2	1.856445	TRUE	-0.4444110	-0.4605166	1.394524	0
285:	2024-12-13 20:00:00	37.2	2.118158	TRUE	-0.6191456	-0.6676542	1.539641	0
286:	2024-12-13 21:00:00	37.2	2.379871	TRUE	-0.7679298	-0.8756029	1.709361	0
287:	2024-12-13 22:00:00	37.2	2.641583	TRUE	-0.8806309	-1.0771921	1.946876	0
288:	2024-12-13 23:00:00	37.2	2.903296	TRUE	-0.9495736	-1.2518732	2.377338	0

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.

```
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. : -0.404783	Min. :0.9675	Min. : -0.0614793	Min. : -1.900
1st Qu.:2024-04-07	1st Qu.: -0.256032	1st Qu.:0.9771	1st Qu.: -0.0289759	1st Qu.: -1.767
Median :2024-06-29	Median : -0.002305	Median :1.0007	Median : 0.0005052	Median : -1.569
Mean :2024-07-01	Mean : -0.001618	Mean :1.0009	Mean : 0.0008748	Mean : -1.569
3rd Qu.:2024-09-25	3rd Qu.: 0.251172	3rd Qu.:1.0249	3rd Qu.: 0.0204515	3rd Qu.: -1.370
Max. :2024-12-13	Max. : 0.402578	Max. :1.0340	Max. : 0.0689613	Max. : -1.240

BoOd

Min. : 4284
1st Qu.: 5841
Median : 8297
Mean : 8109
3rd Qu.:10416
Max. :11574

Intradaily values:

Dates	w	night	cosThzS
Min. :2024-01-17 00:00:00	Min. : -3.1393050	Mode :logical	Min. : -0.9700256
1st Qu.:2024-04-07 11:45:00	1st Qu.: -1.5692285	FALSE:145	1st Qu.: -0.5004531
Median :2024-06-29 11:30:00	Median : 0.0010871	TRUE :143	Median : 0.0062923
Mean :2024-07-01 15:30:00	Mean : 0.0009975		Mean : -0.0009523
3rd Qu.:2024-09-26 11:15:00	3rd Qu.: 1.5716412		3rd Qu.: 0.5007129
Max. :2024-12-13 23:00:00	Max. : 3.1413972		Max. : 0.9697262

ALS	AzS	BoO
Min. : -1.325336	Min. : -3.139169	Min. : 0.000
1st Qu.: -0.524130	1st Qu.: -1.570722	1st Qu.: 0.000
Median : 0.006292	Median : 0.003834	Median : 8.748
Mean : -0.001202	Mean : 0.001011	Mean : 337.752
3rd Qu.: 0.524433	3rd Qu.: 1.555342	3rd Qu.: 698.153
Max. : 1.324107	Max. : 3.141331	Max. :1284.718

## 4.2. Datos meteorológicos

Para el procesamiento de datos meteorológicos, **solar2** provee una serie de funciones<sup>4</sup> 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 4.3. Estas funciones son:

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

```
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 show(BD)
```

```
Object of class Meteo

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

Meteorological Data:
  Dates      G0d      Ta
Min. :2024-01-17 Min. :2179 Min. :10.00
1st Qu.:2024-04-07 1st Qu.:3322 1st Qu.:15.50
Median :2024-06-29 Median :4932 Median :17.70
Mean :2024-07-01 Mean :5022 Mean :19.22
3rd Qu.:2024-09-25 3rd Qu.:6998 3rd Qu.:21.98
Max. :2024-12-13 Max. :7919 Max. :29.90
```

- **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.

```
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/
   aranjuez.csv"
```

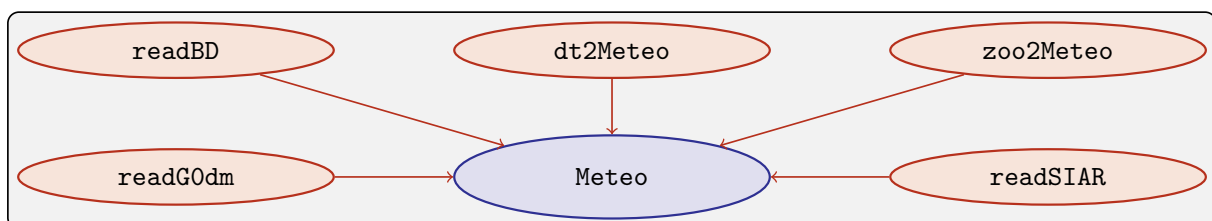


FIGURA 4.3: Los datos meteorológicos se pueden leer mediante las funciones **readG0dm**, **readBD**, **dt2Meteo**, **zoo2Meteo** y **readSIAR** las cuales procesan estos datos y los almacenan en un objeto de clase **Meteo**.

<sup>4</sup>Las funciones comentadas en este apartado, se recogen en la sección A.1.8

```

4 download.file(myURL, 'data/aranjuez.csv', quiet = TRUE)
5 Bdd <- readBdd(file = 'data/aranjuez.csv', lat = lat,
6               format = '%Y-%m-%d', header = TRUE,
7               fill = TRUE, dec = '.', sep = ',', dates.col = '',
8               ta.col = 'TempAvg', g0.col = 'Radiation', keep.cols = TRUE)
9 show(Bdd)

```

Object of class Meteo

Source of meteorological information: bd-data/aranjuez.csv

Latitude of source: 37.2 degrees

Meteorological Data:

Dates	G0	Ta	TempMin	TempMax	
Min. :2004-01-01	Min. : 0.277	Min. :-5.309	Min. :-12.980	Min. :-2.362	
1st Qu.:2005-12-29	1st Qu.: 9.370	1st Qu.: 7.692	1st Qu.: 1.515	1st Qu.:14.530	
Median :2008-01-09	Median :16.660	Median :13.810	Median : 7.170	Median :21.670	
Mean :2008-01-03	Mean :16.742	Mean :14.405	Mean : 6.888	Mean :22.531	
3rd Qu.:2010-01-02	3rd Qu.:24.650	3rd Qu.:21.615	3rd Qu.: 12.590	3rd Qu.:30.875	
Max. :2011-12-31	Max. :32.740	Max. :30.680	Max. : 22.710	Max. :41.910	
	NA's :13		NA's :4		
HumidAvg	HumidMax	WindAvg	WindMax	Rain	ET
Min. : 19.89	Min. : 35.88	Min. :0.251	Min. : 0.000	Min. : 0.000	Min. :0.000
1st Qu.: 47.04	1st Qu.: 81.60	1st Qu.:0.667	1st Qu.: 3.783	1st Qu.: 0.000	1st Qu.:1.168
Median : 62.58	Median : 90.90	Median :0.920	Median : 5.027	Median : 0.000	Median :2.758
Mean : 62.16	Mean : 87.22	Mean :1.174	Mean : 5.208	Mean : 1.094	Mean :3.091
3rd Qu.: 77.38	3rd Qu.: 94.90	3rd Qu.:1.431	3rd Qu.: 6.537	3rd Qu.: 0.200	3rd Qu.:4.926
Max. :100.00	Max. :100.00	Max. :8.260	Max. :10.000	Max. :49.730	Max. :8.564
	NA's :13	NA's :8	NA's :128	NA's :4	NA's :18

- **readBDi**: Procesa datos meteorológicos de tipo intradiarios.

```

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

```

Object of class Meteo

Source of meteorological information: bdI-data/NREL-Hawaii.csv

Latitude of source: 19 degrees

Meteorological Data:

Dates	G0	Ta	Direct Normal [W/m^2]
Min. :2010-01-11 06:32:00.00	Min. : 0.4769	Min. :13.42	Min. : 0.0
1st Qu.:2010-03-11 17:37:45.00	1st Qu.: 147.4328	1st Qu.:22.76	1st Qu.: 0.0
Median :2010-06-11 17:32:30.00	Median : 300.6510	Median :24.15	Median :270.3
Mean :2010-06-26 11:55:22.63	Mean : 370.5293	Mean :23.64	Mean :356.6
3rd Qu.:2010-09-11 17:34:15.00	3rd Qu.: 585.7402	3rd Qu.:25.24	3rd Qu.:715.2
Max. :2010-12-11 17:46:00.00	Max. :1172.3000	Max. :28.12	Max. :943.0
NA's :4660			
Diffuse Horizontal [W/m^2]			
Min. : 0.4769			
1st Qu.: 78.4636			
Median :152.9320			
Mean :171.7706			



```
3rd Qu.:246.3193
Max.    :586.3600
```

- **dt2Meteo**: Transforma un **data.table** o **data.frame** en un objeto de clase **Meteo**.

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

Object of class Meteo

Source of meteorological information: bd-data.frame

Latitude of source: 40 degrees

Meteorological Data:

Dates	G0d	TempMin	TempMax
Min. :2009-01-01 00:00:00.00	Min. : 325.6	Min. : -37.500	Min. : 1.41
1st Qu.:2009-04-08 12:00:00.00	1st Qu.: 2523.2	1st Qu.: 1.950	1st Qu.:14.41
Median :2009-07-07 00:00:00.00	Median : 4745.7	Median : 7.910	Median :23.16
Mean :2009-07-04 21:29:54.93	Mean : 4812.0	Mean : 5.323	Mean :22.59
3rd Qu.:2009-10-03 12:00:00.00	3rd Qu.: 7139.5	3rd Qu.: 15.105	3rd Qu.:31.06
Max. :2009-12-31 00:00:00.00	Max. :11253.9	Max. : 24.800	Max. :38.04

Ta

```
Min. : -23.049
1st Qu.: 7.008
Median : 12.055
Mean : 10.944
3rd Qu.: 19.472
Max. : 28.619
```

- **zoo2Meteo**: Transforma un objeto de clase **zoo**<sup>5</sup> en un objeto de clase **Meteo**.

```
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 :13	NA's :8

WindMax	Rain	Radiation	ET
Min. : 0.000	Min. : 0.000	Min. : 0.277	Min. :0.000

<sup>5</sup>Pese a que este proyecto trate de “desligarse” del paquete **zoo**, sigue siendo un paquete muy extendido. Por lo que es interesante tener una función así para que los usuarios tengan una mayor flexibilidad.

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 SIAR y transformarlo en un objeto de clase **Meteo**.

```

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 show(bd_SIAR)
```

Object of class **Meteo**

Source of meteorological information: prom-https://servicio.mapama.gob.es  
 -Estaciones: Center: Finca experimental(M01), Arganda(M02), San Martín de la Vega(M05)  
 Latitude of source: 40.4 degrees

Meteorological Data:

Dates	G0d	Ta	TempMin	TempMax
Min. :2023-09-18 00:00:00	Min. :1860	Min. : 5.318	Min. : -4.6513	Min. :15.34
1st Qu.:2023-12-06 18:00:00	1st Qu.:2744	1st Qu.: 9.857	1st Qu.: -2.1466	1st Qu.:21.12
Median :2024-02-29 00:00:00	Median :4052	Median :14.890	Median : 0.3663	Median :31.01
Mean :2024-03-01 04:00:00	Mean :4502	Mean :15.307	Mean : 2.4225	Mean :29.41
3rd Qu.:2024-05-21 12:00:00	3rd Qu.:6549	3rd Qu.:20.047	3rd Qu.: 7.1506	3rd Qu.:35.47
Max. :2024-08-18 00:00:00	Max. :7608	Max. :27.069	Max. :12.6082	Max. :40.70

Esta función tiene dos argumentos importantes:

- **tipo**: La API SIAR<sup>6</sup> 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 proximas 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 (IDW) 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.

### 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,

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

**solar2** cuenta con la función **calcG0** [A.1.2] la cual mediante las funciones **fCompD** [A.3.4] y **fCompI** [A.3.5] procesan los objetos de clase **Sol** y clase **Meteo** para dar un objeto de tipo **G0**.

Como se puede ver en la figura 4.4, **calcG0** funciona gracias a las siguientes funciones:

- **fCompD**: La cual calcula todas las componentes de la irradiación diaria en una superficie horizontal mediante regresiones entre los parámetros del índice de claridad y la fracción difusa. Para ello se pueden usar varias correlaciones dependiendo del tipo de datos:

- Mensuales:

```
1 lat <- 37.2
2 BTd <- fBTd(mode = 'prom')
3 sold <- fSold(lat, BTd)
4 G0d <- c
  (2.766,3.491,4.494,5.912,6.989,7.742,7.919,7.027,5.369,3.562,2.814,2.179)
  * 1000
5 compD_page <- fCompD(sol = sold, G0d = G0d, corr = "Page")
6 compD_page
```

Key: <Dates>						
Dates	Fd	Kt	G0d	D0d	B0d	
<POS>	<num>	<num>	<num>	<num>	<num>	<num>
1: 2024-01-17	0.3404548	0.5836683	2766	941.698	1824.302	
2: 2024-02-14	0.3572461	0.5688088	3491	1247.146	2243.854	
3: 2024-03-15	0.3719989	0.5557532	4494	1671.763	2822.237	
4: 2024-04-15	0.3266485	0.5958862	5912	1931.146	3980.854	
5: 2024-05-15	0.2895069	0.6287549	6989	2023.364	4965.636	
6: 2024-06-10	0.2441221	0.6689185	7742	1889.994	5852.006	
7: 2024-07-18	0.2050844	0.7034651	7919	1624.064	6294.936	
8: 2024-08-18	0.2202349	0.6900576	7027	1547.591	5479.409	
9: 2024-09-18	0.2869638	0.6310055	5369	1540.708	3828.292	
10: 2024-10-19	0.3858825	0.5434669	3562	1374.513	2187.487	
11: 2024-11-18	0.3578392	0.5682839	2814	1006.959	1807.041	
12: 2024-12-13	0.4253038	0.5085807	2179	926.737	1252.263	

```
1 compD_lj <- fCompD(sol = sold, G0d = G0d, corr = "LJ")
2 compD_lj
```

Key: <Dates>						
Dates	Fd	Kt	G0d	D0d	B0d	
<POS>	<num>	<num>	<num>	<num>	<num>	<num>
1: 2024-01-17	0.3058193	0.5836683	2766	845.8961	1920.104	

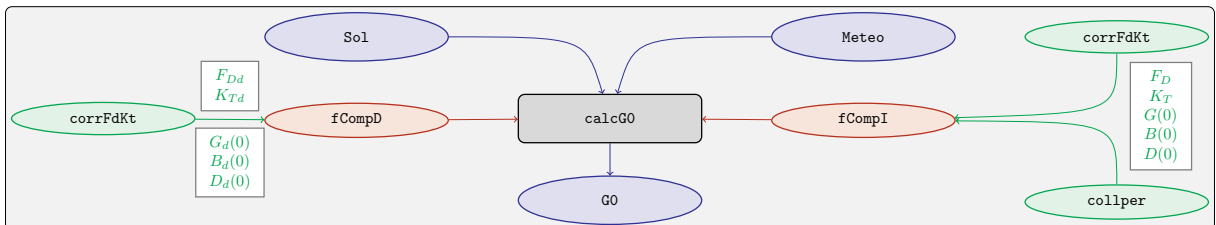


FIGURA 4.4: Cálculo de la radiación incidente en el plano horizontal mediante la función **calcG0**, la cual procesa un objeto clase **Sol** y otro clase **Meteo** mediante las funciones **fCompD** y **fCompI** resultando en un objeto clase **G0**.

#### 4. DESARROLLO DEL CÓDIGO

```

2: 2024-02-14 0.3169470 0.5688088 3491 1106.4621 2384.538
3: 2024-03-15 0.3268047 0.5557532 4494 1468.6603 3025.340
4: 2024-04-15 0.2967018 0.5958862 5912 1754.1011 4157.899
5: 2024-05-15 0.2720419 0.6287549 6989 1901.3006 5087.699
6: 2024-06-10 0.2408700 0.6689185 7742 1864.8154 5877.185
7: 2024-07-18 0.2152460 0.7034651 7919 1704.5331 6214.467
8: 2024-08-18 0.2236251 0.6900576 7027 1571.4138 5455.586
9: 2024-09-18 0.2703347 0.6310055 5369 1451.4268 3917.573
10: 2024-10-19 0.3361895 0.5434669 3562 1197.5071 2364.493
11: 2024-11-18 0.3173415 0.5682839 2814 892.9990 1921.001
12: 2024-12-13 0.3637158 0.5085807 2179 792.5367 1386.463

```

- Diarios:

```

1 G0d <- readSIAR(Lat = 40.40596822621351, Lon = -3.70038308516172,
2                 inicio = '2024-07-15', final = '2024-08-01',
3                 tipo = 'Diarios', n_est = 3)
4 sol <- calcSol(lat, BTd = indexD(G0d))
5 compD_cpr <- fCompD(sol = sol, G0d = G0d, corr = "CPR")
6 compD_cpr

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
  <POS< <num> <num> <num> <num> <num>
1: 2024-07-15 0.2833125 0.6798139 7697.945 2180.924 5517.021
2: 2024-07-16 0.2597185 0.7000272 7911.858 2054.856 5857.002
3: 2024-07-17 0.2815044 0.6812283 7684.293 2163.163 5521.131
4: 2024-07-18 0.6627754 0.4674993 5262.702 3487.989 1774.713
5: 2024-07-19 0.2595844 0.7001561 7865.166 2041.675 5823.491
6: 2024-07-20 0.2594075 0.7003266 7849.961 2036.339 5813.622
7: 2024-07-21 0.2315068 0.7365959 8237.938 1907.138 6330.799
8: 2024-07-22 0.2269337 0.7493438 8361.056 1897.406 6463.650
9: 2024-07-23 0.2451723 0.7156288 7965.753 1952.982 6012.771
10: 2024-07-24 0.2620008 0.6978638 7748.845 2030.204 5718.641
11: 2024-07-25 0.2746548 0.6867564 7606.140 2089.063 5517.077
12: 2024-07-26 0.3320728 0.6462270 7138.548 2370.518 4768.030
13: 2024-07-27 0.3186769 0.6547900 7213.697 2298.839 4914.858
14: 2024-07-28 0.2767163 0.6850625 7526.355 2082.665 5443.689
15: 2024-07-29 0.6566999 0.4709412 5159.260 3388.086 1771.174
16: 2024-07-30 0.3185533 0.6548709 7153.359 2278.726 4874.633
17: 2024-07-31 0.2503814 0.7096003 7728.034 1934.956 5793.078
18: 2024-08-01 0.2428514 0.7185406 7801.435 1894.589 5906.846

```

```

1 compD_ekdd <- fCompD(sol = sol, G0d = G0d, corr = 'EKDd')
2 compD_ekdd

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
  <POS< <num> <num> <num> <num> <num>
1: 2024-07-15 1 0.6798139 7697.945 7697.945 0
2: 2024-07-16 1 0.7000272 7911.858 7911.858 0
3: 2024-07-17 1 0.6812283 7684.293 7684.293 0
4: 2024-07-18 1 0.4674993 5262.702 5262.702 0
5: 2024-07-19 1 0.7001561 7865.166 7865.166 0
6: 2024-07-20 1 0.7003266 7849.961 7849.961 0
7: 2024-07-21 1 0.7365959 8237.938 8237.938 0
8: 2024-07-22 1 0.7493438 8361.056 8361.056 0
9: 2024-07-23 1 0.7156288 7965.753 7965.753 0
10: 2024-07-24 1 0.6978638 7748.845 7748.845 0
11: 2024-07-25 1 0.6867564 7606.140 7606.140 0
12: 2024-07-26 1 0.6462270 7138.548 7138.548 0
13: 2024-07-27 1 0.6547900 7213.697 7213.697 0
14: 2024-07-28 1 0.6850625 7526.355 7526.355 0

```

```

15: 2024-07-29      1 0.4709412 5159.260 5159.260      0
16: 2024-07-30      1 0.6548709 7153.359 7153.359      0
17: 2024-07-31      1 0.7096003 7728.034 7728.034      0
18: 2024-08-01      1 0.7185406 7801.435 7801.435      0

```

```

1 compD_climeddd <- fCompD(sol = sol, G0d = G0d, corr = 'CLIMEDd')
2 compD_climeddd

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
  <POS<  <num>  <num>  <num>  <num>  <num>
1: 2024-07-15 0.2724591 0.6798139 7697.945 2097.375 5600.570
2: 2024-07-16 0.2455880 0.7000272 7911.858 1943.057 5968.801
3: 2024-07-17 0.2705287 0.6812283 7684.293 2078.822 5605.472
4: 2024-07-18 0.6086148 0.4674993 5262.702 3202.958 2059.744
5: 2024-07-19 0.2454217 0.7001561 7865.166 1930.282 5934.884
6: 2024-07-20 0.2452020 0.7003266 7849.961 1924.826 5925.135
7: 2024-07-21 0.2013208 0.7365959 8237.938 1658.468 6579.470
8: 2024-07-22 0.1873678 0.7493438 8361.056 1566.592 6794.463
9: 2024-07-23 0.2259736 0.7156288 7965.753 1800.050 6165.703
10: 2024-07-24 0.2483878 0.6978638 7748.845 1924.718 5824.126
11: 2024-07-25 0.2630540 0.6867564 7606.140 2000.826 5605.314
12: 2024-07-26 0.3202837 0.6462270 7138.548 2286.361 4852.187
13: 2024-07-27 0.3077503 0.6547900 7213.697 2220.018 4993.679
14: 2024-07-28 0.2653324 0.6850625 7526.355 1996.986 5529.369
15: 2024-07-29 0.6029930 0.4709412 5159.260 3110.998 2048.263
16: 2024-07-30 0.3076331 0.6548709 7153.359 2200.610 4952.749
17: 2024-07-31 0.2334298 0.7096003 7728.034 1803.954 5924.080
18: 2024-08-01 0.2224291 0.7185406 7801.435 1735.266 6066.168

```

También, se puede aportar una función de correlación propia.

```

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 = G0d, corr = 'user', f = f_corrd)
10 compD_user

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
  <POS<  <num>  <num>  <num>  <num>  <num>
1: 2024-07-15 0.2724591 0.6798139 7697.945 2097.375 5600.570
2: 2024-07-16 0.2455880 0.7000272 7911.858 1943.057 5968.801
3: 2024-07-17 0.2705287 0.6812283 7684.293 2078.822 5605.472
4: 2024-07-18 0.6086148 0.4674993 5262.702 3202.958 2059.744
5: 2024-07-19 0.2454217 0.7001561 7865.166 1930.282 5934.884
6: 2024-07-20 0.2452020 0.7003266 7849.961 1924.826 5925.135
7: 2024-07-21 0.2013208 0.7365959 8237.938 1658.468 6579.470
8: 2024-07-22 0.1873678 0.7493438 8361.056 1566.592 6794.463
9: 2024-07-23 0.2259736 0.7156288 7965.753 1800.050 6165.703
10: 2024-07-24 0.2483878 0.6978638 7748.845 1924.718 5824.126
11: 2024-07-25 0.2630540 0.6867564 7606.140 2000.826 5605.314
12: 2024-07-26 0.3202837 0.6462270 7138.548 2286.361 4852.187
13: 2024-07-27 0.3077503 0.6547900 7213.697 2220.018 4993.679
14: 2024-07-28 0.2653324 0.6850625 7526.355 1996.986 5529.369
15: 2024-07-29 0.6029930 0.4709412 5159.260 3110.998 2048.263
16: 2024-07-30 0.3076331 0.6548709 7153.359 2200.610 4952.749
17: 2024-07-31 0.2334298 0.7096003 7728.034 1803.954 5924.080
18: 2024-08-01 0.2224291 0.7185406 7801.435 1735.266 6066.168

```

Por último, si **G0d** ya contiene todos los componentes, se puede especifica 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: 2024-07-15 0.2724591 0.6798139 7697.945 2097.375 5600.570
2: 2024-07-16 0.2455880 0.7000272 7911.858 1943.057 5968.801
3: 2024-07-17 0.2705287 0.6812283 7684.293 2078.822 5605.472
4: 2024-07-18 0.6086148 0.4674993 5262.702 3202.958 2059.744
5: 2024-07-19 0.2454217 0.7001561 7865.166 1930.282 5934.884
6: 2024-07-20 0.2452020 0.7003266 7849.961 1924.826 5925.135
7: 2024-07-21 0.2013208 0.7365959 8237.938 1658.468 6579.470
8: 2024-07-22 0.1873678 0.7493438 8361.056 1566.592 6794.463
9: 2024-07-23 0.2259736 0.7156288 7965.753 1800.050 6165.703
10: 2024-07-24 0.2483878 0.6978638 7748.845 1924.718 5824.126
11: 2024-07-25 0.2630540 0.6867564 7606.140 2000.826 5605.314
12: 2024-07-26 0.3202837 0.6462270 7138.548 2286.361 4852.187
13: 2024-07-27 0.3077503 0.6547900 7213.697 2220.018 4993.679
14: 2024-07-28 0.2653324 0.6850625 7526.355 1996.986 5529.369
15: 2024-07-29 0.6029930 0.4709412 5159.260 3110.998 2048.263
16: 2024-07-30 0.3076331 0.6548709 7153.359 2200.610 4952.749
17: 2024-07-31 0.2334298 0.7096003 7728.034 1803.954 5924.080
18: 2024-08-01 0.2224291 0.7185406 7801.435 1735.266 6066.168
```

- **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 sol <- calcSol(lat = 37.2, BTd = fBTd(mode = 'prom'),
2           sample = 'hour', keep.night = FALSE)
3 G0d <- c(2.766,3.491,4.494,5.912,6.989,7.742,7.919,
4         7.027,5.369,3.562,2.814,2.179) * 1000
5 compD <- fCompD(sol = sol, G0d = G0d, corr = 'CPR')
6 compI <- fCompI(sol = sol, compD = compD)
7 show(compI)
```

```
Key: <Dates>
      Dates      Fd      Kt      G0      D0      B0
   <POS<   <num>   <num>   <num>   <num>   <num>
1: 2024-01-17 08:00:00 0.5656199 0.4583592  84.06042  47.54625  36.40399
2: 2024-01-17 09:00:00 0.4912826 0.5277148 215.49558 105.86922 109.51548
3: 2024-01-17 10:00:00 0.4453619 0.5821268 340.45500 151.62569 188.82159
4: 2024-01-17 11:00:00 0.4195854 0.6178887 433.04376 181.69885 251.45464
5: 2024-01-17 12:00:00 0.4098508 0.6325646 473.44106 194.04019 279.57020
---
141: 2024-12-13 12:00:00 0.5437347 0.5488870 382.71443 208.09513 174.85828
142: 2024-12-13 13:00:00 0.5556284 0.5371376 352.10710 195.64071 156.62669
143: 2024-12-13 14:00:00 0.5893861 0.5063725 276.60890 163.02945 113.57257
144: 2024-12-13 15:00:00 0.6506594 0.4586869 172.87432 112.48231  60.23704
145: 2024-12-13 16:00:00 0.7511394 0.3973283  63.15968  47.44173  15.57107
```

- **G0I**: Este argumento recibe datos de irradiancia, para después, poder aplicar las correcciones indicadas en el argumento **corr**.

```

1 GOI <- compI$GO
2 compI_ekdh <- fCompI(sol = sol, GOI = GOI, corr = 'EKDh')
3 show(compI_ekdh)

```

```

Key: <Dates>
      Dates      Fd      Kt      GO      DO      BO
      <POS<      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 0.7417600 0.4583592 84.06042 62.35265 21.70776
2: 2024-01-17 09:00:00 0.6000150 0.5277148 215.49558 129.30057 86.19500
3: 2024-01-17 10:00:00 0.4791716 0.5821268 340.45500 163.13636 177.31865
4: 2024-01-17 11:00:00 0.4004462 0.6178887 433.04376 173.41074 259.63302
5: 2024-01-17 12:00:00 0.3692679 0.6325646 473.44106 174.82659 298.61447
---
141: 2024-12-13 12:00:00 0.5533972 0.5488870 382.71443 211.79307 170.92135
142: 2024-12-13 13:00:00 0.5793829 0.5371376 352.10710 204.00484 148.10226
143: 2024-12-13 14:00:00 0.6457949 0.5063725 276.60890 178.63262 97.97628
144: 2024-12-13 15:00:00 0.7411461 0.4586869 172.87432 128.12512 44.74920
145: 2024-12-13 16:00:00 0.8439123 0.3973283 63.15968 53.30123 9.85845

```

```

1 compI_brl <- fCompI(sol = sol, GOI = GOI, corr = 'BRL')
2 show(compI_brl)

```

```

Key: <Dates>
      Dates      Fd      Kt      GO      DO      BO
      <POS<      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 0.6689300 0.4583592 84.06042 56.23053 27.82988
2: 2024-01-17 09:00:00 0.5775367 0.5277148 215.49558 124.45660 91.03897
3: 2024-01-17 10:00:00 0.4826595 0.5821268 340.45500 164.32384 176.13116
4: 2024-01-17 11:00:00 0.4204896 0.6178887 433.04376 182.09040 250.95337
5: 2024-01-17 12:00:00 0.3948666 0.6325646 473.44106 186.94604 286.49502
---
141: 2024-12-13 12:00:00 0.5872522 0.5488870 382.71443 224.74989 157.96454
142: 2024-12-13 13:00:00 0.6048894 0.5371376 352.10710 212.98583 139.12126
143: 2024-12-13 14:00:00 0.6521416 0.5063725 276.60890 180.38818 96.22073
144: 2024-12-13 15:00:00 0.7207149 0.4586869 172.87432 124.59311 48.28121
145: 2024-12-13 16:00:00 0.7818945 0.3973283 63.15968 49.38421 13.77547

```

```

1 compI_climedh <- fCompI(sol = sol, GOI = GOI, corr = 'CLIMEDh')
2 show(compI_climedh)

```

```

Key: <Dates>
      Dates      Fd      Kt      GO      DO      BO
      <POS<      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 0.7093252 0.4583592 84.06042 59.62617 24.43424
2: 2024-01-17 09:00:00 0.5818534 0.5277148 215.49558 125.38683 90.10875
3: 2024-01-17 10:00:00 0.4782729 0.5821268 340.45500 162.83039 177.62462
4: 2024-01-17 11:00:00 0.4110389 0.6178887 433.04376 177.99784 255.04592
5: 2024-01-17 12:00:00 0.3840268 0.6325646 473.44106 181.81406 291.62701
---
141: 2024-12-13 12:00:00 0.5416063 0.5488870 382.71443 207.28055 175.43387
142: 2024-12-13 13:00:00 0.5639749 0.5371376 352.10710 198.57956 153.52754
143: 2024-12-13 14:00:00 0.6220088 0.5063725 276.60890 172.05317 104.55573
144: 2024-12-13 15:00:00 0.7087489 0.4586869 172.87432 122.52448 50.34984
145: 2024-12-13 16:00:00 0.8099691 0.3973283 63.15968 51.15739 12.00229

```

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      GO      DO      BO
      <POS<      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 0.7093252 0.4583592 84.06042 59.62617 24.43424
2: 2024-01-17 09:00:00 0.5818534 0.5277148 215.49558 125.38683 90.10875
3: 2024-01-17 10:00:00 0.4782729 0.5821268 340.45500 162.83039 177.62462
4: 2024-01-17 11:00:00 0.4110389 0.6178887 433.04376 177.99784 255.04592
5: 2024-01-17 12:00:00 0.3840268 0.6325646 473.44106 181.81406 291.62701
---
141: 2024-12-13 12:00:00 0.5416063 0.5488870 382.71443 207.28055 175.43387
142: 2024-12-13 13:00:00 0.5639749 0.5371376 352.10710 198.57956 153.52754
143: 2024-12-13 14:00:00 0.6220088 0.5063725 276.60890 172.05317 104.55573
144: 2024-12-13 15:00:00 0.7087489 0.4586869 172.87432 122.52448 50.34984
145: 2024-12-13 16:00:00 0.8099691 0.3973283 63.15968 51.15739 12.00229

```

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      GO      DO      BO
      <POS<      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 0.7093252 0.4583592 84.06042 59.62617 24.43424
2: 2024-01-17 09:00:00 0.5818534 0.5277148 215.49558 125.38683 90.10875
3: 2024-01-17 10:00:00 0.4782729 0.5821268 340.45500 162.83039 177.62462
4: 2024-01-17 11:00:00 0.4110389 0.6178887 433.04376 177.99784 255.04592
5: 2024-01-17 12:00:00 0.3840268 0.6325646 473.44106 181.81406 291.62701
---
141: 2024-12-13 12:00:00 0.5416063 0.5488870 382.71443 207.28055 175.43387
142: 2024-12-13 13:00:00 0.5639749 0.5371376 352.10710 198.57956 153.52754
143: 2024-12-13 14:00:00 0.6220088 0.5063725 276.60890 172.05317 104.55573
144: 2024-12-13 15:00:00 0.7087489 0.4586869 172.87432 122.52448 50.34984
145: 2024-12-13 16:00:00 0.8099691 0.3973283 63.15968 51.15739 12.00229

```

Por último, esta función incluye un argumento extra, **filterGO** 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 4.4, convergen en la función constructora **calcGO**, dando como resultado un objeto de clase **GO**. 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 avarages:
  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
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  G0

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 1.146197 1.619803
2: Feb. 2024 3.491 1.509201 1.981799
3: Mar. 2024 4.494 1.995879 2.498121
4: Apr. 2024 5.912 2.259830 3.652170
5: May. 2024 6.989 2.473548 4.515452
6: Jun. 2024 7.742 2.470506 5.271494
7: Jul. 2024 7.919 2.396801 5.522199
8: Aug. 2024 7.027 2.107768 4.919232
9: Sep. 2024 5.369 1.964577 3.404423
10: Oct. 2024 3.562 1.648904 1.913096
11: Nov. 2024 2.814 1.222759 1.591241
12: Dec. 2024 2.179 1.042617 1.136383

Yearly values:

```

```
Key: <Dates>
  Dates      G0d      D0d      B0d
  <int>     <num>     <num>     <num>
1:  2024 1839.365 678.4601 1160.905
```

#### ■ Diarios.

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

```
Object of class  G0

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 1.146197 1.619803
2: Feb. 2024 3.491 1.509201 1.981799
3: Mar. 2024 4.494 1.995879 2.498121
4: Apr. 2024 5.912 2.259830 3.652170
5: May. 2024 6.989 2.473548 4.515452
6: Jun. 2024 7.742 2.470506 5.271494
7: Jul. 2024 7.919 2.396801 5.522199
8: Aug. 2024 7.027 2.107768 4.919232
9: Sep. 2024 5.369 1.964577 3.404423
10: Oct. 2024 3.562 1.648904 1.913096
11: Nov. 2024 2.814 1.222759 1.591241
12: Dec. 2024 2.179 1.042617 1.136383

Yearly values:
Key: <Dates>
  Dates      G0d      D0d      B0d
  <int>     <num>     <num>     <num>
1:  2024 1839.365 678.4601 1160.905
```

#### ■ Intradiarios

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

```
Object of class  G0

Source of meteorological information: bdI-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 2.766000 1.159456 1.606544
2: Feb. 2024 3.491000 1.534494 1.956506
3: Mar. 2024 4.494000 2.098539 2.395461
4: Apr. 2024 5.912000 2.239783 3.672217
5: May. 2024 6.866824 2.308807 4.558017
6: Jun. 2024 7.742000 2.241544 5.500456
7: Jul. 2024 7.919000 1.881773 6.037227
8: Aug. 2024 7.027000 2.017100 5.009900
9: Sep. 2024 5.369000 1.973292 3.395708
```

```

10: Oct. 2024 3.562000 1.640782 1.921218
11: Nov. 2024 2.814000 1.237715 1.576285
12: Dec. 2024 2.179000 1.056340 1.122660

```

Yearly values:

Key: <Dates>

Dates	G0d	D0d	B0d
<int>	<num>	<num>	<num>
1: 2024	1835.578	652.3171	1183.26

## 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** [A.1.3] 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 4.5, **calcGef** funciona gracias a las siguientes funciones:

- **fTheta**: la cual, partiendo del ángulo de inclinación ( $\beta$ ) y la orientación ( $\alpha$ ), calcula el ángulo de inclinación en cada instante ( $\beta$ ), el ángulo azimutal ( $\psi_s$ ) 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 = 'prom')[6]
2 sol <- calcSol(lat, BTd = BTd, keep.night = FALSE)
3 beta <- lat - 10
4 alfa <- 0
5 angGen_fixed <- fTheta(sol = sol, beta = beta, alfa = alfa,
6                       modeTrk = 'fixed')
7 show(angGen_fixed)

```

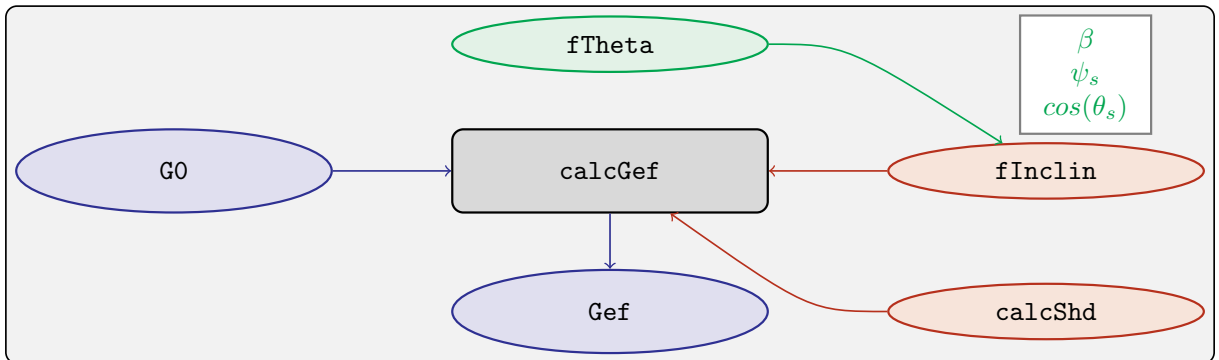


FIGURA 4.5: Cálculo de la radiación efectiva incidente en el plano del generador mediante la función **calcGef**, la cual emplea la función **fInclin** para el computo de las componentes efectivas, la función **fTheta** que provee a la función anterior los ángulos necesarios para su computo y la función **calcShd** que reprocesa el objeto de clase **Gef** resultante, añadiéndole el efecto de las sombras producidas entre módulos.

#### 4. DESARROLLO DEL CÓDIGO

	Dates	Beta	Alfa	cosTheta
	<POS>	<num>	<num>	<num>
1:	2024-06-10 05:00:00	0.4747296	0	0.00000000
2:	2024-06-10 06:00:00	0.4747296	0	0.06990810
3:	2024-06-10 07:00:00	0.4747296	0	0.30432148
4:	2024-06-10 08:00:00	0.4747296	0	0.52263672
5:	2024-06-10 09:00:00	0.4747296	0	0.70998013
6:	2024-06-10 10:00:00	0.4747296	0	0.85358815
7:	2024-06-10 11:00:00	0.4747296	0	0.94367686
8:	2024-06-10 12:00:00	0.4747296	0	0.97410861
9:	2024-06-10 13:00:00	0.4747296	0	0.94281011
10:	2024-06-10 14:00:00	0.4747296	0	0.85191372
11:	2024-06-10 15:00:00	0.4747296	0	0.70761218
12:	2024-06-10 16:00:00	0.4747296	0	0.51973665
13:	2024-06-10 17:00:00	0.4747296	0	0.30108697
14:	2024-06-10 18:00:00	0.4747296	0	0.06655958
15:	2024-06-10 19:00:00	0.4747296	0	0.00000000

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

```
1 angGen_two <- fTheta(sol = sol, beta = beta, alfa = alfa,  
2                       modeTrk = 'two')  
3 show(angGen_two)
```

	Dates	Beta	Alfa	cosTheta
	<POS>	<num>	<num>	<num>
1:	2024-06-10 05:00:00	1.5220852	-2.043678875	1
2:	2024-06-10 06:00:00	1.3300857	-1.896688029	1
3:	2024-06-10 07:00:00	1.1285281	-1.756655282	1
4:	2024-06-10 08:00:00	0.9215732	-1.612213267	1
5:	2024-06-10 09:00:00	0.7134716	-1.445120762	1
6:	2024-06-10 10:00:00	0.5110180	-1.215351693	1
7:	2024-06-10 11:00:00	0.3328578	-0.809087856	1
8:	2024-06-10 12:00:00	0.2466893	0.006963841	1
9:	2024-06-10 13:00:00	0.3349967	0.817155564	1
10:	2024-06-10 14:00:00	0.5137803	1.219398208	1
11:	2024-06-10 15:00:00	0.7163931	1.447776194	1
12:	2024-06-10 16:00:00	0.9245147	1.614353339	1
13:	2024-06-10 17:00:00	1.1314208	1.758631827	1
14:	2024-06-10 18:00:00	1.3328735	1.898691776	1
15:	2024-06-10 19:00:00	1.5247042	2.045849315	1

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

```
1 angGen_horiz <- fTheta(sol = sol, beta = beta, alfa = alfa,  
2                      modeTrk = 'horiz')  
3 show(angGen_horiz)
```

	Dates	Beta	Alfa	cosTheta
	<POS>	<num>	<num>	<num>
1:	2024-06-10 05:00:00	1.516091993	-1.570796	0.8905353
2:	2024-06-10 06:00:00	1.317263961	-1.570796	0.9504350
3:	2024-06-10 07:00:00	1.121771495	-1.570796	0.9859551
4:	2024-06-10 08:00:00	0.921160041	-1.570796	0.9994560
5:	2024-06-10 09:00:00	0.709555740	-1.570796	0.9966296
6:	2024-06-10 10:00:00	0.483954771	-1.570796	0.9854098
7:	2024-06-10 11:00:00	0.245151627	-1.570796	0.9742418
8:	2024-06-10 12:00:00	0.001753607	1.570796	0.9697277
9:	2024-06-10 13:00:00	0.248597042	1.570796	0.9743648
10:	2024-06-10 14:00:00	0.487239436	1.570796	0.9855868
11:	2024-06-10 15:00:00	0.712638107	1.570796	0.9967482

```

12: 2024-06-10 16:00:00 0.924058412 1.570796 0.9993956
13: 2024-06-10 17:00:00 1.124550569 1.570796 0.9856166
14: 2024-06-10 18:00:00 1.320024608 1.570796 0.9497600
15: 2024-06-10 19:00:00 1.518974473 1.570796 0.8895182

```

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, alfa = alfa,
4                     modeTrk = 'horiz', BT = TRUE,
5                     struct = struct, dist = distances)
6 show(angGen_BT)

```

	Dates <POS>	Beta <num>	Alfa <num>	cosTheta <num>
1:	2024-06-10 05:00:00	0.054868903	-1.570796	0.09738369
2:	2024-06-10 06:00:00	0.271972628	-1.570796	0.47678565
3:	2024-06-10 07:00:00	0.602487004	-1.570796	0.85598103
4:	2024-06-10 08:00:00	0.921160041	-1.570796	0.99945597
5:	2024-06-10 09:00:00	0.709555740	-1.570796	0.99662956
6:	2024-06-10 10:00:00	0.483954771	-1.570796	0.98540983
7:	2024-06-10 11:00:00	0.245151627	-1.570796	0.97424175
8:	2024-06-10 12:00:00	0.001753607	1.570796	0.96972767
9:	2024-06-10 13:00:00	0.248597042	1.570796	0.97436477
10:	2024-06-10 14:00:00	0.487239436	1.570796	0.98558683
11:	2024-06-10 15:00:00	0.712638107	1.570796	0.99674816
12:	2024-06-10 16:00:00	0.924058412	1.570796	0.99939563
13:	2024-06-10 17:00:00	0.595256963	1.570796	0.85074877
14:	2024-06-10 18:00:00	0.268563625	1.570796	0.47136897
15:	2024-06-10 19:00:00	0.051961679	1.570796	0.09215170

- **fInclin**: la cual, partiendo del resultado de **fTheta** y de un objeto de clase **GO**, 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 basandose en los valores de la tabla 3.2 calcula la irradiancia efectiva. Por defecto tiene valor 2 (grado de suciedad bajo).

```

1 compI <- calcGO(lat, dataRad = prom, keep.night = FALSE)
2 sol <- calcSol(lat, BTi = indexI(compI))
3 angGen <- fTheta(sol = sol, beta = beta, alfa = alfa)
4 inclin_limpio <- fInclin(compI = compI, angGen = angGen, iS = 1)
5 show(inclin_limpio)

```

	Dates <POS>	Bo <num>	Bn <num>	G <num>	D <num>	Di <num>	Dc <num>	B <num>	R <num>
1:	2024-01-17 08:00:00	514.5612	365.8727	186.4590	52.34286	25.82073	26.52212	133.18653	0.9295706
2:	2024-01-17 09:00:00	792.6980	464.2106	366.6704	103.96230	52.12242	51.83988	260.32510	2.3830282
3:	2024-01-17 10:00:00	1010.9063	541.3602	536.6247	145.69981	68.60264	77.09717	387.15997	3.7648749
4:	2024-01-17 11:00:00	1154.3223	592.0663	662.0048	173.72247	77.44190	96.28057	483.49354	4.7887550
5:	2024-01-17 12:00:00	1213.1770	612.8750	716.5974	185.35767	80.61172	104.74595	526.00427	5.2354830
---									
141:	2024-12-13 12:00:00	1181.1554	470.2512	578.4583	180.82966	95.85462	84.97504	393.39650	4.2321949

#### 4. DESARROLLO DEL CÓDIGO

```

142: 2024-12-13 13:00:00 1129.5610 453.5904 536.8668 170.08970 91.70559 78.38411 362.88341 3.8937280
143: 2024-12-13 14:00:00 994.4636 409.9651 434.0673 142.25355 79.88147 62.37208 288.75488 3.0588416
144: 2024-12-13 15:00:00 785.0640 342.3463 292.1950 99.92831 58.81096 41.11735 190.35496 1.9117069
145: 2024-12-13 16:00:00 515.6229 255.3390 140.8937 46.94651 26.80445 20.14206 93.24874 0.6984426
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 0.115032290 0.05043622 0.2503398 24.51843 23.47122 166.5523 47.98966 117.86578 0.6968621
2: 0.034235799 0.05043622 0.2503398 49.49356 50.06510 352.7578 99.55866 251.41266 1.7864615
3: 0.012139104 0.05043622 0.2503398 65.14258 76.16128 526.5864 141.30386 382.46020 2.8223770
4: 0.005426675 0.05043622 0.2503398 73.53602 95.75809 653.7538 169.29411 480.86978 3.5899392
5: 0.003640433 0.05043622 0.2503398 76.54597 104.36463 708.9248 180.91060 524.08939 3.9248333
---
141: 0.004516349 0.05043622 0.2503398 91.02007 84.59127 570.4038 175.61134 391.61978 3.1727082
142: 0.006269898 0.05043622 0.2503398 87.08031 77.89265 528.5001 164.97296 360.60816 2.9189730
143: 0.013120704 0.05043622 0.2503398 75.85255 61.55372 424.6656 137.40626 284.96622 2.2930919
144: 0.035287438 0.05043622 0.2503398 55.84476 39.66642 280.5821 95.51118 183.63782 1.4331306
145: 0.114223038 0.05043622 0.2503398 25.45254 17.84137 126.4151 43.29391 82.59758 0.5235947

```

```

1 inclin_sucio <- fInclin(compI = compI, angGen = angGen, iS = 4)
2 show(inclin_sucio)

```

```

      Dates      Bo      Bn      G      D      Di      Dc      B      R
      <POS<      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 514.5612 365.8727 186.4590 52.34286 25.82073 26.52212 133.18653 0.9295706
2: 2024-01-17 09:00:00 792.6980 464.2106 366.6704 103.96230 52.12242 51.83988 260.32510 2.3830282
3: 2024-01-17 10:00:00 1010.9063 541.3602 536.6247 145.69981 68.60264 77.09717 387.15997 3.7648749
4: 2024-01-17 11:00:00 1154.3223 592.0663 662.0048 173.72247 77.44190 96.28057 483.49354 4.7887550
5: 2024-01-17 12:00:00 1213.1770 612.8750 716.5974 185.35767 80.61172 104.74595 526.00427 5.2354830
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 578.4583 180.82966 95.85462 84.97504 393.39650 4.2321949
142: 2024-12-13 13:00:00 1129.5610 453.5904 536.8668 170.08970 91.70559 78.38411 362.88341 3.8937280
143: 2024-12-13 14:00:00 994.4636 409.9651 434.0673 142.25355 79.88147 62.37208 288.75488 3.0588416
144: 2024-12-13 15:00:00 785.0640 342.3463 292.1950 99.92831 58.81096 41.11735 190.35496 1.9117069
145: 2024-12-13 16:00:00 515.6229 255.3390 140.8937 46.94651 26.80445 20.14206 93.24874 0.6984426
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 0.24100175 0.09714708 0.3918962 21.44734 18.51982 133.4885 39.96716 93.00127 0.5200533
2: 0.10321543 0.09714708 0.3918962 43.29416 42.77007 302.1765 86.06424 214.77909 1.3331982
3: 0.04727214 0.09714708 0.3918962 56.98305 67.57641 466.0152 124.55946 339.34944 2.1062799
4: 0.02455379 0.09714708 0.3918962 64.32515 86.40320 587.2996 150.72835 433.89218 2.6790952
5: 0.01743586 0.09714708 0.3918962 66.95809 94.68605 640.0594 161.64413 475.48630 2.9290196
---
141: 0.02100686 0.09714708 0.3918962 79.61921 76.53478 512.8436 156.15400 354.32187 2.3677246
142: 0.02771140 0.09714708 0.3918962 76.17293 70.11502 473.0675 146.28795 324.60121 2.1783674
143: 0.05023795 0.09714708 0.3918962 66.35152 54.49955 374.8709 120.85106 252.30856 1.7112857
144: 0.10550059 0.09714708 0.3918962 48.84983 33.83709 240.4070 82.68692 156.65061 1.0695149
145: 0.23984890 0.09714708 0.3918962 22.26444 14.08613 101.9538 36.35057 65.21248 0.3907476

```

- **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(compI = compI, angGen = angGen, alb = 0)
2 show(inclin_alb0)

```

```

      Dates      Bo      Bn      G      D      Di      Dc      B      R
      <POS<      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 514.5612 365.8727 185.5294 52.34286 25.82073 26.52212 133.18653 0
2: 2024-01-17 09:00:00 792.6980 464.2106 364.2874 103.96230 52.12242 51.83988 260.32510 0
3: 2024-01-17 10:00:00 1010.9063 541.3602 532.8598 145.69981 68.60264 77.09717 387.15997 0
4: 2024-01-17 11:00:00 1154.3223 592.0663 657.2160 173.72247 77.44190 96.28057 483.49354 0
5: 2024-01-17 12:00:00 1213.1770 612.8750 711.3619 185.35767 80.61172 104.74595 526.00427 0
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 574.2262 180.82966 95.85462 84.97504 393.39650 0
142: 2024-12-13 13:00:00 1129.5610 453.5904 532.9731 170.08970 91.70559 78.38411 362.88341 0

```

```

143: 2024-12-13 14:00:00 994.4636 409.9651 431.0084 142.25355 79.88147 62.37208 288.75488 0
144: 2024-12-13 15:00:00 785.0640 342.3463 290.2833 99.92831 58.81096 41.11735 190.35496 0
145: 2024-12-13 16:00:00 515.6229 255.3390 140.1953 46.94651 26.80445 20.14206 93.24874 0
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 0.156321477 0.06473603 0.2994808 23.66622 21.92862 155.7141 45.59484 110.11928 0
2: 0.054197292 0.06473603 0.2994808 47.77325 48.04970 337.1148 95.82295 241.29186 0
3: 0.021399057 0.06473603 0.2994808 62.87835 73.93841 508.1144 136.81676 371.29761 0
4: 0.010185772 0.06473603 0.2994808 70.98005 93.39388 633.3713 164.37393 468.99741 0
5: 0.006996517 0.06473603 0.2994808 73.88537 101.93283 687.6958 175.81821 511.87759 0
---
141: 0.008575046 0.06473603 0.2994808 87.85638 82.56145 552.6405 170.41783 382.22264 0
142: 0.011653979 0.06473603 0.2994808 84.05356 75.92121 511.4560 159.97477 351.48128 0
143: 0.022965930 0.06473603 0.2994808 73.21605 59.72086 409.4178 132.93691 276.48089 0
144: 0.055666181 0.06473603 0.2994808 53.90370 38.05193 268.1191 91.95563 176.16345 0
145: 0.155368802 0.06473603 0.2994808 24.56786 16.67236 118.4258 41.24021 77.18558 0

```

```

1 inclin_alb1 <- fInclin(compI = compI, angGen = angGen, alb = 1)
2 show(inclin_alb1)

```

```

      Dates      Bo      Bn      G      D      Di      Dc      B      R
      <POSc>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 2024-01-17 08:00:00 514.5612 365.8727 190.1772 52.34286 25.82073 26.52212 133.18653 4.647853
2: 2024-01-17 09:00:00 792.6980 464.2106 376.2025 103.96230 52.12242 51.83988 260.32510 11.915141
3: 2024-01-17 10:00:00 1010.9063 541.3602 551.6842 145.69981 68.60264 77.09717 387.15997 18.824375
4: 2024-01-17 11:00:00 1154.3223 592.0663 681.1598 173.72247 77.44190 96.28057 483.49354 23.943775
5: 2024-01-17 12:00:00 1213.1770 612.8750 737.5394 185.35767 80.61172 104.74595 526.00427 26.177415
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 595.3871 180.82966 95.85462 84.97504 393.39650 21.160975
142: 2024-12-13 13:00:00 1129.5610 453.5904 552.4417 170.08970 91.70559 78.38411 362.88341 19.468640
143: 2024-12-13 14:00:00 994.4636 409.9651 446.3026 142.25355 79.88147 62.37208 288.75488 15.294208
144: 2024-12-13 15:00:00 785.0640 342.3463 299.8418 99.92831 58.81096 41.11735 190.35496 9.558535
145: 2024-12-13 16:00:00 515.6229 255.3390 143.6875 46.94651 26.80445 20.14206 93.24874 3.492213
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: 0.156321477 0.06473603 0.2994808 23.66622 21.92862 158.9049 45.59484 110.11928 3.190792
2: 0.054197292 0.06473603 0.2994808 47.77325 48.04970 345.2947 95.82295 241.29186 8.179849
3: 0.021399057 0.06473603 0.2994808 62.87835 73.93841 521.0375 136.81676 371.29761 12.923098
4: 0.010185772 0.06473603 0.2994808 70.98005 93.39388 649.8089 164.37393 468.99741 16.437612
5: 0.006996517 0.06473603 0.2994808 73.88537 101.93283 705.6668 175.81821 511.87759 17.971025
---
141: 0.008575046 0.06473603 0.2994808 87.85638 82.56145 567.1677 170.41783 382.22264 14.527195
142: 0.011653979 0.06473603 0.2994808 84.05356 75.92121 524.8214 159.97477 351.48128 13.365392
143: 0.022965930 0.06473603 0.2994808 73.21605 59.72086 419.9174 132.93691 276.48089 10.499608
144: 0.055666181 0.06473603 0.2994808 53.90370 38.05193 274.6811 91.95563 176.16345 6.562018
145: 0.155368802 0.06473603 0.2994808 24.56786 16.67236 120.8232 41.24021 77.18558 2.397435

```

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**<sup>7</sup> (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(compI = compI, angGen = angGen,
2                               horizBright = FALSE)
3 show(inclin_horizBright)

```

```

      Dates      Bo      Bn      G      D      Di      Dc      B      R
      <POSc>      <num>      <num>      <num>      <num>      <num>      <num>      <num>      <num>

```

<sup>7</sup>la 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 menor cantidad de semiconductores reduciendo los costes.

#### 4. DESARROLLO DEL CÓDIGO

```

1: 2024-01-17 08:00:00 514.5612 365.8727 186.2091 52.09303 25.57090 26.52212 133.18653 0.9295706
2: 2024-01-17 09:00:00 792.6980 464.2106 366.1413 103.43314 51.59325 51.83988 260.32510 2.3830282
3: 2024-01-17 10:00:00 1010.9063 541.3602 535.9087 144.98390 67.88673 77.09717 387.15997 3.7648749
4: 2024-01-17 11:00:00 1154.3223 592.0663 661.1846 172.90227 76.62170 96.28057 483.49354 4.7887550
5: 2024-01-17 12:00:00 1213.1770 612.8750 715.7390 184.49921 79.75326 104.74595 526.00427 5.2354830
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 577.4973 179.86860 94.89356 84.97504 393.39650 4.2321949
142: 2024-12-13 13:00:00 1129.5610 453.5904 535.9539 169.17679 90.79268 78.38411 362.88341 3.8937280
143: 2024-12-13 14:00:00 994.4636 409.9651 433.2885 141.47476 79.10268 62.37208 288.75488 3.0588416
144: 2024-12-13 15:00:00 785.0640 342.3463 291.6442 99.37758 58.26023 41.11735 190.35496 1.9117069
145: 2024-12-13 16:00:00 515.6229 255.3390 140.6606 46.71344 26.57138 20.14206 93.24874 0.6984426
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num> <num> <num> <num> <num> <num> <num> <num> <num>
1: 0.156321477 0.06473603 0.2994808 23.43723 21.92862 156.1233 45.36586 110.11928 0.6381583
2: 0.054197292 0.06473603 0.2994808 47.28824 48.04970 338.2658 95.33794 241.29186 1.6359698
3: 0.021399057 0.06473603 0.2994808 62.22217 73.93841 510.0428 136.16059 371.29761 2.5846197
4: 0.010185772 0.06473603 0.2994808 70.22829 93.39388 635.9071 163.62217 468.99741 3.2875223
5: 0.006996517 0.06473603 0.2994808 73.09855 101.93283 690.5032 175.03138 511.87759 3.5942050
---
141: 0.008575046 0.06473603 0.2994808 86.97552 82.56145 554.6650 169.53697 382.22264 2.9054390
142: 0.011653979 0.06473603 0.2994808 83.21682 75.92121 513.2924 159.13803 351.48128 2.6730784
143: 0.022965930 0.06473603 0.2994808 72.50225 59.72086 410.8039 132.22311 276.48089 2.0999216
144: 0.055666181 0.06473603 0.2994808 53.39892 38.05193 268.9267 91.45086 176.16345 1.3124036
145: 0.155368802 0.06473603 0.2994808 24.35423 16.67236 118.6917 41.02659 77.18558 0.4794870

```

```

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

```

```

      Dates      Bo      Bn      G      D      Di      Dc      B      R
      <POS< <num> <num> <num> <num> <num> <num> <num> <num>
1: 2024-01-17 08:00:00 514.5612 365.8727 186.4590 52.34286 25.82073 26.52212 133.18653 0.9295706
2: 2024-01-17 09:00:00 792.6980 464.2106 366.6704 103.96230 52.12242 51.83988 260.32510 2.3830282
3: 2024-01-17 10:00:00 1010.9063 541.3602 536.6247 145.69981 68.60264 77.09717 387.15997 3.7648749
4: 2024-01-17 11:00:00 1154.3223 592.0663 662.0048 173.72247 77.44190 96.28057 483.49354 4.7887550
5: 2024-01-17 12:00:00 1213.1770 612.8750 716.5974 185.35767 80.61172 104.74595 526.00427 5.2354830
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 578.4583 180.82966 95.85462 84.97504 393.39650 4.2321949
142: 2024-12-13 13:00:00 1129.5610 453.5904 536.8668 170.08970 91.70559 78.38411 362.88341 3.8937280
143: 2024-12-13 14:00:00 994.4636 409.9651 434.0673 142.25355 79.88147 62.37208 288.75488 3.0588416
144: 2024-12-13 15:00:00 785.0640 342.3463 292.1950 99.92831 58.81096 41.11735 190.35496 1.9117069
145: 2024-12-13 16:00:00 515.6229 255.3390 140.8937 46.94651 26.80445 20.14206 93.24874 0.6984426
      FTb      FTd      FTr      Dief      Dcef      Gef      Def      Bef      Ref
      <num> <num> <num> <num> <num> <num> <num> <num> <num>
1: 0.156321477 0.06473603 0.2994808 0 0 110.11928 0 110.11928 0
2: 0.054197292 0.06473603 0.2994808 0 0 241.29186 0 241.29186 0
3: 0.021399057 0.06473603 0.2994808 0 0 371.29761 0 371.29761 0
4: 0.010185772 0.06473603 0.2994808 0 0 468.99741 0 468.99741 0
5: 0.006996517 0.06473603 0.2994808 0 0 511.87759 0 511.87759 0
---
141: 0.008575046 0.06473603 0.2994808 0 0 382.22264 0 382.22264 0
142: 0.011653979 0.06473603 0.2994808 0 0 351.48128 0 351.48128 0
143: 0.022965930 0.06473603 0.2994808 0 0 276.48089 0 276.48089 0
144: 0.055666181 0.06473603 0.2994808 0 0 176.16345 0 176.16345 0
145: 0.155368802 0.06473603 0.2994808 0 0 77.18558 0 77.18558 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 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, alfa = 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
	<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.

```

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 show(gef_shdbt)

```

```

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  8.071623  4.924221  4.069604  1.101792  2.902196  3.802336  1.0232875  2.724604
2: Feb. 2024 10.170791  5.034287  4.943127  1.417056  3.445443  4.680459  1.3258434  3.287780
3: Mar. 2024 12.816149  5.163713  6.094523  1.850253  4.148386  5.841685  1.7419635  4.020914
4: Apr. 2024 15.326568  6.408617  8.007438  2.166491  5.716983  7.711198  2.0485357  5.560571
5: May. 2024 16.624320  7.617499  9.439815  2.303156  7.000336  9.132906  2.1878882  6.833933
6: Jun. 2024 17.408383  9.102430 10.652929  2.206022  8.288629 10.286974  2.0977541  8.059004
7: Jul. 2024 16.861601 10.037233 11.038213  1.944739  8.935057 10.701158  1.8585291  8.712900
8: Aug. 2024 15.551202  8.640959  9.872463  1.850828  7.878525  9.562356  1.7662720  7.678732
9: Sep. 2024 13.422796  6.698488  7.568105  1.795358  5.655421  7.285297  1.7012821  5.487114
10: Oct. 2024 10.764846  4.546024  4.915408  1.521915  3.310678  4.666904  1.4246602  3.173452
11: Nov. 2024  8.434950  4.638289  4.079866  1.156410  2.854293  3.813241  1.0737415  2.681776
12: Dec. 2024  7.370928  3.439788  3.062505  1.023011  1.987550  2.836653  0.9441838  1.849321

Yearly values:
  Dates      Bod      Bnd      Gd      Dd      Bd      Gefd      Defd      Befd
<int>    <num>    <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 2024 4662.615 2326.882 2555.869 620.2896 1896.422 2451.499 585.4392 1833.809
-----
Mode of tracking: horiz
Inclination limit: 90

```

## 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** [A.1.4] la cual mediante la función **fProd** [A.3.7] procesa un objeto de clase **Gef** y obtiene un objeto **ProdGCPV**.

Como se puede ver en la figura 4.6, **prodGCPV** funciona gracias a la siguiente función:

- **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 mérida). Por defecto, tiene un valor de 57,2V.

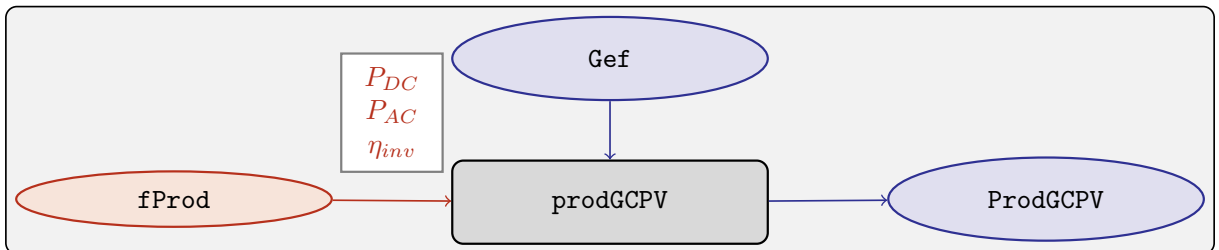


FIGURA 4.6: Estimación de la producción eléctrica de un SFCR mediante la función **prodGCPV**, la cual emplea la función **fProd** para el computo de la potencia a la entrada ( $P_{DC}$ ), a la salida ( $P_{AC}$ ) y el rendimiento ( $\eta_{inv}$ ) del inversor.

- **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.

```

1 inclin <- calcGef(lat, dataRad = prom, keep.night = FALSE)
2 module <- list(Vocn=57.6, Iscn=4.7, Vmn=46.08, Imn=4.35,
3               Ncs=96, Ncp=1, CoefVT=0.0023, TONC=47)

```

```

4 generator <- list(Nms=12, Nmp=11)
5 inverter <- list(Ki=c(0.01, 0.025, 0.05), Pinv=25000,
6               Vmin=420, Vmax=750, 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 show(prod)

```

	Dates <POS<	Tc <num>	Voc <num>	Isc <num>	Vmpp <num>	Imp <num>	Vdc <num>	Idc <num>	Pac <num>
1:	2024-01-17 08:00:00	15.27689	716.9624	8.083413	607.4640	7.620135	607.4640	7.620135	3796.209
2:	2024-01-17 09:00:00	21.43284	700.6516	17.513415	583.9663	16.433741	583.9663	16.433741	8053.912
3:	2024-01-17 10:00:00	27.23609	685.2753	26.403138	562.0190	24.658263	562.0190	24.658263	11650.920
4:	2024-01-17 11:00:00	31.48724	674.0114	32.915263	546.0746	30.625265	546.0746	30.625265	14041.629
5:	2024-01-17 12:00:00	33.33104	669.1261	35.739693	539.1958	33.196772	539.1958	33.196772	15016.481
---									
141:	2024-12-13 12:00:00	33.94967	667.4869	28.721724	542.4718	26.706186	542.4718	26.706186	12177.570
142:	2024-12-13 13:00:00	32.55186	671.1906	26.580476	547.6944	24.746716	547.6944	24.746716	11395.331
143:	2024-12-13 14:00:00	29.08872	680.3665	21.275466	560.6878	19.868077	560.6878	19.868077	9362.088
144:	2024-12-13 15:00:00	24.29331	693.0724	13.929608	578.8034	13.059814	578.8034	13.059814	6316.091
145:	2024-12-13 16:00:00	19.21305	706.5331	6.147403	598.1441	5.786102	598.1441	5.786102	2784.663
---									
	Pdc <num>	EffI <num>							
1:	4290.940	0.9118076							
2:	8895.974	0.9330800							
3:	12846.437	0.9347232							
4:	15502.477	0.9335163							
5:	16592.492	0.9327431							
---									
141:	13429.451	0.9345615							
142:	12563.918	0.9347755							
143:	10326.335	0.9343983							
144:	7007.083	0.9290019							
145:	3208.198	0.8945754							

Esta función brinda estos 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 avarages:
  Dates      Eac      Edc      Yf
  <char>    <num>    <num>    <num>
1: Jan. 2024 95.36291 105.62767 3.604158
2: Feb. 2024 101.50809 112.56166 3.836410
3: Mar. 2024 110.26945 122.11835 4.167538
4: Apr. 2024 124.53728 138.29836 4.706778
5: May. 2024 131.48629 145.91065 4.969410
6: Jun. 2024 135.89421 150.78725 5.136003
7: Jul. 2024 134.98501 149.81246 5.101641
8: Aug. 2024 130.25804 144.39951 4.922989

```

## 4. DESARROLLO DEL CÓDIGO

```
9: Sep. 2024 119.91911 132.77648 4.532238
10: Oct. 2024 96.49455 106.99182 3.646928
11: Nov. 2024 90.17737 99.88152 3.408175
12: Dec. 2024 73.89289 81.80967 2.792718
```

Yearly values:

	Dates	Eac	Edc	Yf
	<int>	<num>	<num>	<num>
1:	2024	41014.8	45473.37	1550.119

Mode of tracking: fixed

Inclination: 27.2

Orientation: 0

Generator:

Modules in series: 12

Modules in parallel: 11

Nominal power (kWp): 26.5

```
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 avarages:

	Dates	Eac	Edc	Yf
	<char>	<num>	<num>	<num>
1:	Jan. 2024	138.6806	153.2566	5.241314
2:	Feb. 2024	143.4987	158.5247	5.423408
3:	Mar. 2024	151.8477	167.7311	5.738952
4:	Apr. 2024	178.6717	197.4274	6.752741
5:	May. 2024	200.8888	222.0523	7.592419
6:	Jun. 2024	223.9959	247.6903	8.465728
7:	Jul. 2024	214.2749	236.9628	8.098332
8:	Aug. 2024	194.6043	215.1439	7.354902
9:	Sep. 2024	168.9824	186.7349	6.386542
10:	Oct. 2024	132.2995	146.0747	5.000145
11:	Nov. 2024	128.5783	141.9871	4.859507
12:	Dec. 2024	102.9116	113.5613	3.889454

Yearly values:

	Dates	Eac	Edc	Yf
	<int>	<num>	<num>	<num>
1:	2024	60369.04	66710.67	2281.595

Mode of tracking: two

Inclination limit: 90

Generator:

Modules in series: 12

Modules in parallel: 11

Nominal power (kWp): 26.5

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

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	99.43006	109.66074	3.757873
2:	Feb. 2024	116.24796	128.22238	4.393490
3:	Mar. 2024	137.39485	151.61074	5.192719
4:	Apr. 2024	172.03044	189.97488	6.501741
5:	May. 2024	196.91337	217.61396	7.442169
6:	Jun. 2024	219.15566	242.31468	8.282797
7:	Jul. 2024	210.33644	232.56087	7.949482
8:	Aug. 2024	189.03576	208.87993	7.144442
9:	Sep. 2024	156.22909	172.44519	5.904542
10:	Oct. 2024	110.69482	122.11859	4.183614
11:	Nov. 2024	94.40734	104.14723	3.568043
12:	Dec. 2024	69.94550	77.30532	2.643529

Yearly values:

	Dates	Eac	Edc	Yf
	<int>	<num>	<num>	<num>
1:	2024	54052.14	59697.16	2042.854

Mode of tracking: horiz  
Inclination limit: 90

Generator:

Modules in series: 12  
Modules in parallel: 11  
Nominal power (kWp): 26.5

## 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 4.7, **prodPVPS** funciona gracias a la siguiente función:

- **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 parametros de la bomba que va a ser simulada. Puede ser una fila de **pumpCoef**:

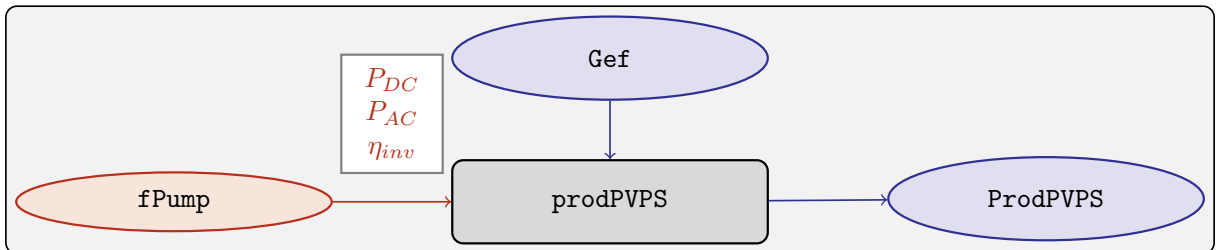


FIGURA 4.7: Estimación de la producción eléctrica de un SFB mediante la función **prodPVPS**, la cual emplea la función **fPump** para el computo del rendimiento de las diferentes parte de una bomba centrífuga alimentada por un convertidor de frecuencia.

```
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: 0x00000276e060f568>
<environment: 0x00000276e060c500>
```

- **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: 0x00000276e060f568>
  <environment: 0x00000276e0639258>

```

- **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: 0x00000276e060f568>
<environment: 0x00000276e063fc68>

```

- **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: 0x00000276e060f568>
<environment: 0x00000276e062e928>

```

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 <num>	Pac <num>	Pb <num>	Ph <num>	etam <num>	etab <num>	f <num>
1:	0.3133325	190.1000	124.8346	34.15325	0.6566786	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 show(prodsfb)

```

```

Object of class  ProdPVPS

Source of meteorological information: prom-

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

Monthly avarages:
      Dates      Eac      Qd      Yf
      <char>    <num>    <num>    <num>
1: Jan. 2024  14.07129  50.46621  3.445284
2: Feb. 2024  15.43701  54.71213  3.779675
3: Mar. 2024  17.00102  59.68995  4.162613
4: Apr. 2024  19.39135  67.24260  4.747874
5: May. 2024  20.65046  71.34195  5.056160
6: Jun. 2024  21.63947  74.27359  5.298315
7: Jul. 2024  22.62915  76.77927  5.540633
8: Aug. 2024  22.17136  75.07166  5.428546
9: Sep. 2024  19.61622  67.34348  4.802932
10: Oct. 2024  14.92078  53.24853  3.653277
11: Nov. 2024  13.75298  49.50040  3.367348
12: Dec. 2024  11.21349  40.90244  2.745567

Yearly values:
      Dates      Eac      Qd      Yf
      <int>    <num>    <num>    <num>
1: 2024  6482.059  22589.95  1587.099
-----
Mode of tracking:  fixed
  Inclination:    27.2
  Orientation:    0
-----
Pump:
  Qn: 8
  Stages: 44
Height (m): 40
Generator (Wp): 4084.218

```

### 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 optar cual 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**.

```

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 show(ShdM2x)

```

```

Object of class  Shade

Source of meteorological information: prom-

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

Monthly avarages:
Dimensions of structure:
$W
[1] 23.11

$L
[1] 9.8

$Nrow
[1] 2

$Ncol
[1] 3

Shade calculation mode:
[1] "area" "prom"
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 138.6806 153.2566 5.241314
2: Feb. 2024 143.4987 158.5247 5.423408
3: Mar. 2024 151.8477 167.7311 5.738952
4: Apr. 2024 178.6717 197.4274 6.752741
5: May. 2024 200.8888 222.0523 7.592419
6: Jun. 2024 223.9959 247.6903 8.465728
7: Jul. 2024 214.2749 236.9628 8.098332
8: Aug. 2024 194.6043 215.1439 7.354902
9: Sep. 2024 168.9824 186.7349 6.386542
10: Oct. 2024 132.2995 146.0747 5.000145
11: Nov. 2024 128.5783 141.9871 4.859507
12: Dec. 2024 102.9116 113.5613 3.889454

Yearly values:
  Dates      Eac      Edc      Yf
  <int>    <num>    <num>    <num>
1: 2024 60369.04 66710.67 2281.595

```

```

-----
Mode of tracking: two
Inclination limit: 90
-----
Generator:
  Modules in series: 12
  Modules in parallel: 11
  Nominal power (kWp): 26.5

Summary of results:

```

Lew		Lns		H		FS		GRR		Yf	
Min.	:30.00	Min.	:20	Min.	:0	Min.	:0.01509	Min.	:2.649	Min.	:2104
1st Qu.	:33.75	1st Qu.	:25	1st Qu.	:0	1st Qu.	:0.02223	1st Qu.	:3.946	1st Qu.	:2192
Median	:37.50	Median	:30	Median	:0	Median	:0.02870	Median	:4.802	Median	:2216
Mean	:37.50	Mean	:30	Mean	:0	Mean	:0.03463	Mean	:4.967	Mean	:2203
3rd Qu.	:41.25	3rd Qu.	:35	3rd Qu.	:0	3rd Qu.	:0.03945	3rd Qu.	:6.016	3rd Qu.	:2231
Max.	:45.00	Max.	:40	Max.	:0	Max.	:0.07769	Max.	:7.948	Max.	:2247

```

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 avarages:
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	97.48365	107.53823	3.684309
2:	Feb. 2024	114.31569	126.11751	4.320462
3:	Mar. 2024	135.67629	149.74056	5.127767
4:	Apr. 2024	170.28530	188.06424	6.435785
5:	May. 2024	194.83600	215.33865	7.363657
6:	Jun. 2024	216.37522	239.26256	8.177713
7:	Jul. 2024	208.20413	230.21074	7.868894
8:	Aug. 2024	187.19428	206.84745	7.074845
9:	Sep. 2024	154.37402	170.41035	5.834432
10:	Oct. 2024	109.27362	120.57435	4.129901
11:	Nov. 2024	92.82584	102.42576	3.508272
12:	Dec. 2024	69.13228	76.42401	2.612794

```

Yearly values:
  Dates      Eac      Edc      Yf
  <int>      <num>      <num>      <num>
1:  2024 53386.77 58969.19 2017.707
-----
Mode of tracking:  horiz
  Inclination limit: 60
-----
Generator:
  Modules in series: 12
  Modules in parallel: 11
  Nominal power (kWp): 26.5

Summary of results:
      Lew      H      FS      GRR      Yf
Min.   : 9.66  Min.   :0  Min.   :0.04804  Min.   :2.000  Min.   :1736
1st Qu.:13.16  1st Qu.:0  1st Qu.:0.05727  1st Qu.:2.725  1st Qu.:1824
Median :16.66  Median :0  Median :0.07295  Median :3.449  Median :1871
Mean   :16.66  Mean   :0  Mean   :0.08078  Mean   :3.449  Mean   :1855
3rd Qu.:20.16  3rd Qu.:0  3rd Qu.:0.09598  3rd Qu.:4.174  3rd Qu.:1902
Max.   :23.66  Max.   :0  Max.   :0.13968  Max.   :4.899  Max.   :1921

```

```

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 show(Shd12Fixed)

```

```

Object of class  Shade

Source of meteorological information: prom-

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

Monthly avarages:
Dimensions of structure:
$L
[1] 5

Shade calculation mode:
[1] "area"
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  95.36291 105.62767 3.604158
2: Feb. 2024 101.50809 112.56166 3.836410
3: Mar. 2024 110.26945 122.11835 4.167538
4: Apr. 2024 124.53728 138.29836 4.706778
5: May. 2024 131.48629 145.91065 4.969410
6: Jun. 2024 135.89421 150.78725 5.136003
7: Jul. 2024 134.98501 149.81246 5.101641
8: Aug. 2024 130.25804 144.39951 4.922989

```

```
9: Sep. 2024 119.91911 132.77648 4.532238
10: Oct. 2024 96.49455 106.99182 3.646928
11: Nov. 2024 90.17737 99.88152 3.408175
12: Dec. 2024 73.89289 81.80967 2.792718

Yearly values:
  Dates      Eac      Edc      Yf
  <int>    <num>    <num>    <num>
1: 2024 41014.8 45473.37 1550.119
-----
Mode of tracking: fixed
  Inclination: 27.2
  Orientation: 0
-----
Generator:
  Modules in series: 12
  Modules in parallel: 11
  Nominal power (kWp): 26.5

Summary of results:
      D      H      FS      GRR      Yf
Min.   : 5.0   Min.   :0   Min.   :0.0008477   Min.   :1.0   Min.   :1364
1st Qu.: 7.5   1st Qu.:0   1st Qu.:0.0015710   1st Qu.:1.5   1st Qu.:1511
Median :10.0   Median :0   Median :0.0038992   Median :2.0   Median :1544
Mean   :10.0   Mean   :0   Mean   :0.0269608   Mean   :2.0   Mean   :1508
3rd Qu.:12.5   3rd Qu.:0   3rd Qu.:0.0252790   3rd Qu.:2.5   3rd Qu.:1548
Max.   :15.0   Max.   :0   Max.   :0.1199180   Max.   :3.0   Max.   :1549
```

4.8. Métodos de visualización

Una vez creados todos los objetos, para mejorar la visualización de los mismos, `solar2` cuenta con una serie de métodos que ayudan a la compresión de los datos obtenidos.

4.8.1. Datos meteorológicos

La clase `Meteo` cuenta con un método para `xyplot`.

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

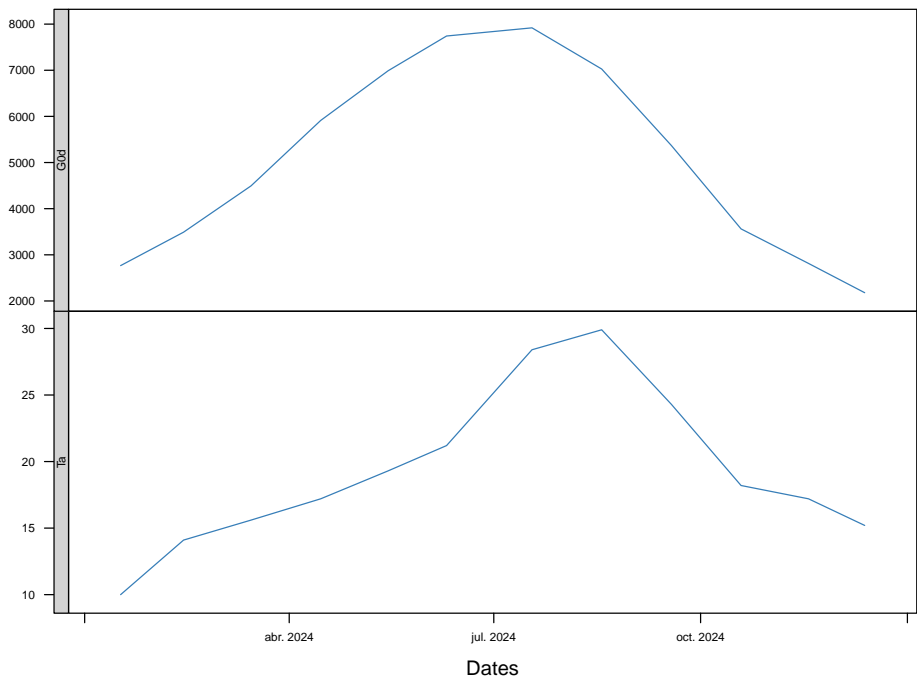
```
Object of class  Meteo

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

Meteorological Data:
      Dates      G0d      Ta
Min.   :2024-01-17   Min.   :2179   Min.   :10.00
1st Qu.:2024-04-07   1st Qu.:3322   1st Qu.:15.50
Median :2024-06-29   Median :4932   Median :17.70
Mean   :2024-07-01   Mean   :5022   Mean   :19.22
3rd Qu.:2024-09-25   3rd Qu.:6998   3rd Qu.:21.98
Max.   :2024-12-13   Max.   :7919   Max.   :29.90
```

```
1 xyplot(BD)
```





4.8.2. Radiación en el plano horizontal

La clase **G0** cuenta con un método para **xyplot**.

```
1 g0 <- calcG0(lat, dataRad = BD)
2 show(g0)
```

```
Object of class  G0

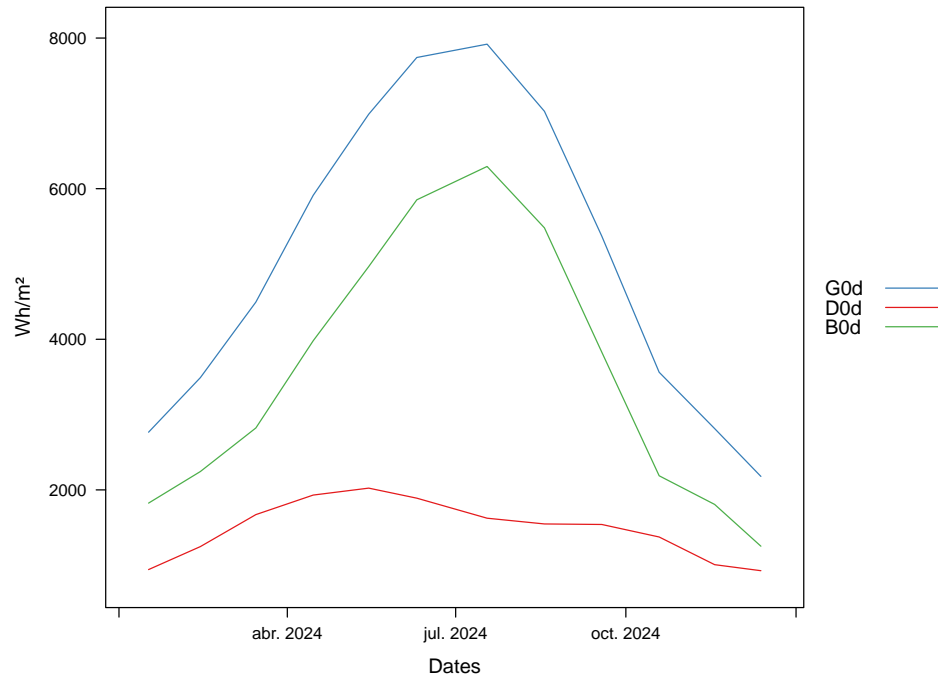
Source of meteorological information: prom-

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  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
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

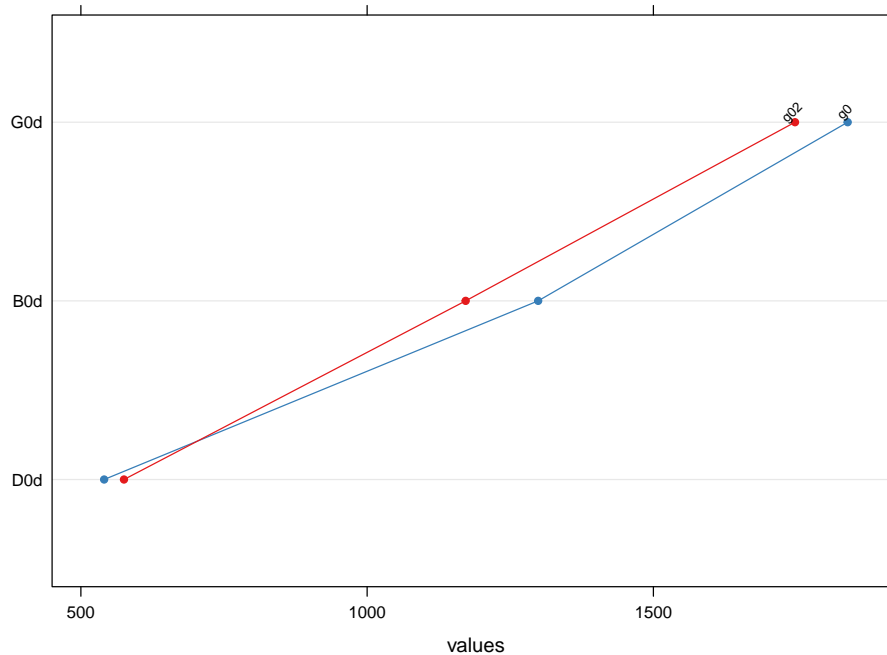
Yearly values:
      Dates      G0d      D0d      B0d
  <int>    <num>    <num>    <num>
1: 2024 1839.365  540.6331 1298.732
```

```
1 xyplot(g0)
```



Y con un método para **compare**.

```
1 g02 <- calcG0(lat, dataRad = list(G0dm = G0dm*0.95, Ta = Ta))
2 compare(g0, g02)
```



### 4.8.3. Radiación efectiva en el plano del generador

La clase **Gef** cuenta con un método para **xyplot**.

```
1 gef <- calcGef(lat, dataRad = BD)
2 show(gef)
```

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	8.724907	4.924221	4.489744	1.200992	3.258164	4.220907	1.119517	3.080392
2:	Feb. 2024	9.592013	5.034287	4.919206	1.451954	3.428647	4.628492	1.352529	3.249460
3:	Mar. 2024	10.281308	5.163713	5.413543	1.779951	3.583896	5.101556	1.657369	3.410070
4:	Apr. 2024	10.527227	6.408617	6.282631	1.936897	4.280357	5.918787	1.803811	4.070094
5:	May. 2024	10.431853	7.617499	6.784202	1.937331	4.769584	6.371295	1.802060	4.516177
6:	Jun. 2024	10.291163	9.102430	7.173475	1.762326	5.325535	6.725684	1.639192	5.027718
7:	Jul. 2024	10.305302	10.037233	7.511733	1.533887	5.890275	7.058263	1.430322	5.567823
8:	Aug. 2024	10.394682	8.640959	7.295543	1.545089	5.672747	6.879777	1.443952	5.382478
9:	Sep. 2024	10.233884	6.698488	6.335591	1.647975	4.628244	5.982520	1.539552	4.402209
10:	Oct. 2024	9.659077	4.546024	4.746760	1.538325	3.169044	4.470026	1.432213	3.010771
11:	Nov. 2024	8.798687	4.638289	4.393712	1.244217	3.118376	4.134590	1.159756	2.953471
12:	Dec. 2024	8.176298	3.439788	3.478125	1.128381	2.325648	3.274677	1.050626	2.207509

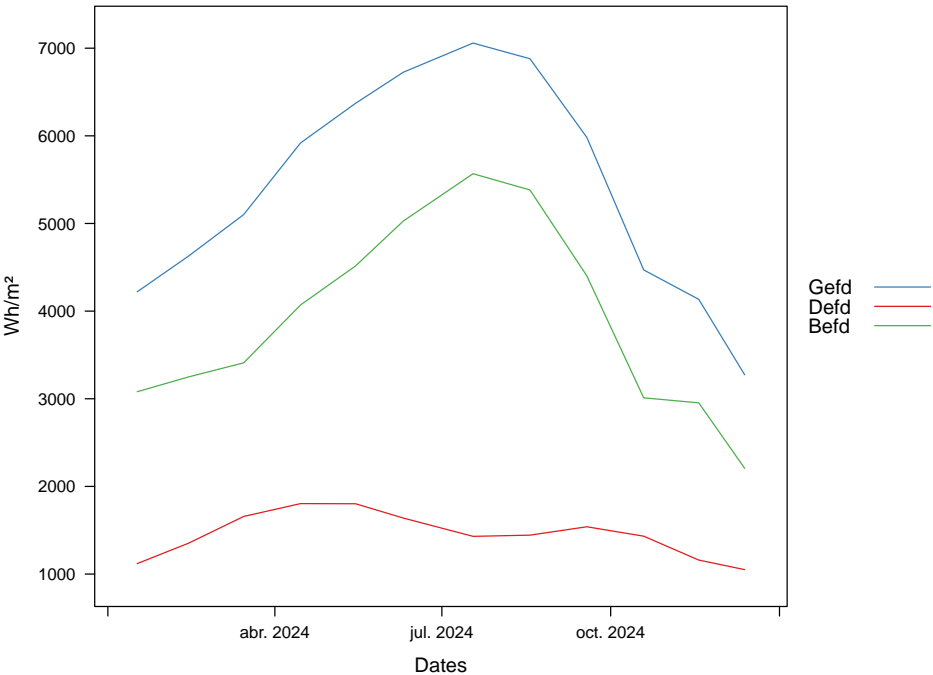
Yearly values:

	Dates	Bod	Bnd	Gd	Dd	Bd	Gefd	Defd	Befd
	<int>	<num>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	2024	3580.873	2326.882	2099.528	570.4317	1508.756	1975.745	531.5105	1430.271

-----

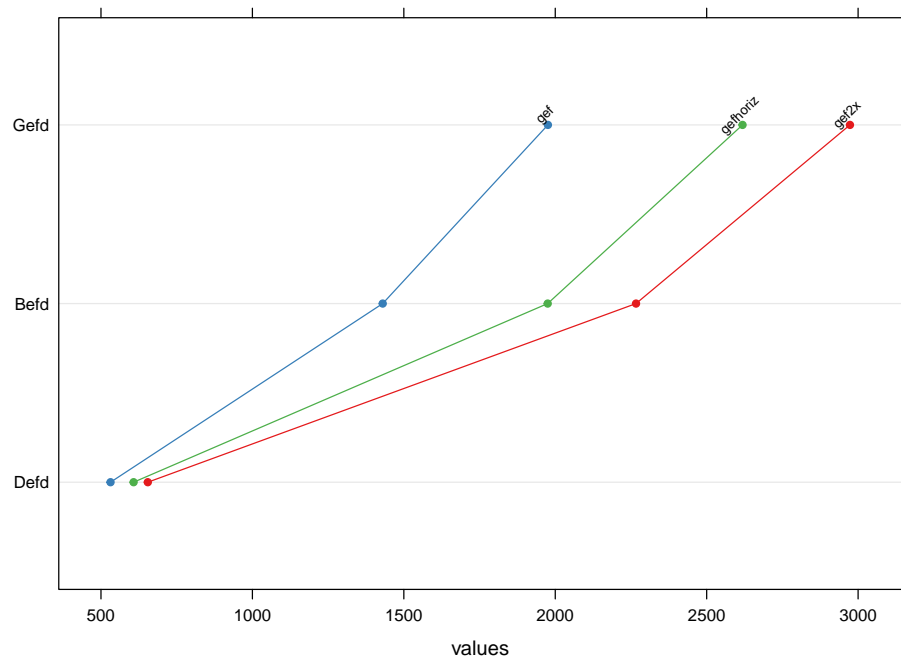
Mode of tracking: fixed  
Inclination: 27.2  
Orientation: 0

```
1 xyplot(gef)
```



Y con un método para **compare**.

```
1 gef2x <- calcGef(lat, modeTrk = 'two', dataRad = BD)
2 gefhoriz <- calcGef(lat, modeTrk = 'horiz', dataRad = BD)
3 compare(gef, gef2x, gefhoriz)
```



4.8.4. Producción eléctrica de un SFCR

La clase **ProdGCPV** cuenta con un método para **xyplot**.

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

```
Object of class ProdGCPV

Source of meteorological information: prom-

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

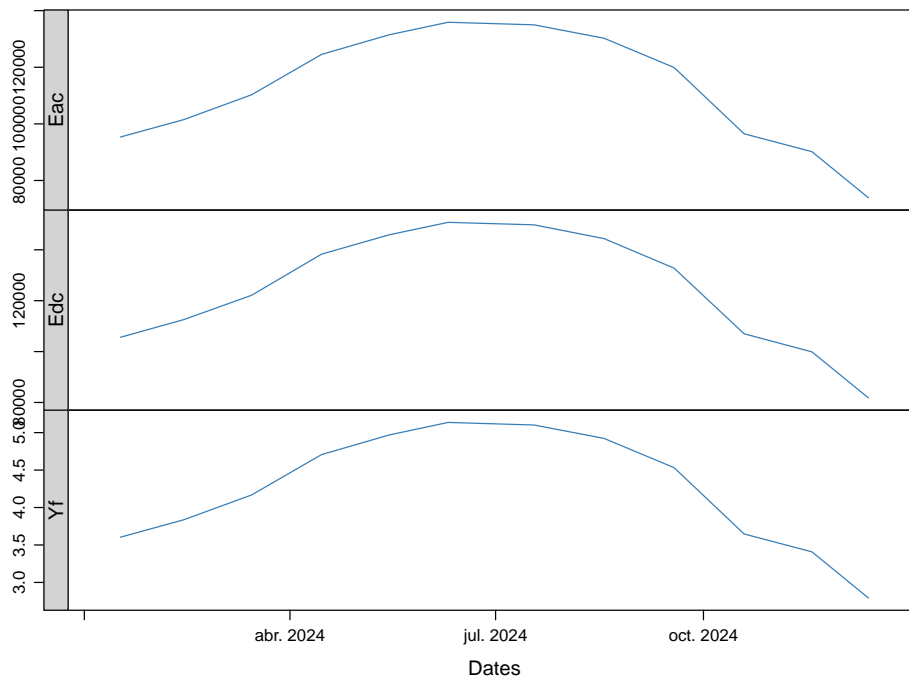
Monthly avarages:
  Dates      Eac      Edc      Yf
  <char>    <num>    <num>    <num>
1: Jan. 2024 95.36291 105.62767 3.604158
2: Feb. 2024 101.50809 112.56166 3.836410
3: Mar. 2024 110.26945 122.11835 4.167538
4: Apr. 2024 124.53728 138.29836 4.706778
5: May. 2024 131.48629 145.91065 4.969410
6: Jun. 2024 135.89421 150.78725 5.136003
7: Jul. 2024 134.98501 149.81246 5.101641
8: Aug. 2024 130.25804 144.39951 4.922989
9: Sep. 2024 119.91911 132.77648 4.532238
10: Oct. 2024 96.49455 106.99182 3.646928
11: Nov. 2024 90.17737 99.88152 3.408175
12: Dec. 2024 73.89289 81.80967 2.792718
```

```

Yearly values:
  Dates      Eac      Edc      Yf
<int> <num> <num> <num>
1: 2024 41014.8 45473.37 1550.119
-----
Mode of tracking: fixed
  Inclination: 27.2
  Orientation: 0
-----
Generator:
  Modules in series: 12
  Modules in parallel: 11
  Nominal power (kWp): 26.5

```

```
1 xyplot(prodFixed)
```

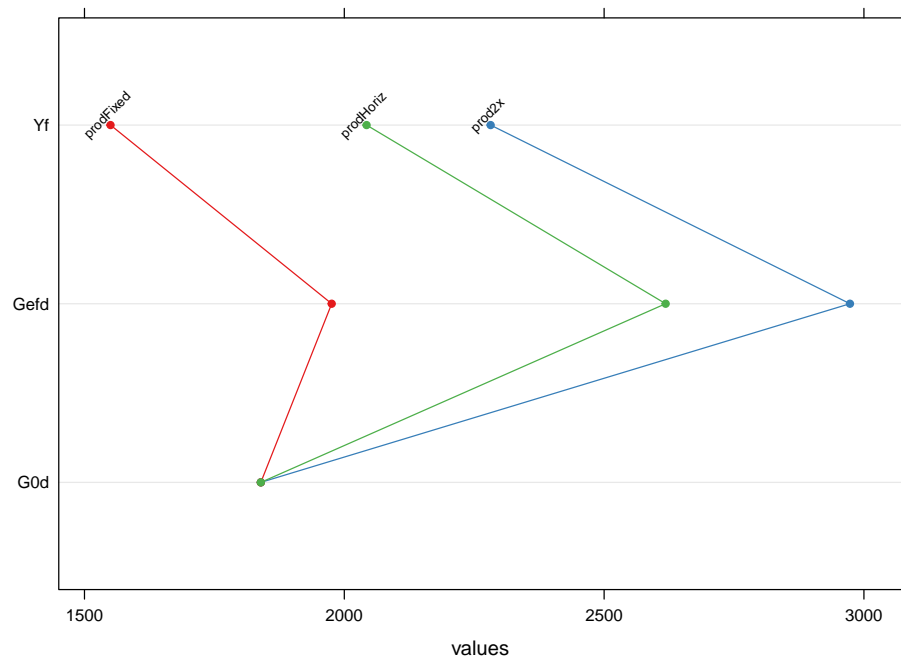


Un método para **compare**.

```

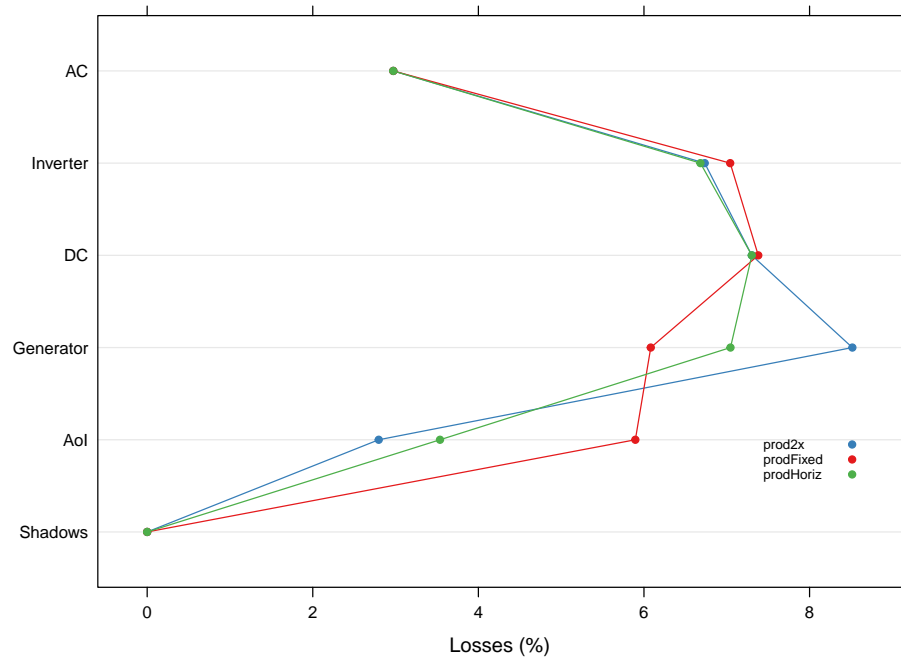
1 prod2x <- prodGCPV(lat, modeTrk = 'two', dataRad = BD)
2 prodHoriz <- prodGCPV(lat, modeTrk = 'horiz', dataRad = BD)
3 compare(prodFixed, prod2x, prodHoriz)

```



Y un método para `compareLosses`.

```
1 compareLosses(prodFixed, prod2x, prodHoriz)
```



4.8.5. Producción electrica de un SFB

La clase `ProdPVPS` cuenta con un método para `xyplot`.

```

1 pump <- prodPVPS(lat, dataRad = BD, pump = CoefSP8A44, H = 40, Pg = 5000)
2 show(pump)

```

```

Object of class  ProdPVPS

Source of meteorological information: prom-

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

Monthly avarages:
      Dates      Eac      Qd      Yf
      <char>    <num>    <num>    <num>
1: Jan. 2024 17.22642 59.71506 3.445284
2: Feb. 2024 18.89837 64.60949 3.779675
3: Mar. 2024 20.81307 70.36542 4.162613
4: Apr. 2024 23.73937 79.08382 4.747874
5: May. 2024 25.28080 83.74003 5.056160
6: Jun. 2024 26.49158 87.02474 5.298315
7: Jul. 2024 27.70317 89.81648 5.540633
8: Aug. 2024 27.14273 87.89528 5.428546
9: Sep. 2024 24.01466 79.04010 4.802932
10: Oct. 2024 18.26638 63.00860 3.653277
11: Nov. 2024 17.06794 59.03182 3.413588
12: Dec. 2024 13.72784 48.99686 2.745567

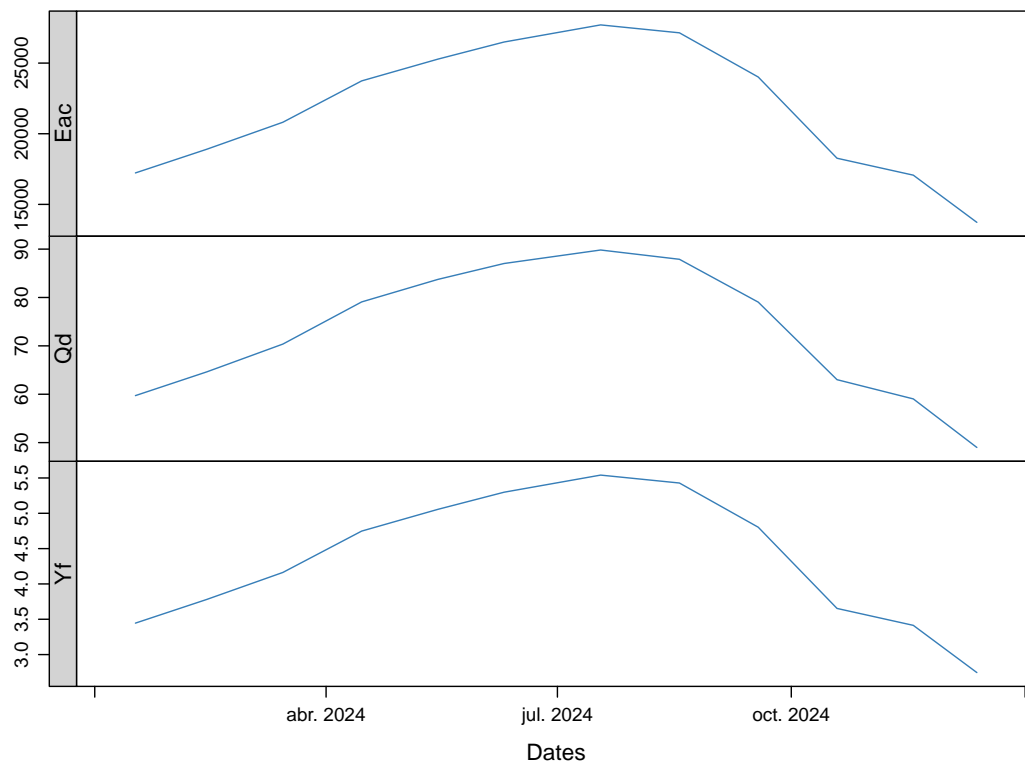
Yearly values:
      Dates      Eac      Qd      Yf
      <int>    <num>    <num>    <num>
1:  2024 7942.432 26608.76 1588.486
-----
Mode of tracking:  fixed
  Inclination:  27.2
  Orientation:   0
-----
Pump:
  Qn:  8
  Stages:  44
Height (m):  40
Generator (Wp): 5000

```

```

1 xyplot(pump)

```



#### 4.8.6. Optimización de distancias

La clase **Shade** cuenta con un método para **shadeplot**.

```

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 = lat, dataRad = prom, modeTrk = 'two',
4                   modeShd = c('area', 'prom'),
5                   distances = dist2x, struct = struct2x,
6                   res = 5, prog = FALSE)
7 show(ShdM2x)

```

```

Object of class Shade

Source of meteorological information: prom-

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

Monthly avarages:
Dimensions of structure:
$W
[1] 23.11

$L
[1] 9.8

$Nrow
[1] 2

$Ncol
[1] 3

Shade calculation mode:
[1] "area" "prom"

```



```
Productivity without shadows:
Object of class  ProdGCPV

Source of meteorological information: prom-

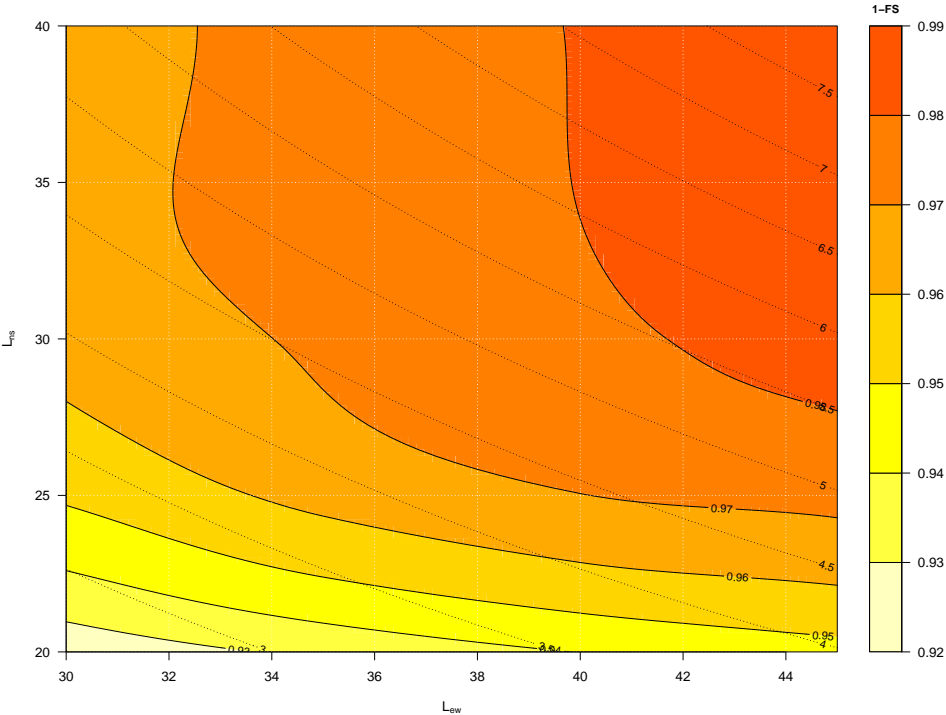
Latitude of source:  37.2 degrees
Latitude for calculations:  37.2 degrees

Monthly avarages:
      Dates      Eac      Edc      Yf
    <char>    <num>    <num>    <num>
1: Jan. 2024 138.6806 153.2566 5.241314
2: Feb. 2024 143.4987 158.5247 5.423408
3: Mar. 2024 151.8477 167.7311 5.738952
4: Apr. 2024 178.6717 197.4274 6.752741
5: May. 2024 200.8888 222.0523 7.592419
6: Jun. 2024 223.9959 247.6903 8.465728
7: Jul. 2024 214.2749 236.9628 8.098332
8: Aug. 2024 194.6043 215.1439 7.354902
9: Sep. 2024 168.9824 186.7349 6.386542
10: Oct. 2024 132.2995 146.0747 5.000145
11: Nov. 2024 128.5783 141.9871 4.859507
12: Dec. 2024 102.9116 113.5613 3.889454

Yearly values:
      Dates      Eac      Edc      Yf
    <int>    <num>    <num>    <num>
1:  2024 60369.04 66710.67 2281.595
-----
Mode of tracking: two
  Inclination limit: 90
-----
Generator:
  Modules in series: 12
  Modules in parallel: 11
  Nominal power (kWp): 26.5

Summary of results:
      Lew      Lns      H      FS      GRR      Yf
Min.   :30.00  Min.   :20  Min.   :0  Min.   :0.01509  Min.   :2.649  Min.   :2104
1st Qu.:33.75  1st Qu.:25  1st Qu.:0  1st Qu.:0.02223  1st Qu.:3.946  1st Qu.:2192
Median :37.50  Median :30  Median :0  Median :0.02870  Median :4.802  Median :2216
Mean   :37.50  Mean   :30  Mean   :0  Mean   :0.03463  Mean   :4.967  Mean   :2203
3rd Qu.:41.25  3rd Qu.:35  3rd Qu.:0  3rd Qu.:0.03945  3rd Qu.:6.016  3rd Qu.:2231
Max.   :45.00  Max.   :40  Max.   :0  Max.   :0.07769  Max.   :7.948  Max.   :2247
```

```
1 shadeplot(ShdM2x)
```



## Ejemplo práctico de aplicación

Una vez explicado como 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 para poder apreciar 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 inicilizando el paquete:

```
1 library(solaR)
```

En el estudio anterior, se recopilaron datos intradiarios de irradiación los cuales fueron almacenados en archivos.

```
1 enemar13 <- readBDi(file = 'TFG/data/ETSIDI/etsidi/EneMar2013_1.csv',
2                       lat = 40.4, time.col = 'Fecha')
3 show(enemar13)
```

Object of class Meteo

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013\_1.csv  
Latitude of source: 40.4 degrees

Meteorological Data:

Index	GO	Ta
Min. :2013-01-24 00:15:00.00	Min. : 0.0	Min. : -33.80
1st Qu.:2013-02-09 18:00:00.00	1st Qu.: 13.0	1st Qu.: 9.50
Median :2013-02-26 11:15:00.00	Median : 13.0	Median : 12.20
Mean :2013-02-26 11:19:20.84	Mean :113.2	Mean : 11.97
3rd Qu.:2013-03-15 04:30:00.00	3rd Qu.:135.0	3rd Qu.: 14.40
Max. :2013-03-31 23:45:00.00	Max. :755.0	Max. : 64.50

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.

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.

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

```

1 ## mc-Si

```

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
$\alpha_{Isc}$ (%/K)	0.0043	0.06
$\beta_{Voc}$ (%/K)	-0.338	-0.37
$\gamma_{Pmpp}$ (%/K)	-0.469	-0.45
Temperatura NOC (°C)	43.7	46

TABLA 5.2: *Sistemas fotovoltaicos.*

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

```

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.

Se almacena esta información en otra lista:

```

1 inverter <- list(Pinv = 1200,
2                 Vmin = 100,
3                 Vmax = 320)

```

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 = enemar13,
3                 beta = 30, alfa = -19, iS = 1,
4                 module = module1, generator = generator1,
5                 inverter = inverter)
6 show(prod1)

```

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 %

```
Object of class  ProdGCPV

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013_1.csv

Latitude of source:  40.4 degrees
Latitude for calculations:  40.4 degrees

Monthly averages:
      Eac      Edc      Yf
ene. 2013 2.288657 2.544214 1.829001
feb. 2013 2.912867 3.246235 2.327844
mar. 2013 2.642931 2.958194 2.112123

Yearly values:
      Eac      Edc      Yf
2013 181.8004 202.9523 145.2875
-----
Mode of tracking:  fixed
      Inclination:  30
      Orientation: -19
-----
Generator:
      Modules in series:  5
      Modules in parallel:  1
      Nominal power (kWp):  1.3
```

```
1 prod2 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
2                   dataRad = enemar13,
3                   beta = 30, alfa = -19, iS = 1,
4                   module = module2, generator = generator2,
5                   inverter = inverter)
6 show(prod2)
```

```
Object of class  ProdGCPV

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013_1.csv

Latitude of source:  40.4 degrees
Latitude for calculations:  40.4 degrees

Monthly averages:
      Eac      Edc      Yf
ene. 2013 1.995563 2.219924 1.813242
feb. 2013 2.546910 2.840829 2.314216
mar. 2013 2.324995 2.608686 2.112576

Yearly values:
      Eac      Edc      Yf
2013 159.3528 178.1718 144.7938
-----
Mode of tracking:  fixed
      Inclination:  30
      Orientation: -19
-----
Generator:
      Modules in series:  5
      Modules in parallel:  1
      Nominal power (kWp):  1.1
```

## 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 las tablas 5.4 y 5.5 se tienen los resultados de la simulación de los sistemas.

## 5.3. solar2

Con los datos obtenidos en la sección 5.1, hacemos la misma operación pero con el paquete **solar2**.

```
1 library(solar2)
```

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

```
1 enemar13 <- readBDi(file = 'TFG/data/ETSIDI/etsidi/EneMar2013_1.csv',
2                       lat = 40.4, dates.col = 'Fecha')
3 show(enemar13)
```

```
Object of class  Meteo

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013_1.csv
Latitude of source: 40.4 degrees

Meteorological Data:
      Dates              G0              Ta
Min.   :2013-01-24 00:15:00.00  Min.   : 0.0  Min.   : -33.80
1st Qu.:2013-02-09 18:00:00.00  1st Qu.: 13.0  1st Qu.:  9.50
Median :2013-02-26 11:15:00.00  Median : 13.0  Median : 12.20
Mean   :2013-02-26 11:19:20.84  Mean   :113.2  Mean   : 11.97
3rd Qu.:2013-03-15 04:30:00.00  3rd Qu.:135.0  3rd Qu.: 14.40
Max.   :2013-03-31 23:45:00.00  Max.   :755.0  Max.   : 64.50
```

TABLA 5.4: *Energía media mensual estimada por PVsyst en KWh del sistema 1.*

Ene	Feb	Mar	Abr	May	Jun	Jul	Ago	Sep	Oct	Nov	Dic	Total
3,7	4,0	5,6	5,3	6,7	6,7	7,9	7,2	6,4	4,8	3,5	3,6	1941,1

TABLA 5.5: *Energía media mensual estimada por PVsyst en KWh del sistema 2.*

Ene	Feb	Mar	Abr	May	Jun	Jul	Ago	Sep	Oct	Nov	Dic	Total
4,3	4,6	6,4	6,1	7,3	7,3	8,3	7,7	6,9	5,4	4,1	4,4	2213,7

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 = enemar13,
3                   beta = 30, alfa = -19, iS = 1,
4                   module = module1, generator = generator1,
5                   inverter = inverter)
6 show(prod1)
```

```
Object of class ProdGCPV

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013_1.csv

Latitude of source: 40.4 degrees
Latitude for calculations: 40.4 degrees

Monthly avarages:
      Dates      Eac      Edc      Yf
    <char>    <num>    <num>    <num>
1: Jan. 2013 2.288657 2.544214 1.829001
2: Feb. 2013 2.912867 3.246235 2.327844
3: Mar. 2013 2.642931 2.958194 2.112123

Yearly values:
      Dates      Eac      Edc      Yf
    <int>    <num>    <num>    <num>
1: 2013 181.8004 202.9523 145.2875
-----
Mode of tracking: fixed
  Inclination: 30
  Orientation: -19
-----
Generator:
  Modules in series: 5
  Modules in parallel: 1
  Nominal power (kWp): 1.3
```

```
1 prod2 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
2                   dataRad = enemar13,
3                   beta = 30, alfa = -19, iS = 1,
4                   module = module2, generator = generator2,
5                   inverter = inverter)
6 show(prod2)
```

```
Object of class ProdGCPV

Source of meteorological information: bdI-TFG/data/ETSIDI/etsidi/EneMar2013_1.csv

Latitude of source: 40.4 degrees
Latitude for calculations: 40.4 degrees

Monthly avarages:
      Dates      Eac      Edc      Yf
    <char>    <num>    <num>    <num>
1: Jan. 2013 1.995563 2.219924 1.813242
2: Feb. 2013 2.546910 2.840829 2.314216
3: Mar. 2013 2.324995 2.608686 2.112576

Yearly values:
      Dates      Eac      Edc      Yf
    <int>    <num>    <num>    <num>
```



```
1: 2013 159.3528 178.1718 144.7938
```

```
-----
Mode of tracking: fixed
Inclination: 30
Orientation: -19
-----
```

```
Generator:
Modules in series: 5
Modules in parallel: 1
Nominal power (kWp): 1.1
```

## 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. Sin embargo, el paquete **solaR2**, a parte de la corrección de algunos errores, presenta unas claras ventajas frente a su antecesor. Estas son:

- **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 prodGCPVcustom <- function(){
4   prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
5                     dataRad = enemar13, beta = 30, alfa = -19,
6                     iS = 1, module = module1,
7                     generator = generator1, inverter = inverter)
8 }
9 microbenchmark(prodGCPVcustom(), times = 50)
```

```
Unit: milliseconds
      expr      min       lq      mean   median       uq      max neval
prodGCPVcustom() 519.6561 529.6309 537.7213 533.5813 541.5372 588.5352    50
```

```
1 ## Con el paquete solaR2
2 library(microbenchmark)
3 prodGCPVcustom <- function(){
4   prod1 <- prodGCPV(lat = 40.4, modeTrk = 'fixed', modeRad = 'bdI',
5                     dataRad = enemar13, beta = 30, alfa = -19,
6                     iS = 1, module = module1,
7                     generator = generator1, inverter = inverter)
8 }
9 microbenchmark(prodGCPVcustom(), times = 50)
```

```
Unit: milliseconds
      expr      min       lq      mean   median       uq      max neval
prodGCPVcustom() 330.2599 333.759 340.0681 336.0833 340.389 375.635    50
```

Aquí se puede ver que la eficiencia mejora. Sin embargo, suponiendo que en vez de un sistema fijo, tuvieramos un sistema de seguimiento de doble eje y quisieramos obtener la mejor combinación de distancias, se podría utilizar la función **optimShd**, la cual al ser una tarea muy exigente se aprecia con más detalle las virtudes del paquete **solaR2** gracias al uso de **data.table**.

```

1 ## Con el paquete solarR
2 struct2x <- list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 3)
3 dist2x <- list(Lew = c(30, 45), Lns = c(20, 40))
4 optimShdcustom <- function(){
5   optim <- optimShd(lat = 40.4, modeTrk = 'two', modeRad = 'bdI',
6                     dataRad = enemar13, beta = 30, alfa = -19,
7                     iS = 1, module = module1,
8                     generator = generator1, inverter = inverter,
9                     modeShd = c('area', 'prom'),
10                    distances = dist2x, struct = struct2x,
11                    res = 5, prog = FALSE)
12 }
13 microbenchmark(optimShdcustom(), times = 20)

```

```

Unit: seconds
      expr      min       lq      mean   median      uq      max  neval
optimShdcustom() 6.229458 6.261952 6.297081 6.286866 6.322821 6.456526    20

```

```

1 ## Con el paquete solarR2
2 struct2x <- list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 3)
3 dist2x <- list(Lew = c(30, 45), Lns = c(20, 40))
4 optimShdcustom <- function(){
5   optim <- optimShd(lat = 40.4, modeTrk = 'two', modeRad = 'bdI',
6                     dataRad = enemar13, beta = 30, alfa = -19,
7                     iS = 1, module = module1,
8                     generator = generator1, inverter = inverter,
9                     modeShd = c('area', 'prom'),
10                    distances = dist2x, struct = struct2x,
11                    res = 5, prog = FALSE)
12 }
13 microbenchmark(optimShdcustom(), times = 20)

```

```

Unit: seconds
      expr      min       lq      mean   median      uq      max  neval
optimShdcustom() 5.037961 5.056003 5.115279 5.089934 5.099124 5.420153    20

```

## Código completo

Todo el código que se muestra a continuación está disponible para descargar, consultar y modificar libremente en el repositorio de Github correspondiente a la versión de desarrollo del paquete: <https://github.com/solarization/solaR2>.

### A.1. Constructores

#### A.1.1. 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) #daily values
9   soli <- fSoli(sold = sold, sample = sample, #intradaily values
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 }

```

EXTRACTO DE CÓDIGO A.1: *calcSol*

## A.1.2. calcG0

```

1  utils::globalVariables('DayOfMonth')
2
3  calcG0 <- function(lat,
4                    modeRad='prom',
5                    dataRad,
6                    sample='hour',
7                    keep.night=TRUE,
8                    sunGeometry='michalsky',
9                    corr, f, ...)
10 {
11
12     if (missing(lat)) stop('lat missing. You must provide a latitude value.')
13
14     stopifnot(modeRad %in% c('prom', 'aguilar', 'bd', 'bdI'))
15
16
17     ###Radiation data
18     if (missing(corr)){
19         corr = switch(modeRad,
20 values          bd = 'CPR', #Correlation between Fd and Kt for daily
21 values          aguilar = 'CPR', #Correlation between Fd and Kt for daily
22 values          prom = 'Page', #Correlation between Fd and Kt for monthly
23 averages        bdI = 'BRL' #Correlation between fd and kt for
intraday values
24                 )
25     }
26
27     if(is(dataRad, 'Meteo')){BD <- dataRad}
28     else{
29         BD <- switch(modeRad,
30 bd = {
31         if (!is.list(dataRad) || is.data.frame(dataRad)){
32             dataRad <- list(file = dataRad)
33         }
34         switch(class(dataRad$file)[1],
35 character = {
36             bd.default=list(file = '', lat = lat)
37             bd=modifyList(bd.default, dataRad)
38             res <- do.call('readBDd', bd)
39             res
40         },
41 data.table = ,
42 data.frame = {
43             bd.default = list(file = '', lat = lat)
44             bd=modifyList(bd.default, dataRad)
45             res <- do.call('dt2Meteo', bd)
46             res
47         },
48 zoo = {
49             bd.default = list(file = '', lat = lat,
50 source = '')
51             bd = modifyList(bd.default, dataRad)
52             res <- do.call('zoo2Meteo', bd)
53             res

```

```

53         })
54     }, #End of bd
55     prom = {
56         if (!is.list(dataRad)) dataRad <- list(G0dm = dataRad)
57         prom.default <- list(G0dm = numeric(), lat = lat)
58         prom = modifyList(prom.default, dataRad)
59         res <- do.call('readG0dm', prom)
60     }, #End of prom
61     aguiar = {
62         if (is.list(dataRad)) dataRad <- dataRad$G0dm
63         BTd <- fBTd(mode = 'serie')
64         sold <- fSold(lat, BTd)
65         G0d <- markovG0(dataRad, sold)
66         res <- dt2Meteo(G0d, lat = lat, source = 'aguiar')
67     }, #End of aguiar
68     bdI = {
69         if (!is.list(dataRad) || is.data.frame(dataRad)){
70             dataRad <- list(file = dataRad)
71         }
72         switch(class(dataRad$file)[1],
73             character = {
74                 bdI.default <- list(file = '', lat = lat)
75                 bdI <- modifyList(bdI.default, dataRad)
76                 res <- do.call('readBDi', bdI)
77                 res
78             },
79             data.table = ,
80             data.frame = {
81                 bdI.default <- list(file = '', lat = lat)
82                 bdI <- modifyList(bdI.default, dataRad)
83                 res <- do.call('dt2Meteo', bdI)
84                 res
85             },
86             zoo = {
87                 bdI.default <- list(file = '', lat = lat, source
= '')
88                 bdI <- modifyList(bdI.default, dataRad)
89                 res <- do.call('zoo2Meteo', bdI)
90                 res
91             },
92             stop('dataRad$file should be a character, a data.
table, a data.frame or a zoo.')
```

```

93         }) #End of btI
94     } #End of general switch
95 }
96
97
98 ### Angulos solares y componentes de irradiancia
99 if (modeRad == 'bdI') {
100     sol <- calcSol(lat, sample = sample,
101     BTi = indexD(BD), keep.night=keep.night, method=
sunGeometry)
102     compI <- fCompI(sol=sol, G0I=BD, corr=corr, f=f, ...)
103     compD <- compI[, lapply(.SD, P2E, sol@sample),
104     .SDcols = c('G0', 'D0', 'B0'),
105     by = truncDay(Dates)]
106     names(compD)[1] <- 'Dates'
107     names(compD)[-1] <- paste(names(compD)[-1], 'd', sep = '')

```

```

108     compD$Fd <- compD$D0d/compD$G0d
109     compD$Kt <- compD$G0d/as.data.tableD(sol)$Bo0d
110   } else { ##modeRad!='bdI'
111     sol <- calcSol(lat, indexD(BD), sample = sample,
112                   keep.night = keep.night, method = sunGeometry)
113     compD<-fCompD(sol=sol, G0d=BD, corr=corr, f, ...)
114     compI<-fCompI(sol=sol, compD=compD, ...)
115   }
116
117   ###Temperature
118
119   Ta = switch(modeRad,
120             bd = {
121               if (all(c("TempMax","TempMin") %in% names(BD@data))) {
122                 fTemp(sol, BD)
123               } else {
124                 if ("Ta" %in% names(getData(BD))) {
125                   data.table(Dates = indexD(sol),
126                             Ta = getData(BD)$Ta)
127                 } else {
128                   warning('No temperature information available!')
129                 }
130               }
131             },
132             bdI={
133               if ("Ta" %in% names(getData(BD))) {
134                 data.table(Dates = indexI(sol),
135                           Ta = getData(BD)$Ta)
136               } else {
137                 warning('No temperature information available!')
138               }
139             },
140             prom = {
141               if ("Ta" %in% names(getData(BD))) {
142                 data.table(Dates = indexD(sol),
143                           Ta = getData(BD)$Ta)
144               } else {
145                 warning('No temperature information available!')
146               }
147             },
148             aguiar={
149               Dates<-indexI(sol)
150               x <- as.Date(Dates)
151               ind.rep <- cumsum(c(1, diff(x) != 0))
152               data.table(Dates = Dates,
153                         Ta = getData(BD)$Ta[ind.rep])
154             }
155           )
156
157   ###Medias mensuales y anuales
158   nms <- c('G0d', 'D0d', 'B0d')
159   G0dm <- compD[, lapply(.SD/1000, mean, na.rm = TRUE),
160                 .SDcols = nms,
161                 by = .(month(Dates), year(Dates))]
162
163   if(modeRad == 'prom'){
164     G0dm[, DayOfMonth := DOM(G0dm)]
165     G0y <- G0dm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),

```

```

166         .SDcols = nms,
167         by = .(Dates = year)]
168     G0dm[, DayOfMonth := NULL]
169 } else{
170     G0y <- compD[, lapply(.SD/1000, sum, na.rm = TRUE),
171                     .SDcols = nms,
172                     by = .(Dates = year(Dates)))]
173 }
174 G0dm[, Dates := paste(month.abb[month], year, sep = '. ')]
175 G0dm[, c('month', 'year') := NULL]
176 setcolororder(G0dm, 'Dates')
177
178 ###Result
179 result <- new(Class='G0',
180               BD,          #G0 contains "Meteo"
181               sol,         #G0 contains 'Sol'
182               G0D=compD,   #results of fCompD
183               G0dm=G0dm,   #monthly means
184               G0y=G0y,     #yearly values
185               G0I=compI,   #results of fCompD
186               Ta=Ta        #ambient temperature
187               )
188     return(result)
189 }

```

EXTRACTO DE CÓDIGO A.2: *calcG0*

### A.1.3. *calcGef*

```

1  calcGef<-function(lat,
2                    modeTrk='fixed',      #c('two','horiz','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='',          #modeShd=c('area','bt','prom')
12                   struct=list(), #list(W=23.11, L=9.8, Nrow=2, Ncol=8),
13                   distances=data.table(),#data.table(Lew=40, Lns=30, H=0)){
14     ...){
15
16     stopifnot(is.list(struct), is.data.frame(distances))
17
18     if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
19       modeShd[which(modeShd=='bt')]='area'
20       warning('backtracking is only implemented for modeTrk=horiz')}
21
22     if (modeRad!='prev'){ #not use a prev calculation
23       radHoriz <- calcG0(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, 'G0')

```

```

30 }
31
32 ### Inclined and effective radiation
33 BT=("bt" %in% modeShd)
34 angGen <- fTheta(radHoriz, beta, alpha, modeTrk, betaLim, BT, struct,
35 distances)
36 inclin <- fInclin(radHoriz, angGen, iS, alb, horizBright, HCPV)
37
38 ### Daily, monthly and yearly values
39 by <- radHoriz@sample
40 nms <- c('Bo', 'Bn', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
41 nmsd <- paste(nms, 'd', sep = '')
42
43 if(radHoriz@type == 'prom'){
44   Gefdm <- inclin[, lapply(.SD/1000, P2E, by),
45                     .SDcols = nms,
46                     by = .(month(Dates), year(Dates))]
47   names(Gefdm)[-c(1,2)] <- nmsd
48   GefD <- Gefdm[, .SD*1000,
49                  .SDcols = nmsd,
50                  by = .(Dates = indexD(radHoriz))]
51
52   Gefdm[, DayOfMonth := DOM(Gefdm)]
53   Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
54                 .SDcols = nmsd,
55                 by = .(Dates = year)]
56   Gefdm[, DayOfMonth := NULL]
57 } else{
58   GefD <- inclin[, lapply(.SD, P2E, by),
59                 .SDcols = nms,
60                 by = .(Dates = truncDay(Dates))]
61   names(GefD)[-1] <- nmsd
62
63   Gefdm <- GefD[, lapply(.SD/1000, mean, na.rm = TRUE),
64                 .SDcols = nmsd,
65                 by = .(month(indexD(radHoriz)), year(indexD(radHoriz)))]
66   Gefy <- GefD[, lapply(.SD/1000, sum, na.rm = TRUE),
67                 .SDcols = nmsd,
68                 by = .(Dates = year(indexD(radHoriz)))]
69 }
70
71 Gefdm[, Dates := paste(month.abb[month], year, sep = '. ')]
72 Gefdm[, c('month', 'year') := NULL]
73 setcolorder(Gefdm, 'Dates')
74
75 ###Resultado antes de sombras
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 ###Shadows
92     if (isTRUE(modeShd == "") ||           #If modeShd==' ' there is no shadow
        calculation
93         ('bt' %in% modeShd)) {             #nor if there is backtracking
94         return(result0)
95     } else {
96         result <- calcShd(result0, modeShd, struct, distances)
97         return(result)
98     }
99 }

```

EXTRACTO DE CÓDIGO A.3: *calcGef*

#### A.1.4. prodGCPV

```

1  utils::globalVariables(c('Yf', 'Eac'))
2
3  prodGCPV<-function(lat,
4                      modeTrk='fixed',
5                      modeRad='prom',
6                      dataRad,
7                      sample='hour',
8                      keep.night=TRUE,
9                      sunGeometry='michalsky',
10                     corr, f,
11                     betaLim=90, beta=abs(lat)-10, alpha=0,
12                     iS=2, alb=0.2, horizBright=TRUE, HCPV=FALSE,
13                     module=list(),
14                     generator=list(),
15                     inverter=list(),
16                     effSys=list(),
17                     modeShd=' ',
18                     struct=list(),
19                     distances=data.table(),
20                     ...){
21
22     stopifnot(is.list(module),
23              is.list(generator),
24              is.list(inverter),
25              is.list(effSys),
26              is.list(struct),
27              is.data.table(distances))
28
29     if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
30         modeShd[which(modeShd=='bt')]='area'
31         warning('backtracking is only implemented for modeTrk=horiz')}
32
33     if (modeRad!='prev'){ #We do not use a previous calculation
34
35         radEf<-calcGef(lat=lat, modeTrk = modeTrk, modeRad = modeRad,
36                       dataRad = dataRad,
37                       sample = sample, keep.night = keep.night,
38                       sunGeometry = sunGeometry,
39                       corr = corr, f = f,

```

```

40         betaLim = betaLim, beta = beta, alpha = alpha,
41         iS = iS, alb = alb, horizBright = horizBright, HCPV = HCPV,
42         modeShd = modeShd, struct = struct, distances = distances,
43         ...)
44 } else { #We use a previous calcG0, calcGef or prodGCPV calculation.
45
46     stopifnot(class(dataRad) %in% c('G0', 'Gef', 'ProdGCPV'))
47     radEf <- switch(class(dataRad),
48                     G0 = calcGef(lat = lat,
49                                 modeTrk = modeTrk, modeRad = 'prev',
50                                 dataRad = dataRad,
51                                 betaLim = betaLim, beta = beta, alpha = alpha
52                                 ,
53                                 iS = iS, alb = alb, horizBright = horizBright
54                                 , HCPV = HCPV,
55                                 modeShd = modeShd, struct = struct, distances
56                                 = distances, ...),
57                     Gef = dataRad,
58                     ProdGCPV = as(dataRad, 'Gef')
59                     )
60 }
61
62 ##Production
63 prodI <- fProd(radEf,module,generator,inverter,effSys)
64 module = attr(prodI, 'module')
65 generator = attr(prodI, 'generator')
66 inverter = attr(prodI, 'inverter')
67 effSys = attr(prodI, 'effSys')
68
69 ##Calculation of daily, monthly and annual values
70 Pg = generator$Pg #Wp
71
72 by <- radEf@sample
73 nms1 <- c('Pac', 'Pdc')
74 nms2 <- c('Eac', 'Edc', 'Yf')
75
76 if(radEf@type == 'prom'){
77     prodDm <- prodI[, lapply(.SD/1000, P2E, by),
78                        .SDcols = nms1,
79                        by = .(month(Dates), year(Dates))]
80     names(prodDm)[-c(1,2)] <- nms2[-3]
81     prodDm[, Yf := Eac/(Pg/1000)]
82     prodD <- prodDm[, .SD*1000,
83                      .SDcols = nms2,
84                      by = .(Dates = indexD(radEf))]
85     prodD[, Yf := Yf/1000]
86
87     prodDm[, DayOfMonth := DOM(prodDm)]
88     prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
89                     .SDcols = nms2,
90                     by = .(Dates = year)]
91     prodDm[, DayOfMonth := NULL]
92 } else {
93     prodD <- prodI[, lapply(.SD, P2E, by),
94                      .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 prodDm[, Dates := paste(month.abb[month], year, sep = '. ')]
109 prodDm[, c('month', 'year') := NULL]
110 setcolororder(prodDm, 'Dates')
111
112 result <- new('ProdGCPV',
113              radEf,                      #contains 'Gef'
114              prodD=prodD,
115              prodDm=prodDm,
116              prody=prody,
117              prodI=prodI,
118              module=module,
119              generator=generator,
120              inverter=inverter,
121              effSys=effSys
122              )
123 }

```

EXTRACTO DE CÓDIGO A.4: *prodGCPV*

### A.1.5. prodPVPS

```

1  utils::globalVariables('Qd')
2
3  prodPVPS<-function(lat,
4                    modeTrk='fixed',
5                    modeRad='prom',
6                    dataRad,
7                    sample='hour',
8                    keep.night=TRUE,
9                    sunGeometry='michalsky',
10                   corr, f,
11                   betaLim=90, beta=abs(lat)-10, alpha = 0,
12                   iS=2, alb=0.2, horizBright=TRUE, HCPV=FALSE,
13                   pump , H,
14                   Pg, converter= list(), #Pnom=Pg, Ki=c(0.01,0.025,0.05)),
15                   effSys=list(),
16                   ...){
17
18     stopifnot(is.list(converter),
19              is.list(effSys))
20
21     if (modeRad!='prev'){ #We do not use a previous calculation
22
23       radEf<-calcGef(lat=lat, modeTrk=modeTrk, modeRad=modeRad,

```

```

24         dataRad=dataRad,
25         sample=sample, keep.night=keep.night,
26         sunGeometry=sunGeometry,
27         corr=corr, f=f,
28         betaLim=betaLim, beta=beta, alpha = alpha,
29         iS=iS, alb=alb, horizBright=horizBright, HCPV=HCPV,
30         modeShd='', ...)
31
32     } else { #We use a previous calculation of calcG0, calcGef or prodPVPS
33         stopifnot(class(dataRad) %in% c('G0', 'Gef', 'ProdPVPS'))
34         radEf <- switch(class(dataRad),
35             G0=calcGef(lat=lat,
36                 modeTrk=modeTrk, modeRad='prev',
37                 dataRad=dataRad,
38                 betaLim=betaLim, beta=beta, alpha = alpha,
39                 iS=iS, alb=alb, horizBright=horizBright, HCPV=HCPV,
40                 modeShd='', ...),
41             Gef=dataRad,
42             ProdPVPS=as(dataRad, 'Gef')
43         )
44     }
45
46     ###Electric production
47     converter.default=list(Ki = c(0.01,0.025,0.05), Pnom=Pg)
48     converter=modifyList(converter.default, converter)
49
50     effSys.default=list(ModQual=3,ModDisp=2,OhmDC=1.5,OhmAC=1.5,MPP=1,TrafoMT=1,
51         Disp=0.5)
52     effSys=modifyList(effSys.default, effSys)
53
54     TONC=47
55     Ct=(TONC-20)/800
56     lambda=0.0045
57     Gef=radEf@GefI$Gef
58     night=radEf@solI$night
59     Ta=radEf@Ta$Ta
60
61     Tc=Ta+Ct*Gef
62     Pdc=Pg*Gef/1000*(1-lambda*(Tc-25))
63     Pdc[is.na(Pdc)]=0 #Necessary for the functions provided by fPump
64     PdcN=with(effSys,
65         Pdc/converter$Pnom*(1-ModQual/100)*(1-ModDisp/100)*(1-OhmDC/100)
66     )
67     PacN=with(converter,{
68         A=Ki[3]
69         B=Ki[2]+1
70         C=Ki[1]-(PdcN)
71         ##AC power normalized to the inverter
72         result=(-B+sqrt(B^2-4*A*C))/(2*A)
73     })
74     PacN[PacN<0]<-0
75
76     Pac=with(converter,
77         PacN*Pnom*(1-effSys$OhmAC/100))
78     Pdc=PdcN*converter$Pnom*(Pac>0)
79
80     ###Pump

```

```

81 fun<-fPump(pump=pump, H=H)
82 ##I limit power to the pump operating range.
83 rango=with(fun,Pac>=lim[1] & Pac<=lim[2])
84 Pac[!rango]<-0
85 Pdc[!rango]<-0
86 prodI=data.table(Pac=Pac,Pdc=Pdc,Q=0,Pb=0,Ph=0,f=0)
87 prodI=within(prodI,{
88   Q[rango]<-fun$fQ(Pac[rango])
89   Pb[rango]<-fun$fPb(Pac[rango])
90   Ph[rango]<-fun$fPh(Pac[rango])
91   f[rango]<-fun$fFreq(Pac[rango])
92   etam=Pb/Pac
93   etab=Ph/Pb
94 })
95
96 prodI[night,]<-NA
97 prodI[, Dates := indexI(radEf)]
98 setcolorder(prodI, c('Dates', names(prodI)[-length(prodI)]))
99
100 ###daily, monthly and yearly values
101
102 by <- radEf@sample
103
104 if(radEf@type == 'prom'){
105   prodDm <- prodI[, .(Eac = P2E(Pac, by)/1000,
106     Qd = P2E(Q, by)),
107     by = .(month(Dates), year(Dates))]
108   prodDm[, Yf := Eac/(Pg/1000)]
109
110   prodD <- prodDm[, .(Eac = Eac*1000,
111     Qd,
112     Yf),
113     by = .(Dates = indexD(radEf))]
114
115   prodDm[, DayOfMonth := DOM(prodDm)]
116
117   prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
118     .SDcols = c('Eac', 'Qd', 'Yf'),
119     by = .(Dates = year)]
120   prodDm[, DayOfMonth := NULL]
121 } else {
122   prodD <- prodI[, .(Eac = P2E(Pac, by)/1000,
123     Qd = P2E(Q, by)),
124     by = .(Dates = truncDay(Dates))]
125   prodD[, Yf := Eac/Pg*1000]
126
127   prodDm <- prodD[, lapply(.SD, mean, na.rm = TRUE),
128     .SDcols = c('Eac', 'Qd', 'Yf'),
129     by = .(month(Dates), year(Dates))]
130   prody <- prodD[, lapply(.SD, sum, na.rm = TRUE),
131     .SDcols = c('Eac', 'Qd', 'Yf'),
132     by = .(Dates = year(Dates))]
133
134 }
135
136 prodDm[, Dates := paste(month.abb[month], year, sep = '. ')]
137 prodDm[, c('month', 'year') := NULL]
138 setcolorder(prodDm, 'Dates')

```

```

139
140   result <- new('ProdPVPS',
141                radEf,                #contains 'Gef'
142                prodD=prodD,
143                prodDm=prodDm,
144                prody=prody,
145                prodI=prodI,
146                pump=pump,
147                H=H,
148                Pg=Pg,
149                converter=converter,
150                effSys=effSys
151                )
152 }

```

EXTRACTO DE CÓDIGO A.5: *prodGCPV*A.1.6. *calcShd*

```

1  calcShd<-function(radEf,##class='Gef'
2      modeShd='prom',      #modeShd=c('area','bt','prom')
3      struct=list(), #list(W=23.11, L=9.8, Nrow=2, Ncol=8),
4      distances=data.table() #data.table(Lew=40, Lns=30, H=0)){
5      )
6  {
7      stopifnot(is.list(struct), is.data.frame(distances))
8
9      ##For now I only use modeShd = 'area'
10     ##With different modeShd (to be defined) I will be able to calculate Gef in
    a different way
11     ##See macagnan thesis
12     prom=("prom" %in% modeShd)
13     prev <- as.data.tableI(radEf, complete=TRUE)
14     ## shadow calculations
15     modeTrk <- radEf@modeTrk
16     sol <- data.table(AzS = prev$AzS,
17                      ALS = prev$ALS)
18     theta <- radEf@Theta
19     AngGen <- data.table(theta, sol)
20     FS <- fSombra(AngGen, distances, struct, modeTrk, prom)
21     ## irradiance calculation
22     gef0 <- radEf@GefI
23     Bef0 <- gef0$Bef
24     Dcef0 <- gef0$Dcef
25     Gef0 <- gef0$Gef
26     Dief0 <- gef0$Dief
27     Ref0 <- gef0$Ref
28     ## calculation
29     Bef <- Bef0*(1-FS)
30     Dcef <- Dcef0*(1-FS)
31     Def <- Dief0+Dcef
32     Gef <- Dief0+Ref0+Bef+Dcef #Including shadows
33     ##Change names
34     nms <- c('Gef', 'Def', 'Dcef', 'Bef')
35     nmsIndex <- which(names(gef0) %in% nms)
36     names(gef0)[nmsIndex] <- paste(names(gef0)[nmsIndex], '0', sep='')
37     GefShd <- gef0
38     GefShd[, c(nms, 'FS')] := .(Gef, Def, Dcef, Bef, FS)]

```

```

39
40 ## daily, monthly and yearly values
41 by <- radEf@sample
42 nms <- c('Gef0', 'Def0', 'Bef0', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
43 nmsd <- paste(nms, 'd', sep = '')
44
45 Gefdm <- GefShd[, lapply(.SD/1000, P2E, by),
46                     by = .(month(truncDay(Dates)), year(truncDay(Dates))),
47                     .SDcols = nms]
48 names(Gefdm)[-c(1, 2)] <- nmsd
49
50 if(radEf@type == 'prom'){
51   GefD <- Gefdm[, .SD[, -c(1, 2)] * 1000,
52                 .SDcols = nmsd,
53                 by = .(Dates = indexD(radEf))]
54
55   Gefdm[, DayOfMonth := DOM(Gefdm)]
56
57   Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
58                 .SDcols = nmsd,
59                 by = .(Dates = year)]
60   Gefdm[, DayOfMonth := NULL]
61 } else{
62   GefD <- GefShd[, lapply(.SD/1000, P2E, by),
63                 .SDcols = nms,
64                 by = .(Dates = truncDay(Dates))]
65   names(GefD)[-1] <- nmsd
66
67   Gefy <- GefD[, lapply(.SD[, -1], sum, na.rm = TRUE),
68                 .SDcols = nmsd,
69                 by = .(Dates = year(Dates))]
70 }
71
72 Gefdm[, Dates := paste(month.abb[month], year, sep = '. ')]
73 Gefdm[, c('month', 'year') := NULL]
74 setcolorder(Gefdm, c('Dates', names(Gefdm)[-length(Gefdm)]))
75
76 ## Object of class Gef
77 ## modifying the 'modeShd', 'GefI', 'GefD', 'Gefdm', and 'Gefy' slots
78 ## from the original radEf object
79 radEf@modeShd=modeShd
80 radEf@GefI=GefShd
81 radEf@GefD=GefD
82 radEf@Gefdm=Gefdm
83 radEf@Gefy=Gefy
84 return(radEf)
85 }

```

EXTRACTO DE CÓDIGO A.6: *calcShd*

### A.1.7. 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,      #resolution, distance spacing
18      prog=TRUE){ #Drawing progress bar
19
20      if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
21          modeShd[which(modeShd=='bt')]='area'
22          warning('backtracking is only implemented for modeTrk=horiz')}
23
24      ##I save function arguments for later use
25
26      listArgs<-list(lat=lat, modeTrk=modeTrk, modeRad=modeRad,
27                    dataRad=dataRad,
28                    sample=sample, keep.night=keep.night,
29                    sunGeometry=sunGeometry,
30                    betaLim=betaLim, beta=beta, alpha = alpha,
31                    iS=iS, alb=alb, HCPV=HCPV,
32                    module=module, generator=generator,
33                    inverter=inverter, effSys=effSys,
34                    modeShd=modeShd, struct=struct,
35                    distances=data.table(Lew=NA, Lns=NA, D=NA))
36
37
38      ##Create network on which I will do the calculations
39      Red=switch(modeTrk,
40                horiz=with(distances,
41                            data.table(Lew=seq(Lew[1],Lew[2],by=res),
42                                          H=0)),
43                two=with(distances,
44                          data.table(
45                            expand.grid(Lew=seq(Lew[1],Lew[2],by=res),
46                                              Lns=seq(Lns[1],Lns[2],by=res),
47                                              H=0))),
48                fixed=with(distances,
49                            data.table(D=seq(D[1],D[2],by=res),
50                                          H=0))
51      )
52
53      casos<-dim(Red)[1] #Number of possibilities to study
54
55      ##I prepare the progress bar
56      if (prog) {pb <- txtProgressBar(min = 0, max = casos+1, style = 3)
57                setTxtProgressBar(pb, 0)}
58
59      ###Calculations
60      ##Reference: No shadows
61      listArgs0 <- modifyList(listArgs,
62                             list(modeShd='', struct=NULL, distances=NULL) )
63      Prod0<-do.call(prodGCPV, listArgs0)
64      YfAnual0=mean(Prod0@prody$Yf) #I use mean in case there are several years

```



```

65   if (prog) {setTxtProgressBar(pb, 1)}
66
67   ##The loop begins
68
69   ##I create an empty vector of the same length as the cases to be studied
70   YfAnnual<-numeric(casos)
71
72   BT=('bt' %in% modeShd)
73   if (BT) { ##There is backtracking, then I must start from horizontal
radiation.
74       RadBT <- as(Prod0, 'GO')
75       for (i in seq_len(casos)){
76           listArgsBT <- modifyList(listArgs,
77                                   list(modeRad='prev', dataRad=RadBT,
78                                       distances=Red[i,]))
79           prod.i <- do.call(prodGCPV, listArgsBT)
80           YfAnnual[i]=mean(prod.i@prody$Yf)
81           if (prog) {setTxtProgressBar(pb, i+1)}
82       }
83   } else {
84       prom=('prom' %in% modeShd)
85       for (i in seq_len(casos)){
86           Gef0=as(Prod0, 'Gef')
87           GefShd=calcShd(Gef0, modeShd=modeShd,
88                          struct=struct, distances=Red[i,])
89           listArgsShd <- modifyList(listArgs,
90                                   list(modeRad='prev', dataRad=GefShd)
91                                   )
92           prod.i <- do.call(prodGCPV, listArgsShd)
93           YfAnnual[i]=mean(prod.i@prody$Yf)
94           if (prog) {setTxtProgressBar(pb, i+1)}
95       }
96   }
97   if (prog) {close(pb)}
98
99
100  ###Results
101  FS=1-YfAnnual/YfAnnual0
102  GRR=switch(modeTrk,
103             two=with(Red,Lew*Lns)/with(struct,L*W),
104             fixed=Red$D/struct$L,
105             horiz=Red$Lew/struct$L)
106  SombraDF=data.table(Red,GRR,FS,Yf=YfAnnual)
107  FS.loess=switch(modeTrk,
108                 two=loess(FS~Lew*Lns,data=SombraDF),
109                 horiz=loess(FS~Lew,data=SombraDF),
110                 fixed=loess(FS~D,data=SombraDF))
111  Yf.loess=switch(modeTrk,
112                 two=loess(Yf~Lew*Lns,data=SombraDF),
113                 horiz=loess(Yf~Lew,data=SombraDF),
114                 fixed=loess(Yf~D,data=SombraDF))
115  result <- new('Shade',
116               Prod0, ##contains ProdGCPV
117               FS=FS,
118               GRR=GRR,
119               Yf=YfAnnual,
120               FS.loess=FS.loess,
121               Yf.loess=Yf.loess,

```

```

122         modeShd=modeShd,
123         struct=struct,
124         distances=Red,
125         res=res
126     )
127     result
128 }

```

EXTRACTO DE CÓDIGO A.7: *optimShd*A.1.8. *meteoReaders*

```

1  utils::globalVariables(c('.', 'Dates', 'Bo0d', 'Bo0m', 'G0d',
2                          'Bo0', 'Ta', '..cols', 'est_SIAR',
3                          'Fecha_Instalacion', 'Fecha_Baja',
4                          'Latitud', 'Longitud', 'peso', 'Estacion',
5                          'Codigo', 'req_url_query', 'req_url_path',
6                          'request', 'req_perform', 'resp_body_json',
7                          'HoraMin', 'Fecha', 'Radiacion', 'TempMin',
8                          'TempMax', 'TempMedia', 'Year', 'Mes', 'index'))
9
10 ##### monthly means of irradiation #####
11 readG0dm <- function(G0dm, Ta = 25, lat = 0,
12                      year = as.POSIXlt(Sys.Date())$year + 1900,
13                      promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18,
14                                  13),
15                      source = '')
16 {
17     if(missing(lat)){lat <- 0}
18     Dates <- as.IDate(paste(year[1], 1:12, promDays, sep = '-'), tz = 'UTC')
19     if (length(year)>1){
20         for (i in year[-1]){
21             x <- as.IDate(paste(i, 1:12, promDays, sep = '-'), tz = 'UTC')
22             Dates <- c(Dates, x)
23         }
24     }
25     G0dm.dt <- data.table(Dates = Dates,
26                          G0d = G0dm,
27                          Ta = Ta)
28     setkey(G0dm.dt, 'Dates')
29     results <- new(Class = 'Meteo',
30                   latm = lat,
31                   data = G0dm.dt,
32                   type = 'prom',
33                   source = source)
34 }
35 ##### file to Meteo (daily) #####
36 readBDd <- function(file, lat,
37                     format = "%d/%m/%Y", header = TRUE,
38                     fill = TRUE, dec = '.', sep = ';',
39                     dates.col = 'Dates', ta.col = 'Ta',
40                     g0.col = 'G0', keep.cols = FALSE, ...)
41 {
42     #stops if the arguments are not characters or numerics
43     stopifnot(is.character(dates.col) || is.numeric(dates.col))
44     stopifnot(is.character(ta.col) || is.numeric(ta.col))
45     stopifnot(is.character(g0.col) || is.numeric(g0.col))

```

```

46 #read from file and set it in a data.table
47 bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep, ...)
48
49
50 if(dates.col == ''){
51   names(bd)[1] <- 'Dates'
52   dates.col <- 'Dates'
53 }
54
55 #check the columns
56 if(!(dates.col %in% names(bd))) stop(paste('The column', dates.col, 'is not
in the file'))
57 if(!(g0.col %in% names(bd))) stop(paste('The column', g0.col, 'is not in
the file'))
58 if(!(ta.col %in% names(bd))) stop(paste('The column', ta.col, 'is not in
the file'))
59
60 #name the dates column by Dates
61 Dates <- bd[[dates.col]]
62 bd[, (dates.col) := NULL]
63 bd[, Dates := as.IDate(Dates, format = format)]
64
65 #name the g0 column by G0
66 G0 <- bd[[g0.col]]
67 bd[, (g0.col) := NULL]
68 bd[, G0 := as.numeric(G0)]
69
70 #name the ta column by Ta
71 Ta <- bd[[ta.col]]
72 bd[, (ta.col) := NULL]
73 bd[, Ta := as.numeric(Ta)]
74
75 names0 <- NULL
76 if(all(c('D0', 'B0') %in% names(bd))){
77   names0 <- c(names0, 'D0', 'B0')
78 }
79
80 names0 <- c(names0, 'Ta')
81
82 if(all(c('TempMin', 'TempMax') %in% names(bd))){
83   names0 <- c(names0, 'TempMin', 'TempMax')
84 }
85 if(keep.cols)
86 {
87   #keep the rest of the columns but reorder the columns
88   setcolorder(bd, c('Dates', 'G0', names0))
89 }
90 else
91 {
92   #erase the rest of the columns
93   cols <- c('Dates', 'G0', names0)
94   bd <- bd[, ..cols]
95 }
96
97 setkey(bd, 'Dates')
98 result <- new(Class = 'Meteo',
99               latm = lat,
100              data = bd,

```

```

101         type = 'bd',
102         source = file)
103     }
104
105     ##### file to Meteo (intradaily) #####
106     readBDi <- function(file, lat,
107                         format = "%d/%m/%Y %H:%M:%S",
108                         header = TRUE, fill = TRUE, dec = '.',
109                         sep = ';', dates.col = 'Dates', times.col,
110                         ta.col = 'Ta', g0.col = 'G0', keep.cols = FALSE, ...)
111     {
112         #stops if the arguments are not characters or numerics
113         stopifnot(is.character(dates.col) || is.numeric(dates.col))
114         stopifnot(is.character(ta.col) || is.numeric(ta.col))
115         stopifnot(is.character(g0.col) || is.numeric(g0.col))
116
117         #read from file and set it in a data.table
118         bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep, ...)
119
120         if(dates.col == ''){
121             names(bd)[1] <- 'Dates'
122             dates.col <- 'Dates'
123         }
124
125         #check the columns
126         if(!(dates.col %in% names(bd))) stop(paste('The column', dates.col, 'is not
127             in the file'))
128         if(!(g0.col %in% names(bd))) stop(paste('The column', g0.col, 'is not in
129             the file'))
130         if(!(ta.col %in% names(bd))) stop(paste('The column', ta.col, 'is not in
131             the file'))
132
133         if(!missing(times.col)){
134             stopifnot(is.character(times.col) || is.numeric(times.col))
135             if(!(times.col %in% names(bd))) stop(paste('The column', times.col, 'is
136                 not in the file'))
137
138             #name the dates column by Dates
139             format <- strsplit(format, ' ')
140             dd <- as.IDate(bd[[dates.col]], format = format[[1]][1])
141             tt <- as.ITime(bd[[times.col]], format = format[[1]][2])
142             bd[, (dates.col) := NULL]
143             bd[, (times.col) := NULL]
144             bd[, Dates := as.POSIXct(dd, tt, tz = 'UTC')]
145         }
146
147         else
148         {
149             dd <- as.POSIXct(bd[[dates.col]], format = format, tz = 'UTC')
150             bd[, (dates.col) := NULL]
151             bd[, Dates := dd]
152         }
153
154         #name the g0 column by G0
155         G0 <- bd[[g0.col]]
156         bd[, (g0.col) := NULL]
157         bd[, G0 := as.numeric(G0)]

```

```

155 #name the ta column by Ta
156 Ta <- bd[[ta.col]]
157 bd[, (ta.col) := NULL]
158 bd[, Ta := as.numeric(Ta)]
159
160 names0 <- NULL
161 if(all(c('D0', 'B0') %in% names(bd))){
162   names0 <- c(names0, 'D0', 'B0')
163 }
164
165 names0 <- c(names0, 'Ta')
166
167 if(keep.cols)
168 {
169   #keep the rest of the columns but reorder the columns
170   setcolorder(bd, c('Dates', 'G0', names0))
171 }
172 else
173 {
174   #erase the rest of the columns
175   cols <- c('Dates', 'G0', names0)
176   bd <- bd[, ..cols]
177 }
178
179 setkey(bd, 'Dates')
180 result <- new(Class = 'Meteo',
181               latm = lat,
182               data = bd,
183               type = 'bdI',
184               source = file)
185 }
186
187
188 dt2Meteo <- function(file, lat, source = '', type){
189   if(missing(lat)) stop('lat is missing')
190
191   if(source == '') source <- class(file)[1]
192
193   ## Make sure its a data.table
194   bd <- data.table(file)
195
196   ## Dates is an as.POSIX element
197   bd[, Dates := as.POSIXct(Dates, tz = 'UTC')]
198
199   ## type
200   if(missing(type)){
201     sample <- median(diff(bd$Dates))
202     IsDaily <- as.numeric(sample, units = 'days')
203     if(is.na(IsDaily)) IsDaily <- ifelse('G0d' %in% names(bd),
204                                         1, 0)
205     if(IsDaily >= 30) type <- 'prom'
206     else{
207       type <- ifelse(IsDaily >= 1, 'bd', 'bdI')
208     }
209   }
210
211   ## Columns of the data.table
212   nms0 <- switch(type,

```

```

213         bd = ,
214         prom = {
215             nms0 <- 'G0d'
216             if(all(c('D0d', 'B0d') %in% names(bd))){
217                 nms0 <- c(nms0, 'D0d', 'B0d')
218             }
219             if('Ta' %in% names(bd)) nms0 <- c(nms0, 'Ta')
220             if(all(c('TempMin', 'TempMax') %in% names(bd))){
221                 nms0 <- c(nms0, 'TempMin', 'TempMax')
222             }
223             nms0
224         },
225         bdI = {
226             nms0 <- 'G0'
227             if(all(c('D0', 'B0') %in% names(bd))){
228                 nms0 <- c(nms0, 'D0', 'B0')
229             }
230             if('Ta' %in% names(bd)) nms0 <- c(nms0, 'Ta')
231             nms0
232         })
233     ## Columns order and set key
234     setcolorder(bd, c('Dates', nms0))
235     setkey(bd, 'Dates')
236     ## Result
237     result <- new(Class = 'Meteo',
238                   latm = lat,
239                   data = bd,
240                   type = type,
241                   source = source)
242
243     if(!('Ta' %in% names(bd))){
244         if(all(c('TempMin', 'TempMax') %in% names(bd))){
245             sol <- calcSol(lat = lat, BTi = indexD(result))
246             bd[, Ta := fTemp(sol, result)$Ta]
247         }
248         else bd[, Ta := 25]
249         result@data <- bd
250     }
251     return(result)
252 }
253
254 ##### Liu and Jordan, Collares-Pereira and Rabl proposals #####
255 collper <- function(sol, compD)
256 {
257     Dates <- indexI(sol)
258     x <- as.Date(Dates)
259     ind.rep <- cumsum(c(1, diff(x) != 0))
260     solI <- as.data.tableI(sol, complete = T)
261     ws <- solI$ws
262     w <- solI$w
263
264     a <- 0.409-0.5016*sin(ws+pi/3)
265     b <- 0.6609+0.4767*sin(ws+pi/3)
266
267     rd <- solI[, Bo0/Bo0d]
268     rg <- rd * (a + b * cos(w))
269
270     # Daily irradiation components

```

```

271   G0d <- compD$G0d[ind.rep]
272   B0d <- compD$B0d[ind.rep]
273   D0d <- compD$D0d[ind.rep]
274
275   # Daily profile
276   G0 <- G0d * rg
277   D0 <- D0d * rd
278
279   # This method may produce diffuse irradiance higher than
280   # global irradiance
281   G0 <- pmax(G0, D0, na.rm = TRUE)
282   B0 <- G0 - D0
283
284   # Negative values are set to NA
285   neg <- (B0 < 0) | (D0 < 0) | (G0 < 0)
286   is.na(G0) <- neg
287   is.na(B0) <- neg
288   is.na(D0) <- neg
289
290   # Daily profiles are scaled to keep daily irradiation values
291   day <- truncDay(indexI(sol))
292   sample <- sol@sample
293
294   G0dCP <- ave(G0, day, FUN=function(x) P2E(x, sample))
295   B0dCP <- ave(B0, day, FUN=function(x) P2E(x, sample))
296   D0dCP <- ave(D0, day, FUN=function(x) P2E(x, sample))
297
298   G0 <- G0 * G0d/G0dCP
299   B0 <- B0 * B0d/B0dCP
300   D0 <- D0 * D0d/D0dCP
301
302   res <- data.table(G0, B0, D0)
303   return(res)
304 }
305
306
307 ##### intradaily Meteo to daily Meteo #####
308 MeteoI2MeteoD <- function(G0i)
309 {
310   lat <- G0i@latm
311   source <- G0i@source
312
313   dt0 <- getData(G0i)
314   dt <- dt0[, lapply(.SD, sum, na.rm = TRUE),
315             .SDcols = 'G0',
316             by = .(Dates = as.IDate(Dates))]
317   if('Ta' %in% names(dt0)){
318     Ta <- dt0[, .(Ta = mean(Ta),
319                   TempMin = min(Ta),
320                   TempMax = max(Ta)),
321               by = .(Dates = as.IDate(Dates))]
322     if(all(Ta$Ta == c(Ta$TempMin, Ta$TempMax))) Ta[, c('TempMin', 'TempMax')]
323     := NULL]
324     dt <- merge(dt, Ta)
325   }
326   if('G0' %in% names(dt)){
327     names(dt)[names(dt) == 'G0'] <- 'G0d'
328   }

```

```

328   if('D0' %in% names(dt)){
329       names(dt)[names(dt) == 'D0'] <- 'D0d'
330   }
331   if('B0' %in% names(dt)){
332       names(dt)[names(dt) == 'B0'] <- 'B0d'
333   }
334   G0d <- dt2Meteo(dt, lat, source, type = 'bd')
335   return(G0d)
336 }
337
338 ##### daily Meteo to monthly Meteo #####
339 Meteod2Meteom <- function(G0d)
340 {
341     lat <- G0d@latm
342     source <- G0d@source
343
344     dt <- getData(G0d)
345     nms <- names(dt)[-1]
346     dt <- dt[, lapply(.SD, mean),
347                     .SDcols = nms,
348                     by = .(month(Dates), year(Dates))]
349     dt[, Dates := fBTd()]
350     dt <- dt[, c('month', 'year') := NULL]
351
352     setcolorder(dt, 'Dates')
353
354     G0m <- dt2Meteo(dt, lat, source, type = 'prom')
355     return(G0m)
356 }
357
358 zoo2Meteo <- function(file, lat, source = '')
359 {
360     if(source == ''){
361         name <- deparse(substitute(file))
362         cl <- class(file)
363         source <- paste(cl, name, sep = '-')
364     }
365     bd <- data.table(file)
366     sample <- median(diff(index(file)))
367     IsDaily <- as.numeric(sample, units = 'days')>=1
368     type <- ifelse(IsDaily, 'bd', 'bdI')
369     result <- new(Class = 'Meteo',
370                  latm = lat,
371                  data = bd,
372                  type = type,
373                  source = source)
374 }
375
376 siarGET <- function(id, inicio, final, tipo = 'Mensuales', ambito = 'Estacion'){
377     if(!(tipo %in% c('Horarios', 'Diarios', 'Semanales', 'Mensuales'))){
378         stop('argument \'tipo\' must be: Horarios, Diarios, Semanales or
379 Mensuales')
380     }
381     if(!(ambito %in% c('CCAA', 'Provincia', 'Estacion'))){
382         stop('argument \'ambito\' must be: CCAA, Provincia or Estacion')
383     }
384
385     mainURL <- "https://servicio.mapama.gob.es"

```



```

385
386 path <- paste('/apisiar/API/v1/Datos', tipo, ambito, sep = '/')
387
388 ## prepare the APISiar
389 req <- request(mainURL) |>
390   req_url_path(path) |>
391   req_url_query(Id = id,
392                 FechaInicial = inicio,
393                 FechaFinal = final,
394                 ClaveAPI = '_Q8L_niYFBBmBs-
vB3UomUqdUYy98FTRX1aYbrZ8n2FXuHYGTV')
395 ## execute it
396 resp <- req_perform(req)
397
398 ##JSON to R
399 respJSON <- resp_body_json(resp, simplifyVector = TRUE)
400
401 if(!is.null(respJSON$MensajeRespuesta)){
402   stop(respJSON$MensajeRespuesta)
403 }
404
405 res0 <- data.table(respJSON$Datos)
406
407 res <- switch(tipo,
408              Horarios = {
409                res0[, HoraMin := as.ITime(sprintf('%04d', HoraMin),
410                                                  format = '%H%M')]
411                res0[, Fecha := as.IDate(Fecha, format = '%Y-%m-%d')]
412                res0[, Fecha := as.IDate(ifelse(HoraMin == as.ITime(0),
413                                                Fecha+1, Fecha))]
414                res0[, Dates := as.POSIXct(HoraMin, Fecha,
415                                             tz = 'Europe/Madrid')]
416                res0 <- res0[, .(Dates,
417                                G0 = Radiacion,
418                                Ta = TempMedia)]
419                return(res0)
420              },
421              Diarios = {
422                res0[, Dates := as.IDate(Fecha)]
423                res0 <- res0[, .(Dates,
424                                G0d = Radiacion * 277.78,
425                                Ta = TempMedia,
426                                TempMin,
427                                TempMax)]
428                return(res0)
429              },
430              Semanales = res0,
431              Mensuales = {
432                promDays<-c(17,14,15,15,15,10,18,18,18,19,18,13)
433                names(res0)[1] <- 'Year'
434                res0[, Dates := as.IDate(paste(Year, Mes,
435                                              promDays[Mes],
436                                              sep = '-'))]
437                res0 <- res0[, .(Dates,
438                                G0d = Radiacion * 277.78,
439                                Ta = TempMedia,
440                                TempMin,
441                                TempMax)]

```

```

442     })
443
444     return(res)
445 }
446
447 haversine <- function(lat1, lon1, lat2, lon2) {
448     R <- 6371 # Radius of the Earth in kilometers
449     dLat <- (lat2 - lat1) * pi / 180
450     dLon <- (lon2 - lon1) * pi / 180
451     a <- sin(dLat / 2) * sin(dLat / 2) + cos(lat1 * pi / 180) *
452         cos(lat2 * pi / 180) * sin(dLon / 2) * sin(dLon / 2)
453     c <- 2 * atan2(sqrt(a), sqrt(1 - a))
454     d <- R * c
455     return(d)
456 }
457
458 readSIAR <- function(Lon = 0, Lat = 0,
459                     inicio = paste(year(Sys.Date())-1, '01-01', sep = '-'),
460                     final = paste(year(Sys.Date())-1, '12-31', sep = '-'),
461                     tipo = 'Mensuales', n_est = 3){
462     inicio <- as.Date(inicio)
463     final <- as.Date(final)
464
465     n_reg <- switch(tipo,
466                     Horarios = {
467                         tt <- difftime(final, inicio, units = 'days')
468                         tt <- (as.numeric(tt)+1)*48
469                         tt <- tt*n_est
470                         tt
471                     },
472                     Diarios = {
473                         tt <- difftime(final, inicio, units = 'days')
474                         tt <- as.numeric(tt)+1
475                         tt <- tt*n_est
476                         tt
477                     },
478                     Semanales = {
479                         tt <- difftime(final, inicio, units = 'weeks')
480                         tt <- as.numeric(tt)
481                         tt <- tt*n_est
482                         tt
483                     },
484                     Mensuales = {
485                         tt <- difftime(final, inicio, units = 'weeks')
486                         tt <- as.numeric(tt)/4.34524
487                         tt <- ceiling(tt)
488                         tt <- tt*n_est
489                         tt
490                     })
491     if(n_reg > 100) stop(paste('Number of requested records (', n_reg,
492                               ') exceeds the maximum allowed (100)', sep = ''))
493
494     ## Obtain the nearest stations
495     siar <- est_SIAR[
496         Fecha_Instalacion <= final & (is.na(Fecha_Baja) | Fecha_Baja >= inicio)
497     ]
498
499     ## Weights for the interpolation

```

```

499   siar[, dist := haversine(Latitud, Longitud, Lat, Lon)]
500   siar <- siar[order(dist)][1:n_est]
501   siar[, peso := 1/dist]
502   siar[, peso := peso/sum(peso)]
503   ## Is the given location within the polygon formed by the stations?
504   siar <- siar[, .(Estacion,Codigo, dist, peso)]
505
506   ## List for the data.tables of siarGET
507   siar_list <- list()
508   for(codigo in siar$Codigo){
509     siar_list[[codigo]] <- siarGET(id = codigo,
510                                   inicio = as.character(inicio),
511                                   final = as.character(final),
512                                   tipo = tipo)
513     siar_list[[codigo]]$peso <- siar[Codigo == codigo, peso]
514   }
515
516   ## Bind the data.tables
517   s_comb <- rbindlist(siar_list, use.names = TRUE, fill = TRUE)
518
519   nms <- names(s_comb)
520   nms <- nms[-c(1, length(nms))]
521
522   ## Interpole
523   res <- s_comb[, lapply(.SD * peso, sum, na.rm = TRUE),
524                   .SDcols = nms,
525                   by = Dates]
526
527   ## Source
528   mainURL <- "https://servicio.mapama.gob.es"
529   Estaciones <- siar[, paste(Estacion, '(', Codigo, ')', sep = '')]
530   Estaciones <- paste(Estaciones, collapse = ', ')
531   source <- paste(mainURL, '\n -Estaciones:', Estaciones, sep = ' ')
532
533   res <- switch(tipo,
534                 Horarios = {dt2Meteo(res, lat = Lat, source = mainURL, type =
535 'bdI')},
536                 Diarios = {dt2Meteo(res, lat = Lat, source = mainURL, type = '
537 bd')},
538                 Semanales = {res},
539                 Mensuales = {dt2Meteo(res, lat = Lat, source = source, type =
540 'prom')})
541   return(res)
542 }

```

EXTRACTO DE CÓDIGO A.8: *meteoReaders*

## A.2. Clases

### A.2.1. 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',#intraday angles
7     sample='character',#sample of time

```

```

8         method='character'#method used for geometry calculations
9     ),
10    validity=function(object) {return(TRUE)}
11 )

```

EXTRACTO DE CÓDIGO A.9: *Clase Sol***A.2.2. Meteo**

```

1 setClass(
2   Class = 'Meteo', ##radiation and temperature data
3   slots = c(
4     latm='numeric',#latitud in degrees, >0 if North
5     data='data.table',#data, including G (Wh/m2) and Ta (°C)
6     type='character',#choose between 'prom', 'bd' and 'bdI'
7     source='character'#origin of the data
8   ),
9   validity=function(object) {return(TRUE)}
10 )

```

EXTRACTO DE CÓDIGO A.10: *Clase Meteo***A.2.3. G0**

```

1 setClass(
2   Class = 'G0',
3   slots = c(
4     GOD = 'data.table', #result of fCompD
5     GODm = 'data.table', #monthly means
6     GOy = 'data.table', #yearly values
7     GOI = 'data.table', #result of fCompI
8     Ta = 'data.table' #Ambient temperature
9   ),
10  contains = c('Sol', 'Meteo'),
11  validity = function(object) {return(TRUE)}
12 )
13

```

EXTRACTO DE CÓDIGO A.11: *Clase G0***A.2.4. Gef**

```

1 setClass(
2   Class='Gef',
3   slots = c(
4     GefD='data.table', #daily values
5     Gefdm='data.table', #monthly means
6     Gefy='data.table', #yearly values
7     GefI='data.table', #result of fInclin
8     Theta='data.table', #result of fTheta
9     iS='numeric', #dirt index
10    alb='numeric', #albedo
11    modeTrk='character', #tracking mode
12    modeShd='character', #shadow mode
13    angGen='list', #includes alpha, beta and betaLim
14    struct='list', #structure dimensions
15    distances='data.frame' #distances between structures
16  ),

```

```

17     contains='GO',
18     validity=function(object) {return(TRUE)}
19 )

```

EXTRACTO DE CÓDIGO A.12: Clase *Gef*

### A.2.5. ProdGCPV

```

1  setClass(
2      Class='ProdGCPV',
3      slots = c(
4          prodD='data.table', #daily values
5          prodDm='data.table', #monthly means
6          prody='data.table', #yearly values
7          prodI='data.table', #results of fProd
8          module='list',      #module characteristics
9          generator='list',    #generator characteristics
10         inverter='list',      #inverter characteristics
11         effSys='list'         #efficiency values of the system
12     ),
13     contains='Gef',
14     validity=function(object) {return(TRUE)}
15 )

```

EXTRACTO DE CÓDIGO A.13: Clase *ProdGCPV*

### A.2.6. ProdPVPS

```

1  setClass(
2      Class='ProdPVPS',
3      slots = c(
4          prodD='data.table', #daily values
5          prodDm='data.table', #monthly means
6          prody='data.table', #yearly values
7          prodI='data.table', #results of fPump
8          Pg='numeric',       #generator power
9          H='numeric',        #manometric head
10         pump='list',         #parameters of the pump
11         converter='list',    #inverter characteristics
12         effSys='list'        #efficiency values of the system
13     ),
14     contains='Gef',
15     validity=function(object) {return(TRUE)}
16 )

```

EXTRACTO DE CÓDIGO A.14: Clase *ProdPVPS*

### A.2.7. Shade

```

1  setClass(
2      Class='Shade',
3      slots = c(
4          FS='numeric', #shadows factor values
5          GRR='numeric', #Ground Requirement Ratio
6          Yf='numeric', #final productivity
7          FS.loess='loess', #local fitting of FS with loess
8          Yf.loess='loess', #local fitting of Yf with loess
9          modeShd='character', #mode of shadow

```

```

10     struct='list',          #dimensions of the structures
11     distances='data.frame', #distances between structures
12     res='numeric'          #difference between the different steps of
the calculations
13   ),
14   contains='ProdGCPV',##Resultado de prodGCPV sin sombras (Prod0)
15   validity=function(object) {return(TRUE)}
16 )

```

EXTRACTO DE CÓDIGO A.15: Clase **Shade**

## A.3. Funciones

### A.3.1. corrFdKt

```

1  utils::globalVariables(c('G0', 'lag1', 'lag2'))
2
3  ##### monthly Kt #####
4  Ktm <- function(sol, G0dm){
5    solf <- sol@sold[, .(Dates, Bo0d)]
6    solf[, c('month', 'year') := .(month(Dates), year(Dates))]
7    solf[, Bo0m := mean(Bo0d), by = .(month, year)]
8    G0df <- G0dm@data[, .(Dates, G0d)]
9    G0df[, c('month', 'year') := .(month(Dates), year(Dates))]
10   G0df[, G0d := mean(G0d), by = .(month, year)]
11   Ktm <- G0df$G0d/solf$Bo0m
12   return(Ktm)
13 }
14
15 ##### daily Kt #####
16 Ktd <- function(sol, G0d){
17   Bo0d <- sol@sold$Bo0d
18   G0d <- getG0(G0d)
19   Ktd <- G0d/Bo0d
20   return(Ktd)
21 }
22
23 ### intradaily
24 Kti <- function(sol, G0i){
25   Bo0 <- sol@solI$Bo0
26   G0i <- getG0(G0i)
27   Kti <- G0i/Bo0
28   return(Kti)
29 }
30
31
32 ##### monthly correlations #####
33
34 ### Page ###
35 FdKtPage <- function(sol, G0dm){
36   Kt <- Ktm(sol, G0dm)
37   Fd=1-1.13*Kt
38   return(data.table(Fd, Kt))
39 }
40
41 ### Liu and Jordan ###
42 FdKtLJ <- function(sol, G0dm){
43   Kt <- Ktm(sol, G0dm)

```

```

44     Fd=(Kt<0.3)*0.595774 +
45         (Kt>=0.3 & Kt<=0.7)*(1.39-4.027*Kt+5.531*Kt^2-3.108*Kt^3)+
46         (Kt>0.7)*0.215246
47     return(data.table(Fd, Kt))
48 }
49
50
51 ##### daily correlations #####
52
53 ### Collares-Pereira and Rabl
54 FdKtCPR <- function(sol, G0d){
55     Kt <- Ktd(sol, G0d)
56     Fd=(0.99*(Kt<=0.17))+(Kt>0.17 & Kt<0.8)*
57         (1.188-2.272*Kt+9.473*Kt^2-21.856*Kt^3+14.648*Kt^4)+
58         (Kt>=0.8)*0.2426688
59     return(data.table(Fd, Kt))
60 }
61
62 ### Erbs, Klein and Duffie ###
63 FdKtEKDd <- function(sol, G0d){
64     ws <- sol@sold$ws
65     Kt <- Ktd(sol, G0d)
66
67     WS1=(abs(ws)<1.4208)
68     Fd=WS1*((Kt<0.715)*(1-0.2727*Kt+2.4495*Kt^2-11.9514*Kt^3+9.3879*Kt^4)+
69         (Kt>=0.715)*(0.143))+
70         !WS1*((Kt<0.722)*(1+0.2832*Kt-2.5557*Kt^2+0.8448*Kt^3)+
71         (Kt>=0.722)*(0.175))
72     return(data.table(Fd, Kt))
73 }
74
75 ### CLIMED1 ###
76 FdKtCLIMEDd <- function(sol, G0d){
77     Kt <- Ktd(sol, G0d)
78     Fd=(Kt<=0.13)*(0.952)+
79         (Kt>0.13 & Kt<=0.8)*(0.868+1.335*Kt-5.782*Kt^2+3.721*Kt^3)+
80         (Kt>0.8)*0.141
81     return(data.table(Fd, Kt))
82 }
83
84 ##### intradaily correlations #####
85
86 ### intradaily EKD ###
87 FdKtEKDh <- function(sol, G0i){
88     Kt <- Kti(sol, G0i)
89     Fd=(Kt<=0.22)*(1-0.09*Kt)+
90         (Kt>0.22 & Kt<=0.8)*(0.9511-0.1604*Kt+4.388*Kt^2-16.638*Kt^3+12.336*Kt^4)+
91         (Kt>0.8)*0.165
92     return(data.table(Fd, Kt))
93 }
94
95 ### intradaily CLIMED
96 FdKtCLIMEDh <- function(sol, G0i){
97     Kt <- Kti(sol, G0i)
98     Fd=(Kt<=0.21)*(0.995-0.081*Kt)+
99         (Kt>0.21 & Kt<=0.76)*(0.724+2.738*Kt-8.32*Kt^2+4.967*Kt^3)+
100         (Kt>0.76)*0.180
101     return(data.table(Fd, Kt))

```

```

102 }
103
104 ### intradaily Boland, Ridley and Lauret ###
105 FdKtBRL <- function(sol, G0i){
106   Kt <- Kti(sol, G0i)
107   sample <- sol@sample
108   ind <- indexI(sol)
109
110   soli <- as.data.tableI(sol, complete = TRUE)
111   w <- soli$w
112   night <- soli$night
113   Als <- soli$Als
114   Bo0 <- soli$Bo0
115
116   G0d <- data.table(ind,
117                     GO = getG0(G0i),
118                     Bo0 = Bo0)
119   G0d[, G0d := P2E(G0, sample), by = truncDay(ind)]
120   G0d[, Bo0d := P2E(Bo0, sample), by = truncDay(ind)]
121   ktd <- G0d[, ifelse(night, 0, G0d/Bo0d)]
122
123   ##persistence
124   pers <- persistence(sol, Kt)
125
126
127   ##fd calculation
128   Fd=(1+exp(-5.38+6.63*Kt+0.006*r2h(w)-0.007*r2d(Als)+1.75*ktd+1.31*pers))
129   ^(-1)
130   return(data.table(Fd, Kt))
131 }
132
133 persistence <- function(sol, kt){
134   kt <- data.table(ind = indexI(sol), kt)
135   ktNA <- na.omit(copy(kt))
136   iDay <- truncDay(ktNA[[1]])
137
138   x <- rle(as.numeric(iDay))$lengths
139   xLast <- cumsum(x)
140   xFirst <- xLast-x+1
141
142   ktNA[, lag1 := shift(kt, n = -1, type = 'lag', fill = NA)]
143   ktNA[xLast, lag1 := ktNA[xLast-1, kt]]
144
145   ktNA[, lag2 := shift(kt, n = 1, type = 'lag', fill = NA)]
146   ktNA[xFirst, lag2 := ktNA[xFirst + 1, kt]]
147
148   pers <- merge(kt, ktNA, by = 'ind', all = TRUE)
149   return(pers[, 1/2*(lag1+lag2)])
150 }

```

EXTRACTO DE CÓDIGO A.16: *corrFdKt*

## A.3.2. 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='%d-%m-%Y'){
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='UTC')
14    )
15    BTd
16 }

```

EXTRACTO DE CÓDIGO A.17: *fBTd*

### A.3.3. fBTi

```

1 intervalo <- function(day, sample){
2     intervalo <- seq.POSIXt(from = as.POSIXct(paste(day, '00:00:00'), tz = 'UTC'
3     ),
4     to = as.POSIXct(paste(day, '23:59:59'), tz = 'UTC'),
5     by = sample)
6     return(intervalo)
7 }
8 fBTi <- function(BTd, sample = 'hour'){
9     BTi <- lapply(BTd, intervalo, sample)
10    BTi <- do.call(c, BTi)
11    return(BTi)
12 }

```

EXTRACTO DE CÓDIGO A.18: *fBTi*

### A.3.4. fCompD

```

1 utils::globalVariables('lat')
2
3 fCompD <- function(sol, G0d, corr = 'CPR', f)
4 {
5     if(!(corr %in% c('CPR', 'Page', 'LJ', 'EKDd', 'CLIMEDd', 'user', 'none'))){
6         warning('Wrong descriptor of correlation Fd-Ktd. Set CPR.')
7         corr <- 'CPR'
8     }
9     if(class(sol)[1] != 'Sol'){
10        sol <- sol[, calcSol(lat = unique(lat), BTi = Dates)]
11    }
12    if(class(G0d)[1] != 'Meteo'){
13        dt <- copy(data.table(G0d))
14        if(!('Dates' %in% names(dt))){
15            dt[, Dates := indexD(sol)]
16            setcolorder(dt, 'Dates')
17            setkey(dt, 'Dates')
18        }
19        if('lat' %in% names(dt)){
20            latg <- unique(dt$lat)
21            dt[, lat := NULL]
22        }else{latg <- getLat(sol)}

```

```

23   G0d <- dt2Meteo(dt, latg)
24   }
25
26   stopifnot(indexD(sol) == indexD(G0d))
27   Bo0d <- sol@solD$Bo0d
28   G0 <- getData(G0d)$G0
29
30   is.na(G0) <- (G0>Bo0d)
31
32   ### the Direct and Difuse data is not given
33   if(corr != 'none'){
34     Fd <- switch(corr,
35                 CPR = FdKtCPR(sol, G0d),
36                 Page = FdKtPage(sol, G0d),
37                 LJ = FdKtLJ(sol, G0d),
38                 EKDd = FdKtEKDd(sol, G0d),
39                 CLIMEDd = FdKtCLIMEDd(sol, G0d),
40                 user = f(sol, G0d))
41     Kt <- Fd$Kt
42     Fd <- Fd$Fd
43     D0d <- Fd * G0
44     B0d <- G0 - D0d
45   }
46   ### the Direct and Difuse data is given
47   else {
48     G0 <- getData(G0d)$G0d
49     D0d <- getData(G0d)[['D0d']]
50     B0d <- getData(G0d)[['B0d']]
51     Fd <- D0d/G0
52     Kt <- G0/Bo0d
53   }
54
55   result <- data.table(Dates = indexD(sol), Fd, Kt, G0d = G0, D0d, B0d)
56   setkey(result, 'Dates')
57   result
58 }

```

EXTRACTO DE CÓDIGO A.19: *fCompD*

### A.3.5. fCompI

```

1 fCompI <- 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 instantaneous values are not provided, compD is used instead.
20 if (missing(GOI)) {
21
22     GOI <- collper(sol, compD)
23     GO <- GOI$G0
24     B0 <- GOI$B0
25     D0 <- GOI$D0
26
27     Fd <- D0/GO
28     Kt <- GO/Bo0
29
30 } else { ## Use instantaneous values if provided through GOI
31
32     if(class(GOI)[1] != 'Meteo'){
33         dt <- copy(data.table(GOI))
34         if(!('Dates' %in% names(dt))){
35             if(length(dt) == 1) names(dt) <- 'G0'
36             dt[, Dates := indexI(sol)]
37             setcolororder(dt, 'Dates')
38             setkey(dt, 'Dates')
39         }
40         if('lat' %in% names(GOI)){latg <- unique(GOI$lat)}
41         else{latg <- lat}
42         GOI <- dt2Meteo(dt, latg)
43     }
44
45     if (corr!='none'){
46         ## Filter values: surface irradiation must be lower than
47         ## extraterrestrial;
48         if (filterG0) {
49             GO <- getG0(GOI)
50             is.na(GO) <- (GO > Bo0)
51             GOI <- dt2Meteo(data.table(Dates = indexD(GOI),
52                                     GO = GO),
53                           lat = GOI@latm,
54                           source = GOI@source,
55                           type = GOI@type)
56         }
57
58         ## Fd-Kt correlation
59         Fd <- switch(corr,
60                     EKDh = FdKtEKDh(sol, GOI),
61                     CLIMEDh = FdKtCLIMEDh(sol, GOI),
62                     BRL = FdKtBRL(sol, GOI),
63                     user = f(sol, GOI))
64
65         Kt <- Fd$Kt
66         Fd <- Fd$Fd
67         D0 <- Fd * GO
68         B0 <- GO - D0
69
70     } else {
71         GO <- getG0(GOI)
72         D0 <- getData(GOI)[['D0']]
73         B0 <- getData(GOI)[['B0']]
74         ## Filter values: surface irradiation must be lower than
75         ## extraterrestrial;

```

```

76         if (isTRUE(filterG0)) is.na(G0) <- is.na(D0) <- is.na(B0) <- (G0 >
Bo0)
77
78         Fd <- D0/G0
79         Kt <- G0/Bo0
80     }
81 }
82 ## Values outside sunrise-sunset are set to zero
83 G0[night] <- D0[night] <- B0[night] <- Kt[night] <- Fd[night] <- 0
84
85 result <- data.table(Dates, Fd, Kt, G0, D0, B0)
86 setkey(result, 'Dates')
87 result
88 }

```

EXTRACTO DE CÓDIGO A.20: *fCompI*A.3.6. *fInclin*

```

1 fInclin <- function(compI, angGen, iS = 2, alb = 0.2, horizBright = TRUE, HCPV =
FALSE){
2     ##compI es class='G0'
3
4     ##Arguments
5     stopifnot(iS %in% 1:4)
6     Beta <- angGen$Beta
7     Alpha <- angGen$Alpha
8     cosTheta <- angGen$cosTheta
9
10    comp <- as.data.tableI(compI, complete=TRUE)
11    night <- comp$night
12    B0 <- comp$B0
13    Bo0 <- comp$Bo0
14    D0 <- comp$D0
15    G0 <- comp$G0
16    cosThzS <- comp$cosThzS
17    is.na(cosThzS) <- night
18
19    ##N.Martin method for dirt and non-perpendicular incidence
20    Suc <- rbind(c(1, 0.17, -0.069),
21                c(0.98,.2,-0.054),
22                c(0.97,0.21,-0.049),
23                c(0.92,0.27,-0.023))
24    FTb <- (exp(-cosTheta/Suc[iS,2]) - exp(-1/Suc[iS,2]))/(1 - exp(-1/Suc[iS,2])
)
25    FTd <- exp(-1/Suc[iS,2] * (4/(3*pi) * (sin(Beta) + (pi - Beta - sin(Beta))/
(1 + cos(Beta)))) +
26                Suc[iS,3] * (sin(Beta) + (pi - Beta - sin(Beta))/
(1 + cos(Beta)))^2))
27    FTr <- exp(-1/Suc[iS,2] * (4/(3*pi) * (sin(Beta) + (Beta - sin(Beta))/(1 -
cos(Beta)))) +
28                Suc[iS,3] * (sin(Beta) + (Beta - sin(Beta))/(1 -
cos(Beta)))^2))
29
30    ##Hay and Davies method for diffuse treatment
31    B <- B0 * cosTheta/cosThzS * (cosThzS>0.007) #The factor cosThzS>0.007 is
needed to eliminate erroneous results near dawn
32    k1 <- B0/(Bo0)

```

```

33 Di <- D0 * (1-k1) * (1+cos(Beta))/2
34 if (horizBright) Di <- Di * (1+sqrt(B0/G0) * sin(Beta/2)^3)
35 Dc <- D0 * k1 * cosTheta/cosThzS * (cosThzS>0.007)
36 R <- alb * G0 * (1-cos(Beta))/2
37 D <- (Di + Dc)
38 ##Extraterrestrial irradiance on the inclined plane
39 Bo <- Bo0 * cosTheta/cosThzS * (cosThzS>0.007)
40 ##Normal direct irradiance (DNI)
41 Bn <- B0/cosThzS
42 ##Sum of components
43 G <- B + D + R
44 Ref <- R * Suc[iS,1] * (1-FTr) * (!HCPV)
45 Ref[is.nan(FTr)] <- 0 #When cos(Beta)=1, FTr=NaN. Cancel Ref.
46 Dief <- Di * Suc[iS,1] * (1 - FTd) * (!HCPV)
47 Dcef <- Dc * Suc[iS,1] * (1 - FTb) * (!HCPV)
48 Def <- Dief + Dcef
49 Bef <- B * Suc[iS,1] * (1 - FTb)
50 Gef <- Bef + Def + Ref
51
52 result <- data.table(Bo, Bn,
53                      G, D, Di, Dc, B, R,
54                      FTb, FTd, FTr,
55                      Dief, Dcef, Gef, Def, Bef, Ref)
56
57 ## Use 0 instead of NA for irradiance values
58 result[night] <- 0
59 result[, Dates := indexI(compI)]
60 result[, .SD, by = Dates]
61 setcolorder(result, c('Dates', names(result)[-length(result)]))
62 result
63 }

```

EXTRACTO DE CÓDIGO A.21: *fInclin*

### A.3.7. fProd

```

1 ## voc, isc, vmpp, impp : *cell* values
2 ## Voc, Isc, Vmpp, Impp: *module/generator* values
3
4 ## Compute Current - Voltage characteristic of a solar *cell* with Gef
5 ## and Ta
6 iv <- function(vocn, iscn, vmn, imn,
7               TONC, CoefVT = 2.3e-3,
8               Ta, Gef,
9               vmin = NULL, vmax = NULL)
10 {
11   ##Cell Constants
12   Gstc <- 1000
13   Ct <- (TONC - 20) / 800
14   Vtn <- 0.025 * (273 + 25) / 300
15   m <- 1.3
16
17   ##Cell temperature
18   Tc <- Ta + Ct * Gef
19   Vt <- 0.025 * (Tc + 273)/300
20
21   ## Series resistance
22   Rs <- (vocn - vmn + m * Vtn * log(1 - imn/iscn)) / imn

```

```

23
24   ## Voc and Isc at ambient conditions
25   voc <- vocn - CoefVT * (Tc - 25)
26   isc <- iscn * Gef/Gstc
27
28   ## Ruiz method for computing voltage and current characteristic of a *cell*
29   rs <- Rs * isc/voc
30   koc <- voc/(m * Vt)
31
32   ## Maximum Power Point
33   Dm0 <- (koc - 1)/(koc - log(koc))
34   Dm <- Dm0 + 2 * rs * Dm0^2
35
36   impp <- isc * (1 - Dm/koc)
37   vmpp <- voc * (1 - log(koc/Dm)/koc - rs * (1 - Dm/koc))
38
39   vdc <- vmpp
40   idc <- impp
41
42   ## When the MPP is below/above the inverter voltage limits, it
43   ## sets the voltage point at the corresponding limit.
44
45
46   ## Auxiliary functions for computing the current at a defined
47   ## voltage.
48   ilimit <- function(v, koc, rs)
49   {
50     if (is.na(koc))
51       result <- NA
52     else
53     {
54       ## The IV characteristic is an implicit equation. The starting
55       ## point is the voltage of the cell (imposed by the inverter
56       ## limit).
57
58       izero <- function(i, v, koc, rs)
59       {
60         vp <- v + i * rs
61         Is <- 1/(1 - exp(-koc * (1 - rs)))
62         result <- i - (1 - Is * (exp(-koc * (1 - vp)) - exp(-koc * (1 -
rs))))))
63       }
64
65       result <- uniroot(f = izero,
66                        interval = c(0,1),
67                        v = v,
68                        koc = koc,
69                        rs = rs)$root
70     }
71     result
72   }
73   ## Inverter minimum voltage
74   if (!is.null(vmin))
75   {
76     if (any(vmpp < vmin, na.rm = TRUE))
77     {
78       indMIN <- which(vmpp < vmin)
79       imin <- sapply(indMIN, function(i)

```

```

80     {
81         vocMIN <- voc[i]
82         kocMIN <- koc[i]
83         rsMIN <- rs[i]
84         vmin <- vmin/vocMIN
85         ##v debe estar entre 0 y 1
86         vmin[vmin < 0] <- 0
87         vmin[vmin > 1] <- 1
88         ilimit(vmin, kocMIN, rsMIN)
89     })
90     iscMIN <- isc[indMIN]
91     idc[indMIN] <- imin * iscMIN
92     vdc[indMIN] <- vmin
93     warning('Minimum MPP voltage of the inverter has been reached')
94 }
95
96 if (!is.null(vmax))
97 {
98     if (any(vmpp > vmax, na.rm = TRUE))
99     {
100         indMAX <- which(vmpp > vmax)
101         imax <- sapply(indMAX, function(i)
102         {
103             vocMAX <- voc[i]
104             kocMAX <- koc[i]
105             rsMAX <- rs[i]
106             vmax <- vmax / vocMAX
107             ##v debe estar entre 0 y 1
108             vmax[vmax < 0] <- 0
109             vmax[vmax > 1] <- 1
110             ilimit(vmax, kocMAX, rsMAX)
111         })
112         iscMAX <- isc[indMAX]
113         idc[indMAX] <- imax * iscMAX
114         vdc[indMAX] <- vmax
115         warning('Maximum MPP voltage of the inverter has been reached')
116     }
117 }
118 data.table(Ta, Tc, Gef, voc, isc, vmpp, impp, vdc, idc)
119 }
120
121 fProd <- function(inclin,
122                   module=list(),
123                   generator=list(),
124                   inverter=list(),
125                   effSys=list()
126                   )
127 {
128
129     stopifnot(is.list(module),
130               is.list(generator),
131               is.list(inverter),
132               is.list(effSys)
133               )
134     ## Extract data from objects
135     if (class(inclin)[1]=='Gef') {
136         indInclin <- indexI(inclin)
137         gefI <- as.data.tableI(inclin, complete = TRUE)

```

```

138     Gef <- gefI$Gef
139     Ta <- gefI$Ta
140   } else {
141     Gef <- inclin$Gef
142     Ta <- inclin$Ta
143   }
144
145   ## Module, generator, and inverter parameters
146   module.default <- list(Vocn = 57.6,
147                         Iscn = 4.7,
148                         Vmn = 46.08,
149                         Imn = 4.35,
150                         Ncs = 96,
151                         Ncp = 1,
152                         CoefVT = 0.0023,
153                         TONC = 47)
154   module <- modifyList(module.default, module)
155   ## Make these parameters visible because they will be used often.
156   Ncs <- module$Ncs
157   Ncp <- module$Ncp
158
159   generator.default <- list(Nms = 12,
160                             Nmp = 11)
161   generator <- modifyList(generator.default, generator)
162   generator$Pg <- (module$Vmn * generator$Nms) *
163     (module$Imn * generator$Nmp)
164   Nms <- generator$Nms
165   Nmp <- generator$Nmp
166
167   inverter.default <- list(Ki = c(0.01,0.025,0.05),
168                             Pinv = 25000,
169                             Vmin = 420,
170                             Vmax = 750,
171                             Gumb = 20)
172   inverter <- modifyList(inverter.default, inverter)
173   Pinv <- inverter$Pinv
174
175   effSys.default <- list(ModQual = 3,
176                           ModDisp = 2,
177                           OhmDC = 1.5,
178                           OhmAC = 1.5,
179                           MPP = 1,
180                           TrafoMT = 1,
181                           Disp = 0.5)
182   effSys <- modifyList(effSys.default, effSys)
183
184   ## Solar Cell i-v
185   vocn <- with(module, Vocn / Ncs)
186   iscn <- with(module, Iscn / Ncp)
187   vmn <- with(module, Vmn / Ncs)
188   imn <- with(module, Imn / Ncp)
189   vmin <- with(inverter, Vmin / (Ncs * Nms))
190   vmax <- with(inverter, Vmax / (Ncs * Nms))
191
192   cell <- iv(vocn, iscn,
193             vmn, imn,
194             module$TONC, module$CoefVT,
195             Ta, Gef,

```



```

196         vmin, vmax)
197
198     ## Generator voltage and current
199     Idc <- Nmp * Ncp * cell$Idc
200     Isc <- Nmp * Ncp * cell$isc
201     Impp <- Nmp * Ncp * cell$impp
202     Vdc <- Nms * Ncs * cell$vdc
203     Voc <- Nms * Ncs * cell$voc
204     Vmpp <- Nms * Ncs * cell$vmpp
205
206     ##DC power (normalization with nominal power of inverter)
207     ##including losses
208     PdcN <- with(effSys, (Idc * Vdc) / Pinv *
209                       (1 - ModQual / 100) *
210                       (1 - ModDisp / 100) *
211                       (1 - MPP / 100) *
212                       (1 - OhmDC / 100)
213           )
214
215     ##Normalized AC power to the inverter
216     Ki <- inverter$Ki
217     if (is.matrix(Ki)) { #Ki is a matrix of nine coefficients-->dependence with
tension
218         VP <- cbind(Vdc, PdcN)
219         PacN <- apply(VP, 1, solvePac, Ki)
220     } else { #Ki is a vector of three coefficients-->without dependence on
voltage
221         A <- Ki[3]
222         B <- Ki[2] + 1
223         C <- Ki[1] - (PdcN)
224         PacN <- (-B + sqrt(B^2 - 4 * A * C))/(2 * A)
225     }
226     EffI <- PacN / PdcN
227     pacNeg <- PacN <= 0
228     PacN[pacNeg] <- PdcN[pacNeg] <- EffI[pacNeg] <- 0
229
230
231     ##AC and DC power without normalization
232     Pac <- with(effSys, PacN * Pinv *
233               (Gef > inverter$Gumb) *
234               (1 - OhmAC / 100) *
235               (1 - TrafoMT / 100) *
236               (1 - Disp / 100))
237     Pdc <- PdcN * Pinv * (Pac > 0)
238
239
240     ## Result
241     resProd <- data.table(Tc = cell$Tc,
242                          Voc, Isc,
243                          Vmpp, Impp,
244                          Vdc, Idc,
245                          Pac, Pdc,
246                          EffI)
247     if (class(inclin)[1] %in% 'Gef'){
248         result <- resProd[, .SD,
249                           by=.(Dates = indInclin)]
250         attr(result, 'generator') <- generator
251         attr(result, 'module') <- module

```

```

252     attr(result, 'inverter') <- inverter
253     attr(result, 'effSys') <- effSys
254     return(result)
255   } else {
256     result <- cbind(inclin, resProd)
257     return(result)
258   }
259 }

```

EXTRACTO DE CÓDIGO A.22: *fProd***A.3.8. fPump**

```

1 fPump <- function(pump, H){
2
3   w1=3000 ##synchronous rpm frequency
4   wm=2870 ##rpm frequency with slip when applying voltage at 50 Hz
5   s=(w1-wm)/w1
6   fen=50 ##Nominal electrical frequency
7   fmin=sqrt(H/pump$a)
8   fmax=with(pump, (-b*Qmax+sqrt(b^2*Qmax^2-4*a*(c*Qmax^2-H)))/(2*a))
9   ##fb is rotation frequency (Hz) of the pump,
10  ##fe is the electrical frequency applied to the motor
11  ##which makes it rotate at a frequency fb (and therefore also the pump).
12  fb=seq(fmin,min(60,fmax),length=1000) #The maximum frequency is 60
13  fe=fb/(1-s)
14
15  ###Flow
16  Q=with(pump, (-b*fb-sqrt(b^2*fb^2-4*c*(a*fb^2-H)))/(2*c))
17  Qmin=0.1*pump$Qn*fb/50
18  Q=Q+(Qmin-Q)*(Q<Qmin)
19
20  ###Hydraulic power
21  Ph=2.725*Q*H
22
23  ###Mechanical power
24  Q50=50*Q/fb
25  H50=H*(50/fb)^2
26  etab=with(pump, j*Q50^2+k*Q50+1)
27  Pb50=2.725*H50*Q50/etab
28  Pb=Pb50*(fb/50)^3
29
30  ###Electrical power
31  Pbc=Pb*50/fe
32  etam=with(pump, g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
33  Pmc=Pbc/etam
34  Pm=Pmc*fe/50
35  Pac=Pm
36  ##Pdc=Pm/(etac*(1-cab))
37
38  ###I build functions for flow, frequency and powers
39  ###to adjust the AC power.
40  fQ<-splinefun(Pac,Q)
41  fFreq<-splinefun(Pac,fe)
42  fPb<-splinefun(Pac,Pb)
43  fPh<-splinefun(Pac,Ph)
44  lim=c(min(Pac),max(Pac))
45  ##lim marks the operating range of the pump

```

```

46   result<-list(lim = lim,
47               fQ = fQ,
48               fPb = fPb,
49               fPh = fPh,
50               fFreq = fFreq)
51 }

```

EXTRACTO DE CÓDIGO A.23: *fPump*A.3.9. *fSolD*

```

1  utils::globalVariables(c('decl', 'eo', 'EoT', 'ws'))
2
3  fSolD <- function(lat, BTd, method = 'michalsky'){
4    if (abs(lat) > 90){
5      lat <- sign(lat) * 90
6      warning(paste('Latitude outside acceptable values. Set to', lat))
7    }
8    sun <- data.table(Dates = unique(as.IDate(BTd)),
9                     lat = lat)
10
11    #### solarAngles ####
12
13    ##Declination
14    sun[, decl := declination(Dates, method = method)]
15    ##Eccentricity
16    sun[, eo := eccentricity(Dates, method = method)]
17    ##Equation of time
18    sun[, EoT := eot(Dates)]
19    ##Solar time
20    sun[, ws := sunrise(Dates, lat, method = method,
21                       decl = decl)]
22    ##Extraterrestrial irradiance
23    sun[, Bo0d := bo0d(Dates, lat, method = method,
24                      decl = decl,
25                      eo = eo,
26                      ws = ws
27                      )]
28    setkey(sun, Dates)
29    return(sun)
30 }

```

EXTRACTO DE CÓDIGO A.24: *fSolD*A.3.10. *fSolI*

```

1  utils::globalVariables(c('eqtime', 'w', 'night', 'cosThzS',
2                          'AlS', 'AzS', 'Times'))
3
4  fSolI <- function(sold, sample = 'hour', BTi,
5                   EoT = TRUE, keep.night = TRUE, method = 'michalsky')
6  {
7    #Solar constant
8    Bo <- 1367
9
10   if(missing(BTi)){
11     BTd <- sold$Dates
12     BTi <- fBTi(BTd, sample)

```

```

13 }
14 sun <- data.table(Dates = as.IDate(BTi),
15                  Times = as.ITime(BTi))
16 sun <- merge(sold, sun, by = 'Dates')
17 sun[, eqtime := EoT]
18 sun[, EoT := NULL]
19
20 #sun hour angle
21 sun[, w := sunHour(Dates, BTi, EoT = EoT, method = method, eqtime = eqtime)]
22
23 #classify night elements
24 sun[, night := abs(w) >= abs(ws)]
25
26 #zenith angle
27 sun[, cosThzS := zenith(Dates, lat, BTi,
28                        method = method,
29                        decl = decl,
30                        w = w
31                        )]
32
33 #solar altitude angle
34 sun[, AlS := asin(cosThzS)]
35
36 #azimuth
37 sun[, AzS := azimuth(Dates, lat, BTi, sample,
38                    method = method,
39                    decl = decl,
40                    w = w,
41                    cosThzS = cosThzS)]
42
43 #Extraterrestrial irradiance
44 sun[, Bo0 := Bo * eo * cosThzS]
45
46 #When it is night there is no irradiance
47 sun[night == TRUE, Bo0 := 0]
48
49 #Erase columns that are in sold
50 sun[, decl := NULL]
51 sun[, eo := NULL]
52 sun[, eqtime := NULL]
53 sun[, ws := NULL]
54 sun[, Bo0d := NULL]
55
56 #Column Dates with Times
57 sun[, Dates := as.POSIXct(Dates, Times, tz = 'UTC')]
58 sun[, Times := NULL]
59
60 #keep night
61 if(!keep.night){
62   sun <- sun[night == FALSE]
63 }
64
65 return(sun)
66 }

```

EXTRACTO DE CÓDIGO A.25: *fSolI*A.3.11. *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 }

```

EXTRACTO DE CÓDIGO A.26: *fSombra*

```

1 fSombra2X<-function(angGen,distances,struct)
2 {
3   stopifnot(is.list(struct),is.data.frame(distances))
4   ##I prepare starting data
5   P=with(struct,distances/W)
6   b=with(struct,L/W)
7   AzS=angGen$AzS
8   Beta=angGen$Beta
9   AlS=angGen$AlS
10
11   d1=abs(P$Lew*cos(AzS)-P$Lns*sin(AzS))
12   d2=abs(P$Lew*sin(AzS)+P$Lns*cos(AzS))
13   FC=sin(AlS)/sin(Beta+AlS)
14   s=b*cos(Beta)+(b*sin(Beta)+P$H)/tan(AlS)
15   FS1=1-d1
16   FS2=s-d2
17   SombraCond=(FS1>0)*(FS2>0)*(P$Lew*Azs>=0)
18   SombraCond[is.na(SombraCond)]<-FALSE #NAs are of no use to me in a logical
vector. I replace them with FALSE
19   ## Result
20   FS=SombraCond*(FS1*FS2*FC)/b
21   FS[FS>1]<-1
22   return(FS)
23 }

```

EXTRACTO DE CÓDIGO A.27: *fSombra2X*

```

1 utils::globalVariables(c('Lew', 'Lns', 'H'))
2
3 fSombra6<-function(angGen, distances, struct, prom=TRUE)
4 {
5   stopifnot(is.list(struct),
6             is.data.frame(distances))
7   ##distances only has three distances, so I generate a grid
8   if (dim(distances)[1]==1){
9     Red <- distances[, .(Lew = c(-Lew, 0, Lew, -Lew, Lew),
10                             Lns = c(Lns, Lns, Lns, 0, 0),
11                             H=H)]
12   } else { #distances is an array, so there is no need to generate the grid
13     Red<-distances[1:5,]} #I only need the first 5 rows...necessary in case
a wrong data.frame is delivered
14
15   ## I calculate the shadow due to each of the 5 followers
16   SombraGrupo<-matrix(ncol=5,nrow=dim(angGen)[1]) ###VECTORIZE
17   for (i in 1:5) {SombraGrupo[,i]<-fSombra2X(angGen,Red[i,],struct)}

```

```

18  ##To calculate the Average Shadow, I need the number of followers in each
    position (distrib)
19  distrib=with(struct,c(1,Ncol-2,1,Nrow-1,(Ncol-2)*(Nrow-1),Nrow-1))
20  vProm=c(sum(distrib[c(5,6)]),
21          sum(distrib[c(4,5,6)]),
22          sum(distrib[c(4,5)]),
23          sum(distrib[c(2,3,5,6)]),
24          sum(distrib[c(1,2,4,5)]))
25  Nseg=sum(distrib) ##Total number of followers
26  ##With the SWEEP function I multiply the Shadow Factor of each type (
    ShadowGroup columns) by the vProm result
27
28  if (prom==TRUE){
29      ## Average Shadow Factor in the group of SIX followers taking into
    account distribution
30      FS=rowSums(sweep(SombraGrupo,2,vProm,'*'))/Nseg
31      FS[FS>1]<-1
32  } else {
33      ## Shadow factor on follower #5 due to the other 5 followers
34      FS=rowSums(SombraGrupo)
35      FS[FS>1]<-1}
36  return(FS)
37 }

```

EXTRACTO DE CÓDIGO A.28: *fSombra6*

```

1  fSombraEst<-function(angGen, distances, struct)
2  {
3      stopifnot(is.list(struct),is.data.frame(distances))
4      ## I prepare starting data
5      dist <- with(struct, distances/L)
6      Alpha <- angGen$Alpha
7      Beta <- angGen$Beta
8      AlS <- angGen$AlS
9      AzS <- angGen$AzS
10     cosTheta <- angGen$cosTheta
11     h <- dist$H #It must be previously normalized
12     if(is.null(h)) h <- 0
13     d <- dist$D
14     ## Calculations
15     s=cos(Beta)+cos(Alpha-AzS)*(sin(Beta)+h)/tan(AlS)
16     FC=sin(AlS)/sin(Beta+AlS)
17     SombraCond=(s-d>0)
18     FS=(s-d)*SombraCond*FC*(cosTheta>0)
19     ## Result
20     FS=FS*(FS>0)
21     FS[FS>1]<-1
22     return(FS)
23 }

```

EXTRACTO DE CÓDIGO A.29: *fSombraEst*

```

1  fSombraHoriz<-function(angGen, distances, struct)
2  {
3      stopifnot(is.list(struct),is.data.frame(distances))
4      ## I prepare starting data
5      d <- with(struct, distances/L)
6      AzS <- angGen$AzS
7      AlS <- angGen$AlS

```

```

8   Beta <- angGen$Beta
9   lew <- d$Lew #It must be previously normalized
10  ## Calculations
11  Beta0=atan(abs(sin(AzS)/tan(A1S)))
12  FS=1-lew*cos(Beta0)/cos(Beta-Beta0)
13  SombraCond=(FS>0)
14  ## Result
15  FS=FS*SombraCond
16  FS[FS>1]<-1
17  return(FS)
18 }

```

EXTRACTO DE CÓDIGO A.30: *fSombraHoriz*

### A.3.12. fTemp

```

1  fTemp<-function(sol, BD)
2  {
3    ##sol is an object with class='Sol'
4    ##BD is an object with class='Meteo', whose 'data' slot contains two columns
   called "TempMax" and "TempMin"
5
6    stopifnot(class(sol)=='Sol')
7    stopifnot(class(BD)=='Meteo')
8
9    checkIndexD(indexD(sol), indexD(BD))
10
11   Dates<-indexI(sol)
12   x <- as.Date(Dates)
13   ind.rep <- cumsum(c(1, diff(x) != 0))
14
15   TempMax <- BD@data$TempMax[ind.rep]
16   TempMin <- BD@data$TempMin[ind.rep]
17   ws <- sol@solD$ws[ind.rep]
18   w <- sol@solI$w
19
20   ##Generate temperature sequence from database Maxima and Minima
21
22   Tm=(TempMin+TempMax)/2
23   Tr=(TempMax-TempMin)/2
24
25   wp=pi/4
26
27   a1=pi*12*(ws-w)/(21*pi+12*ws)
28   a2=pi*(3*pi-12*w)/(3*pi-12*ws)
29   a3=pi*(24*pi+12*(ws-w))/(21*pi+12*ws)
30
31   T1=Tm-Tr*cos(a1)
32   T2=Tm+Tr*cos(a2)
33   T3=Tm-Tr*cos(a3)
34
35   Ta=T1*(w<=ws)+T2*(w>ws&w<=wp)+T3*(w>wp)
36
37   ##Result
38   result<-data.table(Dates, Ta)
39 }

```

EXTRACTO DE CÓDIGO A.31: *fTemp*

## A.3.13. 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)) ##When lat=0, sign(lat)=0. I
    change it to sign(lat)=1
11
12  solI<-as.data.tableI(sol, complete=TRUE, day = TRUE)
13  AlS=solI$AlS
14  AzS=solI$AzS
15  decl=solI$decl
16  w<-solI$w
17
18  night<-solI$night
19
20  Beta<-switch(modeTrk,
21    two = {Beta2x=pi/2-AlS
22      Beta=Beta2x+(betaLim-Beta2x)*(Beta2x>betaLim)},
23    fixed = rep(d2r(beta), length(w)),
24    horiz={BetaHoriz0=atan(abs(sin(AzS)/tan(AlS)))
25      if (BT){lew=dist$Lew/struct$L
26        Longitud=lew*cos(BetaHoriz0)
27        Cond=(Longitud>=1)
28        Longitud[Cond]=1
29        ## When Cond==TRUE Length=1
30        ## and therefore asin(Length)=pi/2,
31        ## so that BetaHoriz=BetaHoriz0
32        BetaHoriz=BetaHoriz0+asin(Longitud)-pi/2
33      } else {
34        BetaHoriz=BetaHoriz0
35        rm(BetaHoriz0)}
36      Beta=ifelse(BetaHoriz>betaLim,betaLim,BetaHoriz)}
37  )
38  is.na(Beta) <- night
39
40  Alpha<-switch(modeTrk,
41    two = AzS,
42    fixed = rep(d2r(alpha), length(w)),
43    horiz=pi/2*sign(AzS))
44  is.na(Alpha) <- night
45
46  cosTheta<-switch(modeTrk,
47    two=cos(Beta-(pi/2-AlS)),
48    horiz={
49      t1=sin(decl)*sin(lat)*cos(Beta)
50      t2=cos(decl)*cos(w)*cos(lat)*cos(Beta)
51      t3=cos(decl)*abs(sin(w))*sin(Beta)
52      cosTheta=t1+t2+t3
53      rm(t1,t2,t3)
54      cosTheta
55    },
56    fixed={

```



```

57         t1=sin(decl)*sin(lat)*cos(Beta)
58         t2=-signLat*sin(decl)*cos(lat)*sin(Beta)*cos(Alpha)
59         t3=cos(decl)*cos(w)*cos(lat)*cos(Beta)
60         t4=signLat*cos(decl)*cos(w)*sin(lat)*sin(Beta)*cos(
Alpha)
61         t5=cos(decl)*sin(w)*sin(Alpha)*sin(Beta)
62         cosTheta=t1+t2+t3+t4+t5
63         rm(t1,t2,t3,t4,t5)
64         cosTheta
65     }
66 )
67 is.na(cosTheta) <- night
68 cosTheta=cosTheta*(cosTheta>0) #when cosTheta<0, Theta is greater than 90°,
and therefore the Sun is behind the panel.
69
70 result <- data.table(Dates = indexI(sol),
71                     Beta, Alpha, cosTheta)
72 return(result)
73 }

```

EXTRACTO DE CÓDIGO A.32:  $f_{\theta}$ 

### A.3.14. HQCurve

```

1  ## HQCurve: no visible binding for global variable 'fb'
2  ## HQCurve: no visible binding for global variable 'Q'
3  ## HQCurve: no visible binding for global variable 'x'
4  ## HQCurve: no visible binding for global variable 'y'
5  ## HQCurve: no visible binding for global variable 'group.value'
6
7  if(getRversion() >= "2.15.1") globalVariables(c('fb', 'Q', 'x', 'y', 'group.
value'))
8
9  HQCurve<-function(pump){
10     w1=3000 #synchronous rpm frequency
11     wm=2870 #rpm frequency with slip when applying voltage at 50 Hz
12     s=(w1-wm)/w1
13     fen=50 #Nominal electrical frequency
14
15     f=seq(35,50,by=5)
16     Hn=with(pump,a*50^2+b*50*Qn+c*Qn^2) #height corresponding to flow rate and
nominal frequency
17
18     kiso=Hn/pump$Qn^2 #To paint the isoyield curve I take into account the laws of
similarity
19     Qiso=with(pump,seq(0.1*Qn,Qmax,l=10))
20     Hiso=kiso*Qiso^2 #Isoperformance curve
21
22     Curva<-expand.grid(fb=f,Q=Qiso)
23
24     Curva<-within(Curva,{
25         fe=fb/(1-s)
26         H=with(pump,a*fb^2+b*fb*Q+c*Q^2)
27
28         is.na(H) <- (H<0)
29         Q50=50*Q/fb
30         H50=H*(50/fb)^2
31         etab=with(pump,j*Q50^2+k*Q50+1)

```

```

32   Pb50=2.725*H50*Q50/etab
33   Pb=Pb50*(fb/50)^3
34
35   Pbc=Pb*50/fe
36   etam=with(pump,g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
37   Pmc=Pbc/etam
38   Pm=Pmc*fe/50
39
40   etac=0.95 #Variable frequency drive performance
41   cab=0.05  #Cable losses
42   Pdc=Pm/(etac*(1-cab))
43   rm(etac,cab,Pmc,Pbc,Pb50,Q50,H50)
44 })
45
46 ###H-Q curve at different frequencies
47 ##I check if I have the lattice package available, which should have been
   loaded in .First.lib
48 lattice.disp<-"lattice" %in% .packages()
49 latticeExtra.disp<-"latticeExtra" %in% .packages()
50 if (lattice.disp && latticeExtra.disp) {
51   p<-xyplot(H~Q,groups=factor(fb),data=Curva, type='l',
52             par.settings=custom.theme.2(),
53             panel=function(x,y,groups,...){
54               panel.superpose(x,y,groups,...)
55               panel.xyplot(Qiso,Hiso,col='black',...)
56               panel.text(Qiso[1], Hiso[1], 'ISO', pos=3)}
57             )
58   p=p+glayer(panel.text(x[1], y[1], group.value, pos=3))
59   print(p)
60   result<-list(result=Curva, plot=p)
61 } else {
62   warning('lattice and/or latticeExtra packages are not available. Thus, the
   plot could not be created')
63   result<-Curva}
64 }

```

EXTRACTO DE CÓDIGO A.33: *HQCurve*

## A.3.15. 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   AO=3600*dst(x)
6   AOneg=(AO<0)
7   if (any(AOneg)) {
8     AO[AOneg]=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  }

```

EXTRACTO DE CÓDIGO A.34: *local2Solar***A.3.16. markovG0**

```

1  ## Objects loaded at startup from data/MTM.RData
2  if(getRversion() >= "2.15.1") globalVariables(c(
3      'MTM', ## Markov Transition Matrices
4      'Ktmtm', ## Kt limits to choose a matrix from MTM
5      'Ktlim' ## Daily kt range of each matrix.
6  ))
7
8  markovG0 <- function(G0dm, sold){
9      sold <- copy(sold)
10     timeIndex <- sold$Dates
11     Bo0d <- sold$Bo0d
12     Bo0dm <- sold[, mean(Bo0d), by = .(month(Dates), year(Dates))][[3]]
13     ktm <- G0dm/Bo0dm
14
15     ##Calculates which matrix to work with for each month
16     whichMatrix <- findInterval(ktm, Ktmtm, all.inside = TRUE)
17
18     ktd <- state <- numeric(length(timeIndex))
19     state[1] <- 1
20     ktd[1] <- ktm[state[1]]
21     for (i in 2:length(timeIndex)){
22         iMonth <- month(timeIndex[i])
23         colMonth <- whichMatrix[iMonth]
24         rng <- Ktlim[, colMonth]
25         classes <- seq(rng[1], rng[2], length=11)
26         matMonth <- MTM[(10*colMonth-9):(10*colMonth),]
27         ## http://www-rohan.sdsu.edu/~babailey/stat575/mcsim.r
28         state[i] <- sample(1:10, size=1, prob=matMonth[state[i-1],])
29         ktd[i] <- runif(1, min=classes[state[i]], max=classes[state[i]+1])
30     }
31     G0dmMarkov <- data.table(ktd, Bo0d)
32     G0dmMarkov <- G0dmMarkov[, mean(ktd*Bo0d), by = .(month(timeIndex), year(
timeIndex))][[3]]
33     fix <- G0dm/G0dmMarkov
34     indRep <- month(timeIndex)
35     fix <- fix[indRep]
36     G0d <- data.table(Dates = timeIndex, G0d = ktd * Bo0d * fix)
37     G0d
38 }

```

EXTRACTO DE CÓDIGO A.35: *markovG0***A.3.17. NmgPVPS**

```

1  ## NmgPVPS: no visible binding for global variable 'Pnom'
2  ## NmgPVPS: no visible binding for global variable 'group.value'
3
4  if(getRversion() >= "2.15.1") globalVariables(c('Pnom', 'group.value'))
5

```

```

6 NmgPVPS <- function(pump, Pg, H, Gd, Ta=30,
7                     lambda=0.0045, TONC=47,
8                     eta=0.95, Gmax=1200, t0=6, Nm=6,
9                     title='', theme=custom.theme.2()){
10
11   ##I build the type day by IEC procedure
12   t=seq(-t0,t0,l=2*t0*Nm);
13   d=Gd/(Gmax*2*t0)
14   s=(d*pi/2-1)/(1-pi/4)
15   G=Gmax*cos(t/t0*pi/2)*(1+s*(1-cos(t/t0*pi/2)))
16   G[G<0]<-0
17   G=G/(sum(G,na.rm=1)/Nm)*Gd
18   Red<-expand.grid(G=G,Pnom=Pg,H=H,Ta=Ta)
19   Red<-within(Red,{Tcm<-Ta+G*(TONC-20)/800
20                   Pdc=Pnom*G/1000*(1-lambda*(Tcm-25)) #Available DC power
21                   Pac=Pdc*eta}) #Inverter yield
22
23   res=data.table(Red,Q=0)
24
25   for (i in seq_along(H)){
26     fun=fPump(pump, H[i])
27     Cond=res$H==H[i]
28     x=res$Pac[Cond]
29     z=res$Pdc[Cond]
30     rango=with(fun,x>=lim[1] & x<=lim[2]) #I limit the power to the
operating range of the pump.
31     x[!rango]<-0
32     z[!rango]<-0
33     y=res$Q[Cond]
34     y[rango]<-fun$fQ(x[rango])
35     res$Q[Cond]=y
36     res$Pac[Cond]=x
37     res$Pdc[Cond]=z
38   }
39
40   resumen <- res[, lapply(.SD, function(x)sum(x, na.rm = 1)/Nm),
41                   by = .(Pnom, H)]
42   param=list(pump=pump, Pg=Pg, H=H, Gd=Gd, Ta=Ta,
43             lambda=lambda, TONC=TONC, eta=eta,
44             Gmax=Gmax, t0=t0, Nm=Nm)
45
46
47   ###Abacus with common X-axes
48
49   ##I check if I have the lattice package available, which should have been
loaded in .First.lib
50   lattice.disp<-"lattice" %in% .packages()
51   latticeExtra.disp<-"latticeExtra" %in% .packages()
52   if (lattice.disp && latticeExtra.disp){
53     tema<-theme
54     tema1 <- modifyList(tema, list(layout.width = list(panel=1,
55                                                         ylab = 2, axis.left=1.0,
56                                                         left.padding=1, ylab.axis.padding=1,
57                                                         axis.panel=1)))
58     tema2 <- modifyList(tema, list(layout.width = list(panel=1,
59                                                         ylab = 2, axis.left=1.0, left.padding=1,
60                                                         ylab.axis.padding=1, axis.panel=1)))
61     temaT <- modifyList(tema, list(layout.heights = list(panel = c(1, 1))))

```

```

62     p1 <- xyplot(Q~Pdc, groups=H, data=resumen,
63                 ylab="Qd (m\u00b3/d)", type=c('l', 'g'),
64                 par.settings = tema1)
65
66     p1lab<-p1+glayer(panel.text(x[1], y[1], group.value, pos=2, cex=0.7))
67
68     ##I paint the linear regression because Pnom~Pdc depends on the height.
69     p2 <- xyplot(Pnom~Pdc, groups=H, data=resumen,
70                 ylab="Pg", type=c('l', 'g'), #type=c('smooth', 'g'),
71                 par.settings = tema2)
72     p2lab<-p2+glayer(panel.text(x[1], y[1], group.value, pos=2, cex=0.7))
73
74     p<-update(c(p1lab, p2lab, x.same = TRUE),
75              main=paste(title, '\nSP', pump$Qn, 'A', pump$stages, ' ',
76                        'Gd ', Gd/1000, " kWh/m\u00b3", sep=''),
77              layout = c(1, 2),
78              scales=list(x=list(draw=FALSE)),
79              xlab='',
80              ylab = list(c("Qd (m\u00b3/d)", "Pg (Wp)"), y = c(1/4, 3/4)),
81              par.settings = temaT
82              )
83     print(p)
84     result<-list(I=res, D=resumen, plot=p, param=param)
85 } else {
86     warning('lattice, latticeExtra packages are not all available. Thus, the
87     plot could not be created')
88     result<-list(I=res, D=resumen, param=param)
89 }

```

EXTRACTO DE CÓDIGO A.36: *NmgPVPS*

### A.3.18. solarAngles

```

1 ##### Declination #####
2 declination <- function(d, method = 'michalsky')
3 {
4     ##Method check
5     if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
6         warning("'method' must be: michalsky, cooper, strous or spencer. Set
7         michalsky")
8         method = 'michalsky'
9     }
10
11     ## x is an IDate
12     d <- as.IDate(d)
13     ## Day of year
14     dn <- yday(d)
15     ## Days from 2000-01-01
16     origin <- as.IDate('2000-01-01')
17     jd <- as.numeric(d - origin)
18     X <- 2 * pi * (dn - 1) / 365
19
20     switch(method,
21            michalsky = {
22                meanLong <- (280.460 + 0.9856474 * jd) %% 360
23                meanAnomaly <- (357.528 + 0.9856003 * jd) %% 360
24                ecliplong <- (meanLong + 1.915 * sin(d2r(meanAnomaly))) +

```

```

24         0.02 * sin(d2r(2 * meanAnomaly)))%%360
25     excen <- 23.439 - 0.0000004 * jd
26     sinEclip <- sin(d2r(eclipLong))
27     sinExcen <- sin(d2r(excen))
28     asin(sinEclip * sinExcen)
29 },
30     cooper = {
31         ##P.I. Cooper. "The Absorption of Solar Radiation in Solar Stills
32         ". Solar Energy 12 (1969).
33         d2r(23.45) * sin(2 * pi * (dn +284) / 365)
34     },
35     strous = {
36         meanAnomaly <- (357.5291 + 0.98560028 * jd)%%360
37         coefC <- c(1.9148, 0.02, 0.0003)
38         sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))
39         C <- colSums(coefC * sinC)
40         trueAnomaly <- (meanAnomaly + C)%%360
41         eclipLong <- (trueAnomaly + 282.9372)%%360
42         excen <- 23.435
43         sinEclip <- sin(d2r(eclipLong))
44         sinExcen <- sin(d2r(excen))
45         asin(sinEclip * sinExcen)
46     },
47     spencer = {
48         ## J.W. Spencer. "Fourier Series Representation of the Position
49         of the Sun". 2 (1971).
50         ##URL: http://www.mail-archive.com/sundial@uni-koeln.de/msg01050.
51         0.006918 - 0.399912 * cos(X) + 0.070257 * sin(X) -
52         0.006758 * cos(2 * X) + 0.000907 * sin(2 * X) -
53         0.002697 * cos(3 * X) + 0.001480 * sin(3 * X)
54     })
55 }
56 ##### Eccentricity #####
57 eccentricity <- function(d, method = 'michalsky')
58 {
59     ##Method check
60     if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
61         warning("'method' must be: michalsky, cooper, strous or spencer. Set
62         michalsky")
63         method = 'michalsky'
64     }
65     ##x is an IDate
66     d <- as.IDate(d)
67     ##Day of year
68     dn <- yday(d)
69     X <- 2 * pi * (dn-1)/365
70
71     switch(method,
72         cooper = 1 + 0.033*cos(2*pi*dn/365),
73         spencer = ,
74         michalsky = ,
75         strous = 1.000110 + 0.034221*cos(X) +
76         0.001280*sin(X) + 0.000719*cos(2*X) +
77         0.000077*sin(2*X)

```

```

78     )
79 }
80
81
82 ##### Equation of time
83
84 ##Alan M.Whitman "A simple expression for the equation of time"
85 ##EoT=ts-t, donde ts es la hora solar real y t es la hora solar
86 ##media. Valores negativos implican que el sol real se retrasa
87 ##respecto al medio
88 eot <- function(d)
89 {
90     ## d in an IDate
91     d <- as.IDate(d)
92     ## Day of year
93     dn <- yday(d)
94     M <- 2 * pi/365.24 * dn
95     EoT <- 229.18 * (-0.0334 * sin(M) +
96                   0.04184 * sin(2 * M + 3.5884))
97     EoT <- h2r(EoT/60)
98     return(EoT)
99 }
100
101
102 ##### Solar time #####
103 sunrise <- function(d, lat, method = 'michalsky',
104                    decl = declination(d, method = method))
105 {
106     ##Method check
107     if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
108         warning("'method' must be: michalsky, cooper, strous or spencer. Set
109         michalsky")
110         method = 'michalsky'
111     }
112
113     cosWs <- -tan(d2r(lat)) * tan(decl)
114     #sunrise, negative since it is before noon
115     ws <- -acos(cosWs)
116     #Polar day/night
117     polar <- which(is.nan(ws))
118     ws[polar] <- -pi * (cosWs[polar] < -1) + 0 * (cosWs[polar] > 1)
119     return(ws)
120 }
121
122 ##### Extraterrestrial irradiation #####
123 bo0d <- function(d, lat, method = 'michalsky',
124                 decl = declination(d, method = method),
125                 eo = eccentricity(d, method = method),
126                 ws = sunrise(d, lat, method = method))
127 {
128     ##Method check
129     if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
130         warning("'method' must be: michalsky, cooper, strous or spencer. Set
131         michalsky")
132         method = 'michalsky'
133     }
134
135     #solar constant

```

```

134 Bo <- 1367
135 latr <- d2r(lat)
136 #The negative sign due to the definition of ws
137 Bo0d <- -24/pi * Bo * eo * (ws * sin(latr) * sin(decl) +
138                               cos(latr) * cos(decl) * sin(ws))
139 return(Bo0d)
140 }
141
142
143 ##### Sun hour angle #####
144 sunHour <- function(d, BTi, sample = 'hour', EoT = TRUE,
145                     method = 'michalsky',
146                     eqtime = eot(d))
147 {
148   ##Method check
149   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
150     warning("'method' must be: michalsky, cooper, strous or spencer. Set
151     michalsky")
152     method = 'michalsky'
153   }
154
155   if(missing(BTi)){
156     BTi <- fBTi(BTd = d, sample = sample)
157   }else {
158     if (inherits(BTi, 'data.table')) {
159       Times <- as.ITime(BTi$Times)
160       Dates <- as.IDate(BTi$Dates)
161       BTi <- as.POSIXct(Dates, Times, tz = 'UTC')
162     }
163     else {
164       BTi <- as.POSIXct(BTi, tz = 'UTC')
165     }
166   }
167   rep <- cumsum(c(1, diff(as.Date(BTi)) != 0))
168   if(EoT)
169   {
170     EoT <- eqtime
171     if(length(EoT) != length(BTi)){EoT <- EoT[rep]}
172   }else{EoT <- 0}
173
174   jd <- as.numeric(julian(BTi, origin = '2000-01-01 12:00:00 UTC'))
175   T0 <- hms(BTi)
176
177   w=switch(method,
178     cooper = h2r(T0-12)+EoT,
179     spencer = h2r(T0-12)+EoT,
180     michalsky = {
181       meanLong <- (280.460+0.9856474*jd)%%360
182       meanAnomaly <- (357.528+0.9856003*jd)%%360
183       ecliplong <- (meanLong +1.915*sin(d2r(meanAnomaly))+0.02*sin(
184         d2r(2*meanAnomaly)))%%360
185       excen <- 23.439-0.0000004*jd
186
187       sinEclip <- sin(d2r(ecliplong))
188       cosEclip <- cos(d2r(ecliplong))
189       cosExcen <- cos(d2r(excen))
190
191       ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))%%360

```



```

190
191     ##local mean sidereal time, LMST
192     ##TO has been previously corrected with local2Solar in order
193     ##to include the longitude, daylight savings, etc.
194     lmst <- (h2d(6.697375 + 0.0657098242*jd + TO))%%360
195     w <- (lmst-ascension)
196     w <- d2r(w + 360*(w < -180) - 360*(w > 180))
197   },
198   strous = {
199     meanAnomaly <- (357.5291 + 0.98560028*jd)%%360
200     coefC <- c(1.9148, 0.02, 0.0003)
201     sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))
202     C <- colSums(coefC*sinC)
203     trueAnomaly <- (meanAnomaly + C)%%360
204     eclipLong <- (trueAnomaly + 282.9372)%%360
205     excen <- 23.435
206
207     sinEclip <- sin(d2r(eclipLong))
208     cosEclip <- cos(d2r(eclipLong))
209     cosExcen <- cos(d2r(excen))
210
211     ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))%%360
212
213     ##local mean sidereal time, LMST
214     ##TO has been previously corrected with local2Solar in order
215     ##to include the longitude, daylight savings, etc.
216     lmst <- (280.1600+360.9856235*jd)%%360
217     w <- (lmst-ascension)
218     w <- d2r(w + 360*(w< -180) - 360*(w>180))
219   }
220 )
221 return(w)
222 }
223
224 ##### zenith angle #####
225 zenith <- function(d, lat, BTi, sample = 'hour', method = 'michalsky',
226                   decl = declination(d, method = method),
227                   w = sunHour(d, BTi, sample, method = method))
228 {
229   ##Method check
230   if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
231     warning("'method' must be: michalsky, cooper, strous or spencer. Set
232     michalsky")
233     method = 'michalsky'
234   }
235
236   if(missing(BTi)){BTi <- fBTi(d, sample)}
237   x <- as.Date(BTi)
238   rep <- cumsum(c(1, diff(x) != 0))
239   latr <- d2r(lat)
240   if(length(decl) == length(BTi)){decl <- decl}
241   else{decl <- decl[rep]}
242   zenith <- sin(decl) * sin(latr) +
243     cos(decl) * cos(w) * cos(latr)
244   zenith <- ifelse(zenith > 1, 1, zenith)
245   return(zenith)
246 }

```

```

247 ##### azimuth #####
248 azimuth <- function(d, lat, BTi, sample = 'hour', method = 'michalsky',
249                     decl = declination(d, method = method),
250                     w = sunHour(d, BTi, sample, method = method),
251                     cosThzS = zenith(d, lat, BTi, sample,
252                                     method = method,
253                                     decl = decl,
254                                     w = w))
255 {
256     ##Method check
257     if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
258         warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
259         method = 'michalsky'
260     }
261
262     signLat <- ifelse(sign(lat) == 0, 1, sign(lat)) #if the sign of lat is 0, it
changes it to 1
263     if(missing(BTi)){BTi <- fBTi(d, sample)}
264     x <- as.Date(BTi)
265     rep <- cumsum(c(1, diff(x) != 0))
266     latr <- d2r(lat)
267     if(length(decl) != length(BTi)){decl <- decl[rep]}
268     ALS <- asin(cosThzS)
269     cosazimuth <- signLat * (cos(decl) * cos(w) * sin(latr) -
270                           cos(latr) * sin(decl)) / cos(ALS)
271     cosazimuth <- ifelse(abs(cosazimuth)>1, sign(cosazimuth), cosazimuth)
272     azimuth <- sign(w)*acos(cosazimuth)
273     return(azimuth)
274 }

```

EXTRACTO DE CÓDIGO A.37: *solarAngles***A.3.19. utils-angles**

```

1 #degrees to radians
2 d2r<-function(x){x*pi/180}
3
4 #radians to degrees
5 r2d<-function(x){x*180/pi}
6
7 #hours to radians
8 h2r<-function(x){x*pi/12}
9
10 #hours to degrees
11 h2d<-function(x){x*180/12}
12
13 #radians to hours
14 r2h<-function(x){x*12/pi}
15
16 #degrees to hours
17 d2h<-function(x){x*12/180}
18
19 #radians to seconds
20 r2sec<-function(x){x*12/pi*3600}
21
22 #radians to minutes
23 r2min<-function(x){x*12/pi*60}

```

EXTRACTO DE CÓDIGO A.38: *utils-angles*A.3.20. *utils-time*

```

1 #complete time to hours
2 t2h <- function(x)
3 {
4   hour(x)+minute(x)/60+second(x)/3600
5 }
6
7 #hours minutes and seconds to hours
8 hms <- function(x)
9 {
10  hour(x)+minute(x)/60+second(x)/3600
11 }
12
13 #day of the year
14 doy <- function(x){
15   as.numeric(format(x, '%j'))
16 }
17
18 #day of the month
19 dom <- function(x){
20   as.numeric(format(x, '%d'))
21 }
22
23 #trunc days
24 truncDay <- function(x){as.POSIXct(trunc(x, units='days'))}

```

EXTRACTO DE CÓDIGO A.39: *utils-time*

## A.4. Métodos

A.4.1. *as.data.tableI*

```

1 setGeneric('as.data.tableI',
2   function(object, complete=FALSE, day=FALSE){standardGeneric('as.data.
3     tableI')}})
4
5 setMethod('as.data.tableI',
6   signature=(object='Sol'),
7   definition=function(object, complete=FALSE, day=FALSE){
8     sol <- copy(object)
9     BTi <- indexI(sol)
10    BTi <- truncDay(BTi)
11    ind.rep <- cumsum(c(1, diff(BTi, units='days')!=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)]

```

```

21     }
22     return(data)
23   }
24 )
25
26 setMethod('as.data.tableI',
27   signature = (object='G0'),
28   definition = function(object, complete=FALSE, day=FALSE){
29     g0 <- copy(object)
30     BTi <- indexI(g0)
31     BTi <- truncDay(BTi)
32     ind.rep <- cumsum(c(1, diff(BTi)!=0))
33     GOI <- g0@GOI
34     solI <- g0@solI
35     sold <- g0@sold[ind.rep]
36     Ta <- g0@Ta
37     if(length(Ta[[1]]!=length(GOI[[1]]))) Ta <- Ta[ind.rep]
38     if(complete){
39       data <- data.table(solI,
40                           GOI[, Dates := NULL],
41                           sold[, 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     BTi <- indexI(gef)
63     BTi <- truncDay(BTi)
64     ind.rep <- cumsum(c(1, diff(BTi, units='days')!=0))
65     GefI <- gef@GefI
66     GOI <- gef@GOI
67     solI <- gef@solI
68     sold <- gef@sold[ind.rep]
69     Ta <- gef@Ta
70     if(length(Ta[[1]]!=length(GefI[[1]]))) Ta <- Ta[ind.rep]
71     if(complete){
72       data <- data.table(solI,
73                           GOI[, Dates := NULL],
74                           sold[, Dates := NULL],
75                           Ta[, Dates := NULL],
76                           GefI[, Dates := NULL])
77     } else {
78       data <- GefI[, c('Dates', 'Gef',

```

```

79         'Bef', 'Def'])
80     }
81     if(day){
82         ind <- indexI(object)
83         data[, day := doy(ind)]
84         data[, month := month(ind)]
85         data[, year := year(ind)]
86     }
87     return(data)
88 }
89 )
90
91 setMethod('as.data.tableI',
92     signature = (object='ProdGCPV'),
93     definition = function(object, complete=FALSE, day=FALSE){
94         prodgcpv <- copy(object)
95         BTi <- indexI(prodgcpv)
96         BTi <- truncDay(BTi)
97         ind.rep <- cumsum(c(1, diff(BTi, units = 'days')!=0))
98         prodI <- prodgcpv@prodI
99         Theta <- prodgcpv@Theta
100        GefI <- prodgcpv@GefI
101        GOI <- prodgcpv@GOI
102        solI <- prodgcpv@solI
103        sold <- prodgcpv@sold[ind.rep]
104        Ta <- prodgcpv@Ta
105        if(length(Ta[[1]]!=length(prodI[[1]]))) Ta <- Ta[ind.rep]
106        if(complete){
107            data <- data.table(solI,
108                               GOI[, Dates := NULL],
109                               sold[, Dates := NULL],
110                               Ta[, Dates := NULL],
111                               GefI[, Dates := NULL],
112                               prodI[, Dates := NULL],
113                               Theta[, Dates := NULL])
114        } else {
115            data <- prodI[, c('Dates', 'Pac', 'Pdc')]
116        }
117        if(day){
118            ind <- indexI(object)
119            data[, day := doy(ind)]
120            data[, month := month(ind)]
121            data[, year := year(ind)]
122        }
123        return(data)
124    }
125 )
126
127 setMethod('as.data.tableI',
128     signature = (object='ProdPVPS'),
129     definition = function(object, complete=FALSE, day=FALSE){
130         prodpvps <- copy(object)
131         BTi <- indexI(prodpvps)
132         BTi <- truncDay(BTi)
133         ind.rep <- cumsum(c(1, diff(BTi, units='days')!=0))
134         prodI <- prodpvps@prodI
135         Theta <- prodpvps@Theta
136         GefI <- prodpvps@GefI

```

```

137     GOI <- prodpvps@GOI
138     solI <- prodpvps@solI
139     sold <- prodpvps@sold[ind.rep]
140     Ta <- prodpvps@Ta
141     if(length(Ta[[1]]!=length(prodI[[1]]))) Ta <- Ta[ind.rep]
142     if(complete){
143         data <- data.table(solI,
144                             GOI[, Dates := NULL],
145                             sold[, Dates := NULL],
146                             Ta[, Dates := NULL],
147                             GefI[, Dates := NULL],
148                             prodI[, Dates := NULL],
149                             Theta[, Dates := NULL])
150     } else {
151         data <- prodI[, c('Dates', 'Pac', 'Pdc')]
152     }
153     if(day){
154         ind <- indexI(object)
155         data[, day := doy(ind)]
156         data[, month := month(ind)]
157         data[, year := year(ind)]
158     }
159     return(data)
160 }
161 )

```

EXTRACTO DE CÓDIGO A.40: *as.data.tableI*A.4.2. *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(object)
12             data[, day := doy(ind)]
13             data[, month := month(ind)]
14             data[, year := year(ind)]
15         }
16         return(data)
17     }
18 )
19
20 setMethod('as.data.tableD',
21     signature = (object='G0'),
22     definition = function(object, complete=FALSE, day=FALSE){
23         g0 <- copy(object)
24         GOD <- g0@GOD
25         sold <- g0@sold
26         if(complete){
27             data <- data.table(GOD, sold[, Dates := NULL])
28         } else {

```

```

28         GOD[, Fd := NULL]
29         GOD[, Kt := NULL]
30         data <- GOD
31     }
32     if(day){
33         ind <- indexD(object)
34         data[, day := doy(ind)]
35         data[, month := month(ind)]
36         data[, year := year(ind)]
37     }
38     return(data)
39 })
40
41 setMethod('as.data.tableD',
42     signature = (object='Gef'),
43     definition = function(object, complete=FALSE, day=FALSE){
44         gef <- copy(object)
45         GefD <- gef@GefD
46         GOD <- gef@GOD
47         sold <- gef@sold
48         if(complete){
49             data <- data.table(GefD,
50                                 GOD[, Dates := NULL],
51                                 sold[, Dates := NULL])
52         } else {data <- GefD[, c('Dates', 'Gefd',
53                                 'Defd', 'Befd')]}
54         if(day){
55             ind <- indexD(object)
56             data[, day := doy(ind)]
57             data[, month := month(ind)]
58             data[, year := year(ind)]
59         }
60         return(data)
61     }
62 )
63
64 setMethod('as.data.tableD',
65     signature = (object='ProdGCPV'),
66     definition = function(object, complete=FALSE, day=FALSE){
67         prodgcpv <- copy(object)
68         prodD <- prodgcpv@prodD
69         GefD <- prodgcpv@GefD
70         GOD <- prodgcpv@GOD
71         sold <- prodgcpv@sold
72         if(complete){
73             data <- data.table(prodD,
74                                 GefD[, Dates := NULL],
75                                 GOD[, Dates := NULL],
76                                 sold[, Dates := NULL])
77         } else { data <- prodD[, c('Dates', 'Eac',
78                                 'Edc', 'Yf')]}
79         if(day){
80             ind <- indexD(object)
81             data[, day := doy(ind)]
82             data[, month := month(ind)]
83             data[, year := year(ind)]
84         }
85     }

```

```

86         return(data)
87     }
88 )
89
90 setMethod('as.data.tableD',
91   signature = (object='ProdPVPS'),
92   definition = function(object, complete=FALSE, day=FALSE){
93     prodpvps <- copy(object)
94     prodD <- prodpvps@prodD
95     GefD <- prodpvps@GefD
96     GOD <- prodpvps@GOD
97     sold <- prodpvps@sold
98     if(complete){
99       data <- data.table(prodD,
100                           GefD[, Dates := NULL],
101                           GOD[, Dates := NULL],
102                           sold[, Dates := NULL]
103                         )
104     } else { data <- prodD[, c('Dates', 'Eac',
105                               'Qd', 'Yf')]}
106     if(day){
107       ind <- indexD(object)
108       data[, day := doy(ind)]
109       data[, month := month(ind)]
110       data[, year := year(ind)]
111     }
112     return(data)
113   }
114 )

```

EXTRACTO DE CÓDIGO A.41: *as.data.tableD*A.4.3. *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='G0'),
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)]
13      data[, year := year(ind)]
14    }
15    return(data)
16  }
17 )
18
19 setMethod('as.data.tableM',
20   signature=(object='Gef'),
21   definition = function(object, complete=FALSE, day=FALSE){
22     gef <- copy(object)
23     Gefdm <- gef@Gefdm
24     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 )

```

EXTRACTO DE CÓDIGO A.42: *as.data.tableM*

#### A.4.4. as.data.tableY

```

1  setGeneric('as.data.tableY', function(object, complete=FALSE, day=FALSE){
    standardGeneric('as.data.tableY')})
2
3  setMethod('as.data.tableY',
4    signature=(object='G0'),
5    definition=function(object, complete=FALSE, day=FALSE){
6      g0 <- copy(object)
7      G0y <- g0@G0y
8      data <- G0y
9      if(day){data[, year := Dates]}
10     return(data)
11   }
12 )
13
14 setMethod('as.data.tableY',
15   signature = (object='Gef'),
16   definition = function(object, complete=FALSE, day=FALSE){
17     gef <- copy(object)
18     Gefy <- gef@Gefy
19     G0y <- gef@G0y
20     if(complete){
21       data <- data.table(Gefy, G0y[, Dates := NULL])
22     } else {data <- Gefy}
23     if(day){data[, year := Dates]}
24     return(data)
25   }
26 )
27
28 setMethod('as.data.tableY',
29   signature = (object='ProdGCPV'),
30   definition = function(object, complete=FALSE, day=FALSE){
31     prodgcpv <- copy(object)
32     prody <- prodgcpv@prody
33     Gefy <- prodgcpv@Gefy
34     G0y <- prodgcpv@G0y
35     if(complete){
36       data <- data.table(prody,
37                           Gefy[, Dates := NULL],
38                           G0y[, Dates := NULL])
39     } else {data <- prody}
40     if(day){data[, year := Dates]}
41     return(data)
42   }
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 := Dates]}

```

```

58         return(data)
59     }
60 )

```

EXTRACTO DE CÓDIGO A.43: *as.data.tableY*

#### A.4.5. compare

```

1  ## compareFunction: no visible binding for global variable 'name'
2  ## compareFunction: no visible binding for global variable 'x'
3  ## compareFunction: no visible binding for global variable 'y'
4  ## compareFunction: no visible binding for global variable 'group.value'
5
6  if(getRversion() >= "2.15.1") globalVariables(c('name', 'x', 'y', 'group.value',
7  ..vars'))
8
9  setGeneric('compare', signature='...', function(...){standardGeneric('compare')
10  })
11
12  compareFunction <- function(..., vars){
13      dots <- list(...)
14      nms0 <- substitute(list(...))
15      if (!is.null(names(nms0))){ ##in do.call
16          nms <- names(nms0[-1])
17      } else {
18          nms <- as.character(nms0[-1])
19      }
20      foo <- function(object, label){
21          yY <- colMeans(as.data.tableY(object, complete = TRUE)[, ..vars])
22          yY <- cbind(stack(yY), name=label)
23          yY
24      }
25      cdata <- mapply(FUN=foo, dots, nms, SIMPLIFY=FALSE)
26      z <- do.call(rbind, cdata)
27      z$ind <- ordered(z$ind, levels=vars)
28      p <- dotplot(ind~values, groups=name, data=z, type='b',
29      par.settings=solaR.theme)
30      print(p+glayer(panel.text(x[length(x)], y[length(x)],
31      label=group.value, cex=0.7, pos=3, srt=45)))
32      return(z)
33  }
34
35  setMethod('compare',
36      signature='G0',
37      definition=function(...){
38          vars <- c('D0d', 'B0d', 'G0d')
39          res <- compareFunction(..., vars=vars)
40          return(res)
41      })
42
43  setMethod('compare',
44      signature='Gef',
45      definition=function(...){
46          vars <- c('Defd', 'Befd', 'Gefd')
47          res <- compareFunction(..., vars=vars)
48          return(res)

```

```
49     }
50   )
51
52   setMethod('compare',
53             signature='ProdGCPV',
54             definition=function(...){
55               vars <- c('G0d', 'Gefd', 'Yf')
56               res <- compareFunction(..., vars=vars)
57               return(res)
58             }
59   )
```

EXTRACTO DE CÓDIGO A.44: *compare*

#### A.4.6. *getData*

```
1  ## extracts the data for class Meteo ##
2  setGeneric('getData', function(object){standardGeneric('getData')})
3
4  ### getData ###
5  setMethod('getData',
6            signature = (object = 'Meteo'),
7            definition = function(object){
8              result <- object@data
9              return(result)
10           })
```

EXTRACTO DE CÓDIGO A.45: *getData*

#### A.4.7. *getG0*

```
1  ## extracts the global irradiance for class Meteo ##
2  setGeneric('getG0', function(object){standardGeneric('getG0')})
3
4  ### getG0 ###
5  setMethod('getG0',
6            signature = (object = 'Meteo'),
7            definition = function(object){
8              result <- getData(object)
9              return(result$G0)
10           })
```

EXTRACTO DE CÓDIGO A.46: *getG0*

#### A.4.8. *getLat*

```
1  ## extracts the latitude from the objects ##
2  setGeneric('getLat', function(object, units = 'rad')
3  {standardGeneric('getLat')})
4
5  ## extracts the latitude from the objects ##
6  setGeneric('getLat', function(object, units = 'rad')
7  {standardGeneric('getLat')})
8
9  setMethod('getLat',
10            signature = (object = 'Meteo'),
11            definition = function(object, units = 'rad'){
12              stopifnot(units %in% c('deg', 'rad'))
```

```

13         result = switch(units,
14                         rad = d2r(object@latm),
15                         deg = object@latm)
16         return(result)
17     })

```

EXTRACTO DE CÓDIGO A.47: *getLat*

#### A.4.9. indexD

```

1  ## extract the index of the daily data ##
2  setGeneric('indexD', function(object){standardGeneric('indexD')})
3  ### indexD ###
4  setMethod('indexD',
5            signature = (object = 'Sol'),
6            definition = function(object){as.POSIXct(object@solD$Dates)
7            })
8
9  setMethod('indexD',
10           signature = (object = 'Meteo'),
11           definition = function(object){as.POSIXct(getData(object)$Dates)})

```

EXTRACTO DE CÓDIGO A.48: *indexD*

#### A.4.10. indexI

```

1  ## extract the index of the intradaily data ##
2  setGeneric('indexI', function(object){standardGeneric('indexI')})
3  ### indexI ###
4  setMethod('indexI',
5            signature = (object = 'Sol'),
6            definition = function(object){as.POSIXct(object@solI$Dates)
7            })

```

EXTRACTO DE CÓDIGO A.49: *indexI*

#### A.4.11. 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 = TRUE
8            ,
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,

```

```

20         yscale.components = yscale.components,
21         panel = panel, interpolate = interpolate,
22         ...)
23     }
24 )
25
26 setMethod('levelplot',
27           signature=c(x='formula', data='Sol'),
28           definition=function(x, data,
29                             par.settings = solaR.theme,
30                             panel = panel.levelplot.raster, interpolate = TRUE
31           ,
32           xscale.components = xscale.solar,
33           yscale.components = yscale.solar,
34           ...){
35     data0=as.data.tableI(data, complete=TRUE, day=TRUE)
36     ind=data0$Dates
37     data0$day=doy(ind)
38     data0$month=month(ind)
39     data0$year=year(ind)
40     levelplot(x, data0,
41              par.settings = par.settings,
42              xscale.components = xscale.components,
43              yscale.components = yscale.components,
44              panel = panel, interpolate = interpolate,
45              ...)
46   }
47 )
48
49 setMethod('levelplot',
50           signature=c(x='formula', data='G0'),
51           definition=function(x, data,
52                             par.settings = solaR.theme,
53                             panel = panel.levelplot.raster, interpolate = TRUE
54           ,
55           xscale.components = xscale.solar,
56           yscale.components = yscale.solar,
57           ...){
58     data0=as.data.tableI(data, complete=TRUE, day=TRUE)
59     ind=data0$Dates
60     data0$day=doy(ind)
61     data0$month=month(ind)
62     data0$year=year(ind)
63     levelplot(x, data0,
64              par.settings = par.settings,
65              xscale.components = xscale.components,
66              yscale.components = yscale.components,
67              panel = panel, interpolate = interpolate,
68              ...)
69   }
70 )

```

EXTRACTO DE CÓDIGO A.50: *levelplot***A.4.12. losses**

```

1  utils::globalVariables(c('Vmpp', 'Impp', 'Pdc', 'EffI', 'V1'))
2

```

```

3 setGeneric('losses', function(object){standardGeneric('losses')})
4
5 setMethod('losses',
6   signature=(object='Gef'),
7   definition=function(object){
8     dat <- as.data.tableY(object, complete=TRUE)
9     isShd=('Gef0d' %in% names(dat)) ##is there shadows?
10    if (isShd) {
11      shd <- with(dat, mean(1-Gefd/Gef0d))
12      eff <- with(dat, mean(1-Gef0d/Gd))
13    } else {
14      shd <- 0
15      eff <- with(dat, mean(1-Gefd/Gd))
16    }
17    result <- data.table(Shadows = shd, AoI = eff)
18    result <- melt(result, measure.vars = names(result),
19                  variable.name = 'id')
20  }
21 )
22
23 setMethod('losses',
24   signature=(object='ProdGCPV'),
25   definition=function(object){
26     datY <- as.data.tableY(object, complete=TRUE)
27     module0=object@module
28     module0$CoefVT=0 ##No losses with temperature
29     Pg=object@generator$Pg
30     datI <- as.data.tableI(object, complete=TRUE)
31     if (object@type=='prom'){
32       YfDC0 <- datI[, P2E(Vmpp*Imp, object@sample),
33                     by = .(month(Dates), year(Dates))]
34       YfDC0 <- YfDC0[, V1 := V1/Pg*DOM(YfDC0)]
35       YfDC0 <- sum(YfDC0$V1)
36       YfAC0 <- datI[, P2E(Pdc*EffI, object@sample),
37                     by = .(month(Dates), year(Dates))]
38       YfAC0 <- YfAC0[, V1 := V1/Pg*DOM(YfAC0)]
39       YfAC0 <- sum(YfAC0$V1)
40     } else {
41       YfDC0 <- datI[, P2E(Vmpp*Imp, object@sample),
42                     by = year(Dates)]
43       YfDC0 <- YfDC0[, V1 := V1/Pg]
44       YfDC0 <- sum(YfDC0$V1)
45       YfAC0 <- datI[, P2E(Pdc*EffI, object@sample),
46                     by = year(Dates)]
47       YfAC0 <- YfAC0[, V1 := V1/Pg]
48       YfAC0 <- sum(YfAC0$V1)
49     }
50     gen <- mean(1-YfDC0/datY$Gefd)
51     YfDC <- datY$Edc/Pg*1000
52     DC=mean(1-YfDC/YfDC0)
53     inv=mean(1-YfAC0/YfDC)
54     AC=mean(1-datY$Yf/YfAC0)
55     result0 <- losses(as(object, 'Gef'))
56     result1 <- data.table(Generator = gen,
57                           DC = DC,
58                           Inverter = inv,
59                           AC = AC)
60     result1 <- melt(result1, measure.vars = names(result1),

```

```

61         variable.name = 'id')
62     result <- rbind(result0, result1)
63     result
64 }
65 )
66
67 ###compareLosses
68
69 ## compareLosses,ProdGCPV: no visible binding for global variable 'name'
70 if(getRversion() >= "2.15.1") globalVariables(c('name'))
71
72 setGeneric('compareLosses', signature='...', function(...){standardGeneric('
    compareLosses')})
73
74 setMethod('compareLosses', 'ProdGCPV',
75     definition=function(...){
76         dots <- list(...)
77         nms0 <- substitute(list(...))
78         if (!is.null(names(nms0))) { ##do.call
79             nms <- names(nms0[-1])
80         } else {
81             nms <- as.character(nms0[-1])
82         }
83         foo <- function(object, label){
84             yY <- losses(object)
85             yY <- cbind(yY, name=label)
86             yY
87         }
88         cdata <- mapply(FUN=foo, dots, nms, SIMPLIFY=FALSE)
89         z <- do.call(rbind, cdata)
90         z$id <- ordered(z$id, levels=c('Shadows', 'AoI', 'Generator',
91             'DC', 'Inverter', 'AC'))
92         p <- dotplot(id~value*100, groups=name, data=z,
93             par.settings=solaR.theme, type='b',
94             auto.key=list(corner=c(0.95,0.2), cex=0.7), xlab='
    Losses (%)')
95         print(p)
96         return(z)
97     }
98 )

```

EXTRACTO DE CÓDIGO A.51: *losses*

#### A.4.13. mergeSolar

```

1  setGeneric('mergesolaR', signature='...', function(...){standardGeneric('
    mergesolaR')})
2
3  fooMeteo <- function(object, var){yY <- getData(object)[, .SD,
4                                     by = Dates,
5                                     .SDcols = var]}
6
7  fooGO <- function(object, var){yY <- as.data.tableD(object)[, .SD,
8                                     by = Dates,
9                                     .SDcols = var]}
10
11 mergeFunction <- function(..., foo, var){
12     dots <- list(...)

```



```

13   dots <- lapply(dots, as, class(dots[[1]])) ##the first element is the one
    that dictates the class to everyone
14   nms0 <- substitute(list(...))
15   if (!is.null(names(nms0))) { ##do.call
16     nms <- names(nms0[-1])
17   } else {
18     nms <- as.character(nms0[-1])
19   }
20   cdata <- sapply(dots, FUN=foo, var, simplify=FALSE)
21   z <- cdata[[1]]
22   for (i in 2:length(cdata)){
23     z <- merge(z, cdata[i], by = 'Dates', suffixes = c("", paste0('.', i))
24   )
25   }
26   names(z)[-1] <- nms
27   z
28 }
29 setMethod('mergesolaR',
30           signature='Meteo',
31           definition=function(...){
32             res <- mergeFunction(..., foo=fooMeteo, var='G0')
33             res
34           }
35 )
36
37 setMethod('mergesolaR',
38           signature='G0',
39           definition=function(...){
40             res <- mergeFunction(..., foo=fooG0, var='G0d')
41             res
42           }
43 )
44
45 setMethod('mergesolaR',
46           signature='Gef',
47           definition=function(...){
48             res <- mergeFunction(..., foo=fooG0, var='Gefd')
49             res
50           }
51 )
52
53 setMethod('mergesolaR',
54           signature='ProdGCPV',
55           definition=function(...){
56             res <- mergeFunction(..., foo=fooG0, var='Yf')
57             res
58           }
59 )
60
61 setMethod('mergesolaR',
62           signature='ProdPVPS',
63           definition=function(...){
64             res <- mergeFunction(..., foo=fooG0, var='Yf')
65             res
66           }
67 )

```

EXTRACTO DE CÓDIGO A.52: *mergeSolar*A.4.14. *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),l=n+1),3)
30      pruebaCB<-("RColorBrewer" %in% .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, #levels=niveles,
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, #levels=niveles,
45            col="black", labcex=.8, add=TRUE)
46          contour(Lew, Lns, GRR.m,
47            col="black", lty=3, labcex=.8, add=
48            TRUE)
49          grid(col="white",lty=3)},
50        key.title=title("1-FS",cex.main=.8))
51    }
52    if (mode=='horiz') {
53      Lew=seq(min(red$Lew),max(red$Lew),length=100)

```

```

53         FS=predict(FS.loess,Lew)
54         GRR=Lew/struct$L
55         plot(GRR,1-FS,main=main,type='l',...)
56         grid()      }
57     if (mode=='fixed'){
58         D=seq(min(red$D),max(red$D),length=100)
59         FS=predict(FS.loess,D)
60         GRR=D/struct$L
61         plot(GRR,1-FS,main=main,type='l',...)
62         grid()      }
63     }
64 )

```

EXTRACTO DE CÓDIGO A.53: *shadeplot*

## A.4.15. 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 ##The end is the last second of the day
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##The end is the last second of the
35             day
36         } else {
37             nDays <- length(indexD(x))
38             j <- indexD(x)[nDays]+86400-1
39         }
40         stopifnot(j>i)
41         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='G0',
52   definition=function(x, i, j, ...){
53     sol <- as(x, 'Sol')[i=i, j=j, ...] ##Sol method
54     meteo <- as(x, 'Meteo')[i=i, j=j, ...] ##Meteo method
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@G0D[(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('G0',
80       meteo,
81       sol,
82       G0D=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, 'G0')[i=i, j=j, ...] ##G0 method
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('Bod', 'Bnd', 'Gd', 'Dd',
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, ...] ##Gef method
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 <- prodDmw[, 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   setcolorder(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, ...] ##Gef method
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             setcolorder(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 )

```

EXTRACTO DE CÓDIGO A.54: *window*

#### A.4.16. 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, day=FALSE,
9                               timeScales=c('i', 'd', 'm', 'y'), sep=',', ...){
10      name <- strsplit(file, '\\.')[[1]][1]
11      ext <- strsplit(file, '\\.')[[1]][2]
12      timeScales <- match.arg(timeScales, several.ok=TRUE)
13      if ('i' %in% timeScales) {
14          zI <- as.data.tableI(object, complete=complete, day=day)
15          write.table(zI,
16                     file=file, sep=sep, row.names = FALSE, ...)
17      }
18      if ('d' %in% timeScales) {
19          zD <- as.data.tableD(object, complete=complete, day = day)
20          write.table(zD,
21                     file=paste(name, 'D', ext, sep='.'),
22                     sep=sep, row.names = FALSE, ...)
23      }
24      if ('m' %in% timeScales) {
25          zM <- as.data.tableM(object, complete=complete, day = day)
26          write.table(zM,
27                     file=paste(name, 'M', ext, sep='.'),
28                     sep=sep, row.names = FALSE, ...)
29      }
30      if ('y' %in% timeScales) {
31          zY <- as.data.tableY(object, complete=complete, day = day)
32          write.table(zY,
33                     file=paste(name, 'Y', ext, sep='.'),
34                     sep=sep, row.names = FALSE, ...)
35      }
36  })

```

EXTRACTO DE CÓDIGO A.55: *writeSolar*

## A.4.17. xyplot

```

1  utils::globalVariables('variable')
2
3  #####
4  ## THEMES
5  #####
6  xscale.solar <- function(...){ans <- xscale.components.default(...); ans$top=
    FALSE; ans}
7  yscale.solar <- function(...){ans <- yscale.components.default(...); ans$right=
    FALSE; ans}
8
9  solaR.theme <- function(pch=19, cex=0.7, region=rev(brewer.pal(9, 'YlOrRd')),
    ...) {
10     theme <- custom.theme.2(pch=pch, cex=cex, region=region, ...)
11     theme$strip.background$col='transparent'
12     theme$strip.shingle$col='transparent'
13     theme$strip.border$col='transparent'
14     theme
15 }
16
17 solaR.theme.2 <- function(pch=19, cex=0.7, region=rev(brewer.pal(9, 'YlOrRd')),
    ...) {
18     theme <- custom.theme.2(pch=pch, cex=cex, region=region, ...)
19     theme$strip.background$col='lightgray'
20     theme$strip.shingle$col='lightgray'
21     theme
22 }
23
24 #####
25 ## XYPLOT
26 #####
27 setGeneric('xyplot')
28
29 setMethod('xyplot',
30     signature = c(x = 'data.table', data = 'missing'),
31     definition = function(x, data,
32         par.settings = solaR.theme.2,
33         xscale.components=xscale.solar,
34         yscale.components=yscale.solar,
35         scales = list(y = 'free'),
36         ...){
37         N <- length(x)-1
38         x0 <- x[, lapply(.SD, as.numeric), by = Dates]
39         x0 <- melt(x0, id.vars = 'Dates')
40         x0$variable <- factor(x0$variable,
41             levels = rev(levels(factor(x0$variable))))
42         xyplot(value ~ Dates | variable, x0,
43             par.settings = par.settings,
44             xscale.components = xscale.components,
45             yscale.components = yscale.components,
46             scales = scales,
47             type = 'l', layout = c(1,N),
48             ...)
49     })
50
51 setMethod('xyplot',
52     signature=c(x='formula', data='Meteo'),
53     definition=function(x, data,

```



```

54         par.settings=solaR.theme,
55         xscale.components=xscale.solar,
56         yscale.components=yscale.solar,
57         ...){
58     data0=getData(data)
59     xyplot(x, data0,
60         par.settings = par.settings,
61         xscale.components = xscale.components,
62         yscale.components = yscale.components,
63         strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
64     }
65 )
66
67 setMethod('xyplot',
68     signature=c(x='formula', data='Sol'),
69     definition=function(x, data,
70         par.settings=solaR.theme,
71         xscale.components=xscale.solar,
72         yscale.components=yscale.solar,
73         ...){
74         data0=as.data.tableI(data, complete=TRUE, day=TRUE)
75         data0[, w := h2r(hms(Dates)-12)]
76         xyplot(x, data0,
77             par.settings = par.settings,
78             xscale.components = xscale.components,
79             yscale.components = yscale.components,
80             strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
81         }
82     )
83
84 setMethod('xyplot',
85     signature=c(x='formula', data='GO'),
86     definition=function(x, data,
87         par.settings=solaR.theme,
88         xscale.components=xscale.solar,
89         yscale.components=yscale.solar,
90         ...){
91         data0=as.data.tableI(data, complete=TRUE, day=TRUE)
92         xyplot(x, data0,
93             par.settings = par.settings,
94             xscale.components = xscale.components,
95             yscale.components = yscale.components,
96             strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
97         }
98     )
99
100 setMethod('xyplot',
101     signature=c(x='formula', data='Shade'),
102     definition=function(x, data,
103         par.settings=solaR.theme,
104         xscale.components=xscale.solar,
105         yscale.components=yscale.solar,
106         ...){
107         data0=as.data.table(data)
108         xyplot(x, data0,
109             par.settings = par.settings,
110             xscale.components = xscale.components,
111             yscale.components = yscale.components,

```

```

112         strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
113     }
114 )
115
116 setMethod('xyplot',
117     signature=c(x='Meteo', data='missing'),
118     definition=function(x, data,
119         ...){
120         x0=getData(x)
121         xyplot(x0,
122             scales=list(cex=0.6, rot=0, y='free'),
123             strip=FALSE, strip.left=TRUE,
124             par.strip.text=list(cex=0.6),
125             ylab = '',
126             ...)
127     }
128 )
129
130 setMethod('xyplot',
131     signature=c(x='GO', data='missing'),
132     definition=function(x, data, ...){
133         x0 <- as.data.tableD(x, complete=FALSE)
134         x0 <- melt(x0, id.vars = 'Dates')
135         xyplot(value~Dates, x0, groups = variable,
136             par.settings=solaR.theme.2,
137             xscale.components=xscale.solar,
138             yscale.components=yscale.solar,
139             superpose=TRUE,
140             auto.key=list(space='right'),
141             ylab='Wh/m\u00b2',
142             type = 'l',
143             ...)
144     }
145 )
146
147 setMethod('xyplot',
148     signature=c(x='ProdGCPV', data='missing'),
149     definition=function(x, data, ...){
150         x0 <- as.data.tableD(x, complete=FALSE)
151         xyplot(x0,
152             strip = FALSE, strip.left = TRUE,
153             ylab = '', ...)
154     }
155 )
156
157 setMethod('xyplot',
158     signature=c(x='ProdPVPS', data='missing'),
159     definition=function(x, data, ...){
160         x0 <- as.data.tableD(x, complete=FALSE)
161         xyplot(x0,
162             strip = FALSE, strip.left = TRUE,
163             ylab = '', ...)
164     }
165 )

```

EXTRACTO DE CÓDIGO A.56: *xyplot*

## A.5. Conjunto de datos

### A.5.1. aguiar

```
1 data(MTM)
2 Ktlim
```

EXTRACTO DE CÓDIGO A.57: *aguiar*<sub>1</sub>

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 0.031 0.058 0.051 0.052 0.028 0.053 0.044 0.085 0.010 0.319
[2,] 0.705 0.694 0.753 0.753 0.807 0.856 0.818 0.846 0.842 0.865
```

```
1 Ktmtm
```

EXTRACTO DE CÓDIGO A.58: *aguiar*<sub>2</sub>

```
[1] 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 1.00
```

```
1 head(MTM)
```

EXTRACTO DE CÓDIGO A.59: *aguiar*<sub>3</sub>

```
      V1    V2    V3    V4    V5    V6    V7    V8    V9 V10
1 0.229 0.333 0.208 0.042 0.083 0.042 0.042 0.021 0.000 0
2 0.167 0.319 0.194 0.139 0.097 0.028 0.042 0.000 0.014 0
3 0.250 0.250 0.091 0.136 0.091 0.046 0.046 0.023 0.068 0
4 0.158 0.237 0.158 0.263 0.026 0.053 0.079 0.026 0.000 0
5 0.211 0.053 0.211 0.158 0.053 0.053 0.158 0.105 0.000 0
6 0.125 0.125 0.250 0.188 0.063 0.125 0.000 0.125 0.000 0
```

### A.5.2. SIAR

```
1 data(SIAR)
2 head(est_SIAR)
```

EXTRACTO DE CÓDIGO A.60: *SIAR*

	Estacion	Codigo	Longitud	Latitud	Altitud	Fecha_Instalacion	Fecha_Baja
	<char>	<char>	<num>	<num>	<int>	<Date>	<Date>
1:	Villena	A01	-0.884444444	38.67639	519	1999-11-09	2000-03-19
2:	Camp de Mirra	A02	-0.772777778	38.67917	589	1999-11-09	<NA>
3:	Vila Joiosa	A03	-0.256111111	38.52778	73	1999-11-10	<NA>
4:	Ondara	A04	0.006388889	38.81833	38	1999-11-10	<NA>
5:	Dénia Gata	A05	0.082500000	38.79250	86	1999-11-15	<NA>
6:	Pinoso	A06	-1.060555556	38.42722	629	1999-11-14	<NA>

## A.5.3. helios

```
1 data(helios)
2 head(helios)
```

EXTRACTO DE CÓDIGO A.61: *helios*

```

  yyyy.mm.dd      G.O.  TambMax  TambMin
1 2009/01/01  980.14    11.77    6.31
2 2009/01/02 1671.80    15.08    7.27
3 2009/01/03  671.02     9.33    6.36
4 2009/01/04 2482.80    11.71    1.11
5 2009/01/05 1178.19     7.33   -1.54
6 2009/01/06 1722.31     7.77   -0.78
```

## A.5.4. prodEx

```
1 data(prodEx)
2 head(prodEx)
```

EXTRACTO DE CÓDIGO A.62: *prodEx*

```

      Dates      1      2      3      4      5      6      7      8      9
      <Date>    <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: 2007-07-02 8.874982 8.847533 7.173181 8.874982 8.920729 8.975626 8.948177 8.948177 8.948177
2: 2007-07-03 8.710291 8.691992 8.655395 8.710291 8.737740 8.792637 8.774338 8.774338 8.746889
3: 2007-07-04 8.746889 8.737740 8.865832 8.737740 8.765188 8.838384 8.810935 8.792637 8.801786
4: 2007-07-05 8.280266 8.271117 8.408359 8.280266 8.344313 8.380911 8.353462 8.362612 8.316864
5: 2007-07-06 8.399209 8.417508 8.509003 8.435807 8.490704 8.490704 8.499854 8.527302 8.472405
6: 2007-07-07 8.197921 8.170473 8.335163 8.225370 8.243669 8.307715 8.298565 8.280266 8.243669
      10      11      12      13      14      15      16      17      18      19      20
      <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>  <num>
1: 8.984775 8.783487 8.865832 8.966476 8.884131 8.774338 8.829234 8.627946 8.911580 8.807886 6.505270
2: 8.801786 8.545601 8.682843 8.774338 8.691992 8.591348 8.646245 8.426658 8.710291 8.563900 3.952569
3: 8.829234 8.545601 8.618797 8.829234 8.719441 8.618797 8.664544 8.426658 8.728590 8.612697 6.331430
4: 8.380911 8.179622 8.271117 8.353462 8.280266 8.207071 8.261968 8.188772 7.950886 8.222320 5.498829
5: 8.509003 8.316864 8.426658 8.490704 8.435807 8.344313 8.408359 8.371761 8.463256 8.332113 6.551017
6: 8.326014 8.152174 8.161323 8.316864 8.234519 8.143024 8.179622 8.170473 8.243669 8.161323 6.669960
      21      22
      <num>  <num>
1: 3.742131 3.980018
2: 4.080662 3.238911
3: 1.363270 1.043039
4: 3.998316 2.461206
5: 5.361587 4.959010
6: 5.215195 4.922413
```

## A.5.5. pumpCoef

```
1 data(pumpCoef)
2 head(pumpCoef)
```

EXTRACTO DE CÓDIGO A.63: *pumpCoef*

```

      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:      2      6  2.6   370 0.01409736 0.018576 -3.6324 -0.32  0.74  0.22 -0.1614 0.5247 0.0694
2:      2      9  2.6   370 0.02114604 0.027864 -5.4486 -0.32  0.74  0.22 -0.1614 0.5247 0.0694
3:      2     13  2.6   550 0.03054428 0.040248 -7.8702 -0.12  0.49  0.27 -0.1614 0.5247 0.0694
```

4:	2	18	2.6	750	0.04229208	0.055728	-10.8972	-0.16	0.42	0.47	-0.1614	0.5247	0.0694
5:	2	23	2.6	1100	0.05403988	0.071208	-13.9242	-0.20	0.51	0.42	-0.1614	0.5247	0.0694
6:	2	28	2.6	1500	0.06578768	0.086688	-16.9512	-0.24	0.50	0.49	-0.1614	0.5247	0.0694



# 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.
- [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>.
- [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.
- [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.

- [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>.
- [Uni20] European Union. *NextGenerationEU*. 2020. URL: [https://next-generation-eu.europa.eu/index\\_es](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/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-datatable.com>.
- [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.
- [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>.