



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, 26 de agosto 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. Este paquete ofrece herramientas avanzadas para investigaciones reproducibles en el campo de la energía solar fotovoltaica, permitiendo tanto la simulación del rendimiento de sistemas conectados a la red como de sistemas de bombeo de agua alimentados por energía solar. **solaR2** incluye una serie de clases, métodos y funciones que abarcan desde el cálculo de la geometría solar y la radiación solar incidente en un generador fotovoltaico hasta la estimación precisa de la productividad final de estos sistemas, desde la irradiación global horizontal diaria e intradía.

El diseño modular y basado en clases **S4** facilita el manejo de series temporales multivariantes y ofrece métodos de visualización avanzados para el análisis del rendimiento en plantas fotovoltaicas a gran escala. Una característica distintiva de **solaR2** es su implementación apoyada en el paquete **data.table**, que optimiza la manipulación de grandes volúmenes de datos, permitiendo un procesamiento más rápido y eficiente de las series temporales. Esto es fundamental para un análisis detallado y continuo de los datos solares.

Entre sus funcionalidades más destacadas se encuentran el cálculo de la radiación solar en diferentes planos, la estimación del rendimiento de sistemas fotovoltaicos conectados a la red y de sistemas de bombeo, así como la evaluación y optimización de sombras en los sistemas. Además, el paquete incluye herramientas avanzadas para la visualización estadística del rendimiento, permitiendo analizar tanto series temporales como realizar análisis espaciales en combinación con otros paquetes de R. **solaR2** es particularmente útil para investigadores y profesionales involucrados en el diseño, evaluación y optimización de sistemas fotovoltaicos, proporcionando un análisis detallado de su rendimiento bajo diversas condiciones de irradiación y temperatura, lo que es esencial para maximizar la eficiencia energética y la rentabilidad de las instalaciones solares.

Además, el paquete es compatible con otras bibliotecas de R para la manipulación de series temporales y la visualización de datos, lo que garantiza la precisión en los cálculos temporales y la integración con datos geoespaciales. En resumen, la creación de **solaR2** representa una contribución significativa al campo de la energía fotovoltaica, proporcionando una herramienta flexible, reproducible y de fácil uso para el análisis y simulación de sistemas solares. Este TFG no solo detalla el desarrollo técnico del paquete, sino que también presenta aplicaciones prácticas y estudios de caso que demuestran su utilidad en escenarios reales, subrayando su capacidad para mejorar la productividad y eficiencia de los sistemas fotovoltaicos mediante un análisis exhaustivo de la radiación solar y las condiciones ambientales.

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, named **solaR2**, designed to estimate the productivity of photovoltaic systems based on solar irradiation data. This package offers advanced tools for reproducible research in the field of photovoltaic solar energy, allowing both the simulation of the performance of grid-connected systems and water pumping systems powered by solar energy. **solaR2** includes a series of classes, methods, and functions that cover everything from the calculation of solar geometry and the solar radiation incident on a photovoltaic generator to the precise estimation of the final productivity of these systems, from daily and intraday global horizontal irradiation.

The modular and class-based **S4** design facilitates the handling of multivariate time series and offers advanced visualization methods for performance analysis in large-scale photovoltaic plants. A distinctive feature of **solaR2** is its implementation supported by the **data.table** package, which optimizes the handling of large volumes of data, allowing faster and more efficient processing of time series. This is essential for detailed and continuous analysis of solar data.

Among its most notable functionalities are the calculation of solar radiation on different planes, the estimation of the performance of grid-connected photovoltaic systems and pumping systems, as well as the evaluation and optimization of shading in the systems. Additionally, the package includes advanced tools for statistical performance visualization, allowing the analysis of both time series and spatial analysis in combination with other R packages. **solaR2** is particularly useful for researchers and professionals involved in the design, evaluation, and optimization of photovoltaic systems, providing a detailed analysis of their performance under various irradiation and temperature conditions, which is essential to maximize energy efficiency and the profitability of solar installations.

Furthermore, the package is compatible with other **R** libraries for time series manipulation and data visualization, ensuring accuracy in temporal calculations and integration with geospatial data. In summary, the creation of **solaR2** represents a significant contribution to the field of photovoltaic energy, providing a flexible, reproducible, and easy-to-use tool for the analysis and simulation of solar systems. This final degree project not only details the technical development of the package but also presents practical applications and case studies that demonstrate its usefulness in real scenarios, highlighting its ability to improve the productivity and efficiency of photovoltaic systems through comprehensive analysis of solar radiation and environmental conditions.

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

Índice general

Índice general	IX
Índice de figuras	XI
Nomenclatura	XIII
1 Introducción	1
1.1. Objetivos	1
1.2. Análisis previo de soluciones	3
1.3. Aspectos técnicos	3
2 Estado del arte	7
2.1. Situación actual de la generación fotovoltaica	7
2.2. Solución actual y sus carencias	9
3 Marco teórico	11
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
4 Desarrollo del código	25
4.1. Geometría solar	25
4.2. Datos meteorológicos	32
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	59
4.6. Producción eléctrica de un SFB	62
4.7. Optimización de distancias	66
4.8. Métodos de visualización	70
5 Ejemplo práctico de aplicación	77
5.1. <code>solaR</code>	77
5.2. <code>PVsyst</code>	77
5.3. <code>solaR</code>	77
5.4. Comparación entre los tres	77
A Código completo	79
A.1. Constructores	79
A.2. Clases	102
A.3. Funciones	105
A.4. Métodos	134
A.5. Conjunto de datos	157

Bibliografía

161

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 solaR2	26
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.	26
4.3. Los datos meteorologicas 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	32
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 . :	37
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, añadiendole el efecto de las sombras producidas entres módulos.	46
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 (η_{inv}) del inversor.	60
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.	62

Nomenclatura

A_c	Área de una célula
AM	Masa de aire
AO	Adelanto oficial durante el horario de verano
B_0	Irradiancia extra-atmosférica o extra-terrestre
B	Radiación directa
β	Ángulo de inclinación de un sistema fotovoltaico
D	Radiación difusa
D^C	Radiación difusa circunsolar
δ	Declinación
$\Delta\lambda$	Diferencia entre la longitud local y la longitud del huso horario
D^I	Radiación difusa isotrópica
EoT	Ecuación del tiempo
ϵ_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	Radiación global
K_T	Índice de claridad
MPP	Punto de máxima potencia de un dispositivo fotovoltaico
ω	Hora solar o tiempo solar verdadero
ω_s	Ángulo del amanecer
ϕ	Latitud
R	Radiación del albedo

r_D	Relación entre la irradiancia y la irradiación difusa en el plano horizontal
ρ	Coeficiente 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
θ_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 posible generación de una instalación fotovoltaica.

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 podrá 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 eficaz 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 2012. Por la antigüedad del código 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 alojados 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^AT_EX (partiendo de un archivo .org, se puede exportar a un archivo .tex para posteriormente exportar un pdf).
- Obtener conocimientos teóricos acerca de la radiación solar y de la producción de energía solar mediante sistemas fotovoltaicos y sus diversos tipos. Para ello se ha usado en mayor medida el libro “Energía Solar Fotovoltaica” [Per23].

1.2. Análisis previo de soluciones

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 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

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

Herramienta web diseñada por el **Grupo de Sistemas Fotovoltaicos del Instituto de Energía Solar de la Universidad Politécnica de Madrid**.

3. PVGIS

Aplicación web desarrollada por el **European Commission Joint Research Center** desde 2001.

4. System Advisor Model

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

A pesar de las diversas crisis internacionales, la energía solar fotovoltaica 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 REPowerEU² [Eur22]. Dentro de este plan, se lanzó la Estrategia de Energía Solar con el objetivo de alcanzar 400 GWp (320 GW) para 2030, incluyendo medidas para desarrollar tejados solares, impulsar la industria fotovoltaica y apoyar la formación de profesionales en el sector.

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

¹UNEF: Unión Española Fotovoltaica.

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

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

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

Se adjudicaron 1.200 MW del nudo de transición justa de Andorra a Enel Green Power España, con planes para instalar plantas de hidrógeno verde y 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 PERTE⁴, aprobado en diciembre de 2021, espera crear más de 280.000 empleos, con ayudas que se ejecutarán hasta 2026. En 2023 se solicitó a Bruselas una adenda para segunda fase del PERTE, obteniendo 2.700 millones de euros adicionales.

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

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

UNEF promueve la transformación industrial para que España se convierta en un hub industrial fotovoltaico. Se destaca la necesidad de proteger la industria existente, garantizar un crecimiento constante de la capacidad y ofrecer condiciones de financiamiento favorables. Además se propone implementar una Estrategia Industrial Fotovoltaica para contribuir significativamente a la reindustrialización de la economía, aprovechando las medidas del REPower Plan, la Estrategia Solar y la Alianza de la Industria Solar Fotovoltaica.

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.

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

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

⁵PIB: Producto Interior Bruto.

2.2. Solución actual y sus carencias

Como se mencionó en el capítulo 1 este proyecto toma su base en el paquete **solarR** [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 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, **solarR** presenta una serie de carencias relativas al paquete **zoo**:

- **Eficiencia y rendimiento:** el paquete **solarR** 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:** **solarR** 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.
- **Interoperabilidad:** **solarR** está algo limitado en términos de integración con otras tecnologías de datos modernas debido a su dependencia en **zoo**. En cambio, **data.table** es ampliamente compatible y se integra de manera más fluida con otros paquetes y herramientas en el ecosistema de **R**, facilitando la interoperabilidad y la construcción de pipelines de datos más complejos.
- **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 proporcionla en el uso de recursos, haciendo que las operaciones sean más sostenibles en términos de memoria.

Por lo tanto, al adoptar **data.table** en **solarR2**, 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 **solaR2** toma como marco teórico el libro de Oscar Perpiñán, tutor de este trabajo, Energía Solar Fotovoltaica [Per23] para cada una de las operaciones de cálculo que realizan cada una de las funciones. En la figura 3.1, se muestra un diagrama que resume los pasos que se siguen a la hora de calcular la producción de sistemas fotovoltaicos. Estos pasos son:

1. Obtener la irradiación global diaria en el plano horizontal
2. A partir de la irradiación global, obtener las componentes de difusa y directa.
3. Se trasladan estos valores de irradiación a valores de irradiancia.
4. Con estos valores se pueden obtener los valores correspondientes en el plano del generador
 - a) Sin los efectos de la suciedad de los módulos y las sombras que se generan unos con otros.
 - b) Con estos efectos
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: porción de radiación que procede en línea recta desde el Sol.

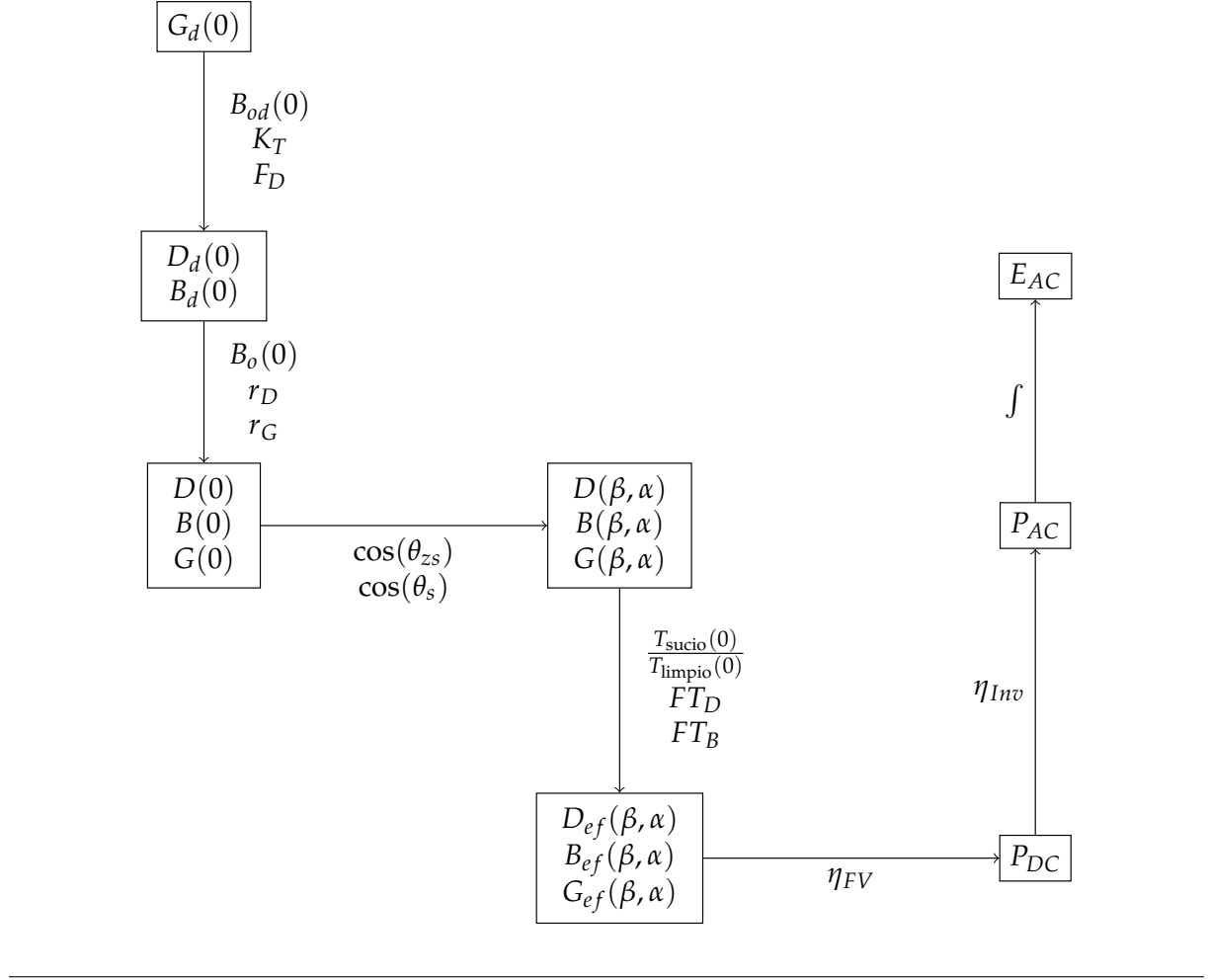


FIGURA 3.1: Procedimiento de cálculo

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

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

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

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 facot 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: $\cos(\theta_{zs}) = \cos(\delta) \cos(\omega) \cos(\phi) + \sin(\delta) \sin(\phi)$ ² {Ángulo cenital solar}

Donde:

- Declinación: $\delta = 23,45^\circ \cdot \sin(\frac{2\pi \cdot (d_n + 284)}{365})$
- Latitud: ϕ
- 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: $EoT = 229,18 \cdot (-0,0334 \cdot \sin(\frac{2\pi}{365,24} \cdot d_n) + 0,04184 \cdot \sin(2 \cdot \frac{2\pi}{365,24} \cdot d_n + 3,5884))$

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)

¹Para 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.

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

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

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 geometríasolar (ángulo zenital) y del generador (ángulo de incidencia).

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

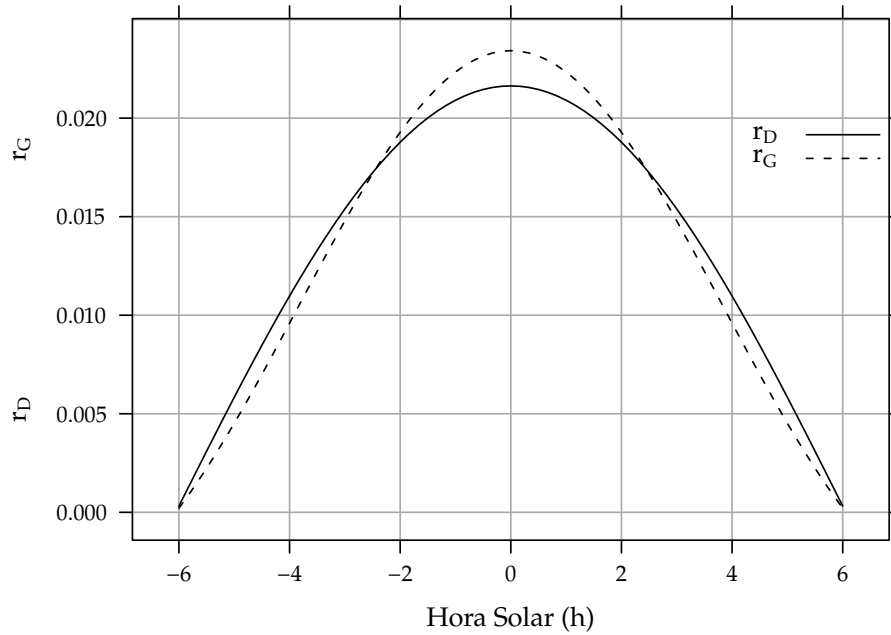


FIGURA 3.2: Perfil de irradiancia difusa y global obtenido a partir del generador empírico de [CR79] para valores de irradiancia tomadas cada 10 minutos

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

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

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

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

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, ρ , 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 θ_s . La suciedad acumulada en la superficie del módulo también reduce la transmitancia del vidrio (representada por $T_{limpio}(0)$), disminuyendo la irradiancia efectiva, es decir, la radiación que realmente puede ser aprovechada por el módulo. La irradiancia efectiva para radiación directa se expresa en la ecuación 3.17:

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

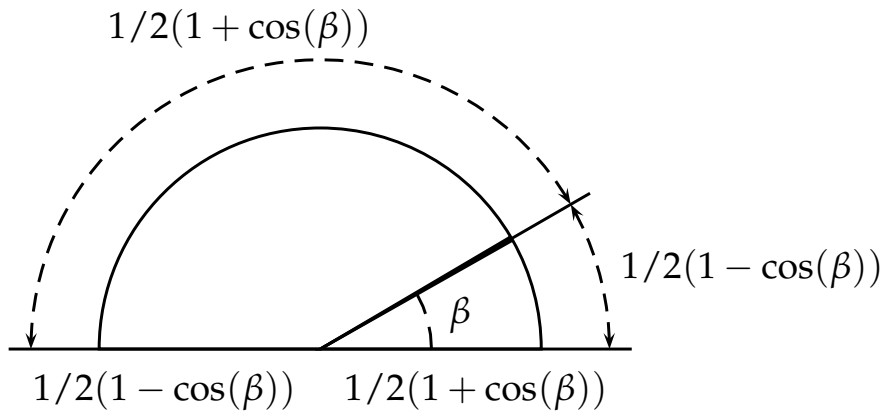


FIGURA 3.3: Ángulo de visión del cielo

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

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

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

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

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

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

Donde:

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

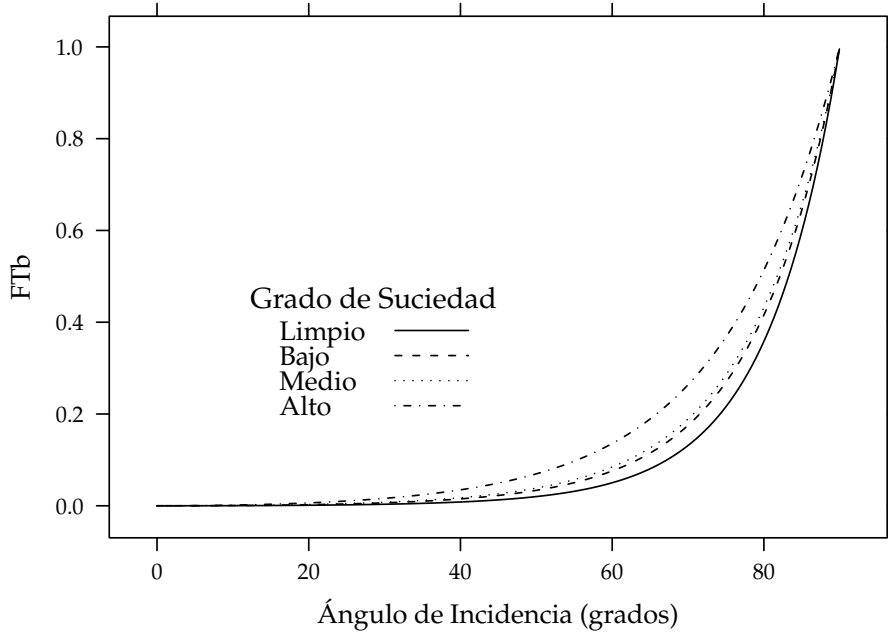


FIGURA 3.4: Pérdidas angulares de un módulo fotovoltaico para diferentes grados de suciedad en función del ángulo de incidencia.

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

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):

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

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

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

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).

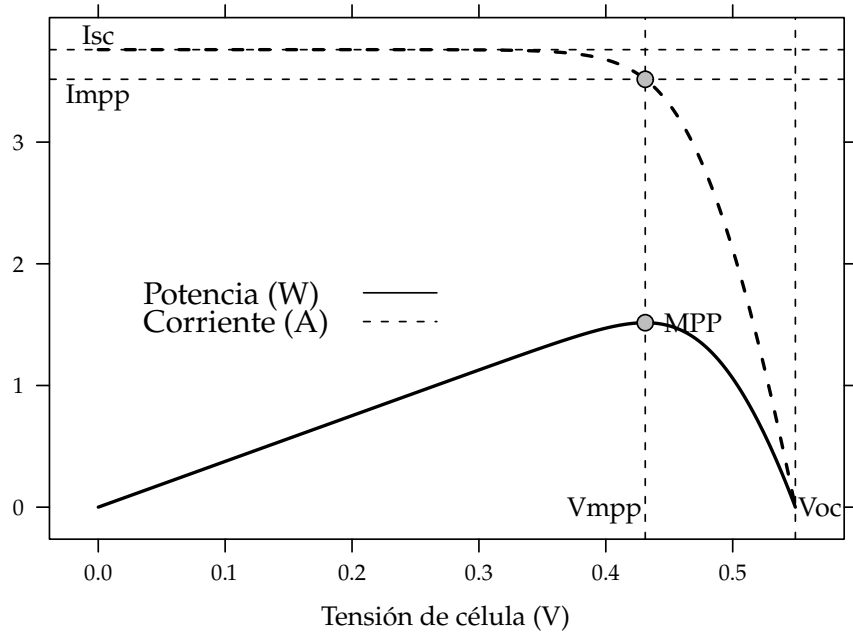


FIGURA 3.5: Curvas corriente-tensión(línea discontinua) y potencia-tensión(línea continua) de una célula solar ($T_a = 20^\circ C$ y $G = 800 W/m^2$)

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:

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

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 $-2,3 \frac{mV}{^\circ C}$. Además, disminuye la eficiencia de la célula solar con $\frac{d\eta}{dT_c} = -0,4 \%/^\circ C$.

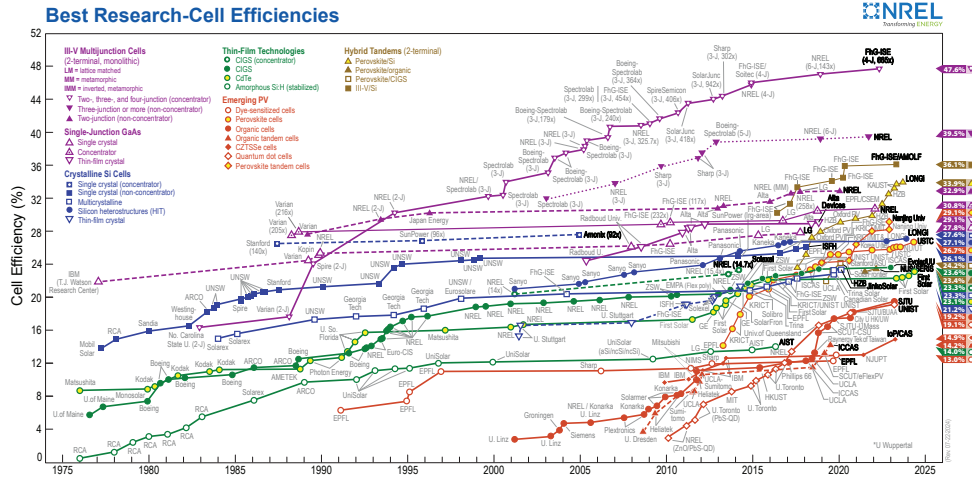


FIGURA 3.6: Evolución de la eficiencia de células según la tecnología (según el National Renewable Energy Laboratory [Nat24] (EEUU)).

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:

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

Frecuentemente los fabricantes informan de los valores de las tensiones V_{oc}^* y V_{mpp}^* y las corrientes I_{sc}^* y I_{mpp}^* .⁴ A partir de estos valores es posible referir a estas condiciones:

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

3.3.2. Funcionamiento de un módulo fotovoltaico

Comportamiento térmico de un módulo

La mayoría de las ecuaciones que definen el comportamiento de un módulo 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

³Relación 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 ($AM = 1/\cos\theta_{zs}$).

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

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.

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

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 $T_a = 25^\circ C$ 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:

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

- **Cálculo de r_s :** Utilizando el valor 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 la 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

La potencia obtenida en el paso anterior es la de un solo módulo. Para conocer la potencia que va a ser capaz de entregar el generador, 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:

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

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 (ϕ), computa la geometría a nivel diario, es decir, los ángulos y componentes que se pueden calcular en cada día independiente.

estas son:

- Declinación (δ): calculada a partir de la función **declination**¹.
- Excentricidad (ϵ_o): obtenida mediante la función **eccentricity**.
- Ecuación del tiempo (EoT): obtenida mediante la función **eot**.
- Ángulo del amanecer (ω_s): calculada a partir de la función **sunrise**.
- Irradiancia diaria extra-atmosférica ($B_{0d}(0)$): obtenida a partir de la función **bo0d**.

```
lat <- 37.2
BTd <- fBTd(mode = 'prom')
solD <- fSolD(lat = lat, BTd = BTd)
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
```

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

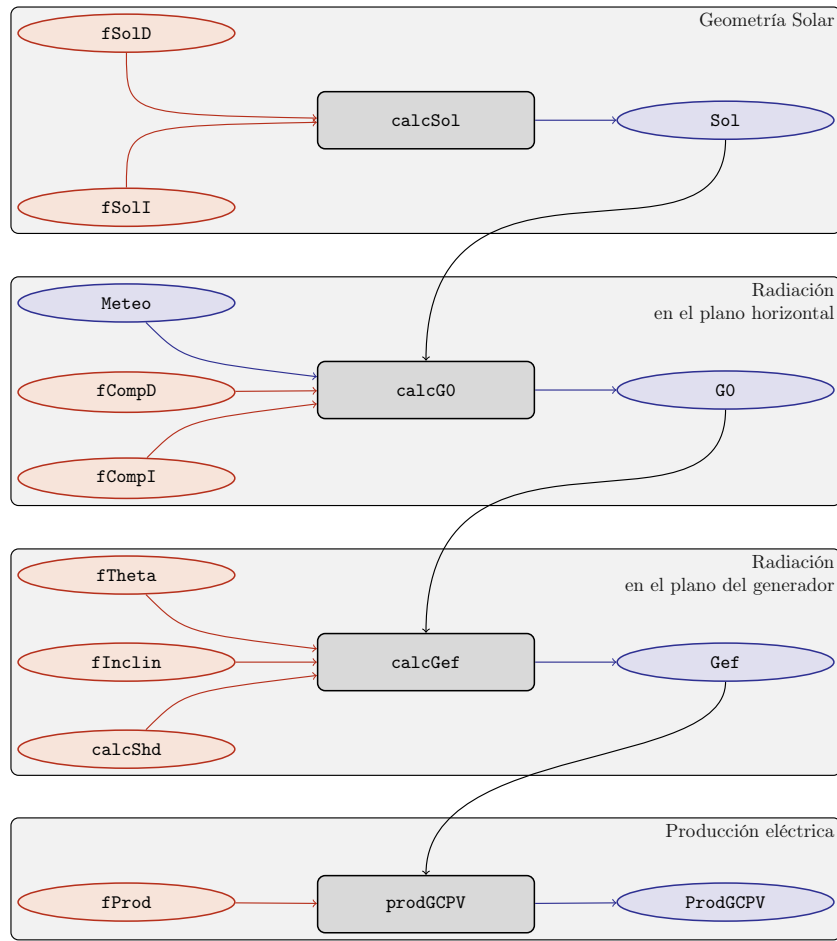


FIGURA 4.1: Proceso de cálculo de las funciones de *solar2*

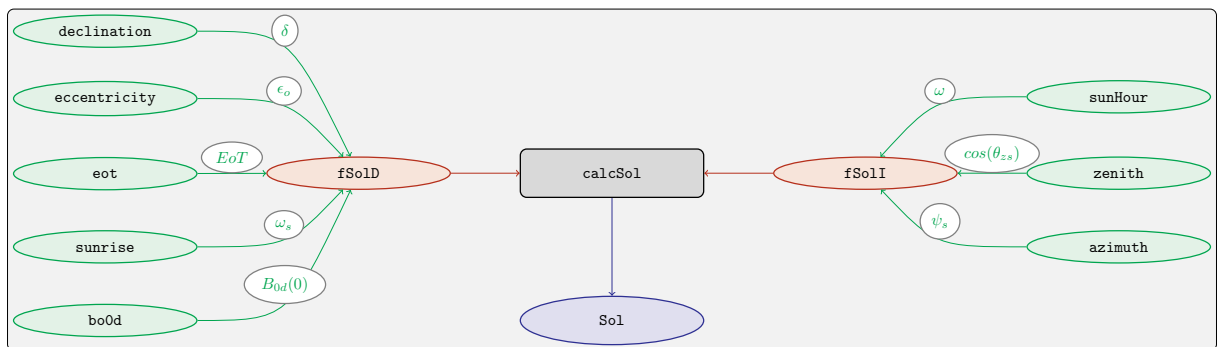


FIGURA 4.2: Cálculo de la geometría solar mediante la función `calcSol`, la cual unifica las funciones `fSolD` y `fSolI` resultando en un objeto clase `Sol` el cual contiene toda la información geométrica necesaria para realizar las siguientes estimaciones.

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

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**):

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

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

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

```
Key: <Dates>
      Dates    lat    decl    eo    EoT    ws    Bo0d
      <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 (ω): calculada a partir de la función **sunHour**.
 - Los momentos del día en los que es de noche (*night*): calculada a partir del resultado anterior y de el ángulo del amanecer (calculada en **fSold**)².
 - El coseno del ángulo cenital solar ($\cos(\theta_{zs})$): obtenida a partir de la función **zenith**.
 - La altura solar (γ_s): obtenida a partir del resultado anterior³.
 - El ángulo zenital solar (θ_{zs}): calculada mediante la función **azimuth**.
 - La irradiancia extra-atmosférica ($B_0(0)$): calculada mediante el coseno del ángulo cenital, la constante solar (B_0) y la excentricidad (calculada en **fSold**) [ecuación 3.2].

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

```
Index: <night>
      Dates    lat    w    night    cosThzS    AlS
      <POSc> <num>    <num>    <lgcl>    <num>    <num>
1: 2024-01-17 00:00:00 37.2  3.09905026  TRUE -0.958552332 -1.281876984
2: 2024-01-17 01:00:00 37.2 -2.92239722  TRUE -0.941407376 -1.226779122
3: 2024-01-17 02:00:00 37.2 -2.66065932  TRUE -0.874749489 -1.064918604
4: 2024-01-17 03:00:00 37.2 -2.39892132  TRUE -0.763119126 -0.868125900
5: 2024-01-17 04:00:00 37.2 -2.13718324  TRUE -0.614120126 -0.661270606
6: 2024-01-17 05:00:00 37.2 -1.87544507  TRUE -0.437901763 -0.453263434
7: 2024-01-17 06:00:00 37.2 -1.61370681  TRUE -0.246467423 -0.249033534
8: 2024-01-17 07:00:00 37.2 -1.35196846  TRUE -0.052856976 -0.052881619
```

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

³ $\gamma_s = \sin(\cos(\theta_s))$.

```

9: 2024-01-17 08:00:00 37.2 -1.09023003 FALSE 0.129741461 0.130108233
10: 2024-01-17 09:00:00 37.2 -0.82849151 FALSE 0.288889848 0.293067041
11: 2024-01-17 10:00:00 37.2 -0.56675290 FALSE 0.413747472 0.426566560
12: 2024-01-17 11:00:00 37.2 -0.30501420 FALSE 0.495809380 0.518766586
13: 2024-01-17 12:00:00 37.2 -0.04327541 FALSE 0.529485721 0.557994217
14: 2024-01-17 13:00:00 37.2 0.21846346 FALSE 0.512482515 0.538073327
15: 2024-01-17 14:00:00 37.2 0.48020243 FALSE 0.445957919 0.462244212
16: 2024-01-17 15:00:00 37.2 0.74194148 FALSE 0.334443348 0.341014503
17: 2024-01-17 16:00:00 37.2 1.00368062 FALSE 0.185534810 0.186616094
18: 2024-01-17 17:00:00 37.2 1.26541985 FALSE 0.009375501 0.009375638
19: 2024-01-17 18:00:00 37.2 1.52715917 TRUE -0.182035120 -0.183055757
20: 2024-01-17 19:00:00 37.2 1.78889857 TRUE -0.375658695 -0.385107424
21: 2024-01-17 20:00:00 37.2 2.05063807 TRUE -0.558306105 -0.592342658
22: 2024-01-17 21:00:00 37.2 2.31237766 TRUE -0.717535874 -0.800258081
23: 2024-01-17 22:00:00 37.2 2.57411733 TRUE -0.842501657 -1.001910427
24: 2024-01-17 23:00:00 37.2 2.83585709 TRUE -0.924691065 -1.180223341

```

```

      Dates lat      w night      cosThzS      ALS
      AzS    Bo0
      <num>    <num>
1: 3.00157749 0.00000
2: -2.49462689 0.00000
3: -2.03862388 0.00000
4: -1.77932134 0.00000
5: -1.59701536 0.00000
6: -1.44469585 0.00000
7: -1.30093496 0.00000
8: -1.15283370 0.00000
9: -0.99014548 183.39419
10: -0.80329847 408.35612
11: -0.58400587 584.84684
12: -0.32921922 700.84427
13: -0.04769723 748.44699
14: 0.23821864 724.41235
15: 0.50355560 630.37745
16: 0.73469016 472.74762
17: 0.93148844 262.26008
18: 1.10112996 13.25261
19: 1.25297092 0.00000
20: 1.39694027 0.00000
21: 1.54466726 0.00000
22: 1.71368519 0.00000
23: 1.93928567 0.00000
24: 2.30977400 0.00000
      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**.

```

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

```

```

      Dates lat      w night      cosThzS      ALS
      <POSc> <num>    <num> <lgcl>    <num>    <num>
1: 2024-01-17 08:00:00 37.2 -1.09023003 FALSE 0.129741461 0.130108233
2: 2024-01-17 09:00:00 37.2 -0.82849151 FALSE 0.288889848 0.293067041
3: 2024-01-17 10:00:00 37.2 -0.56675290 FALSE 0.413747472 0.426566560
4: 2024-01-17 11:00:00 37.2 -0.30501420 FALSE 0.495809380 0.518766586
5: 2024-01-17 12:00:00 37.2 -0.04327541 FALSE 0.529485721 0.557994217
6: 2024-01-17 13:00:00 37.2 0.21846346 FALSE 0.512482515 0.538073327

```

4. DESARROLLO DEL CÓDIGO

```
7: 2024-01-17 14:00:00 37.2 0.48020243 FALSE 0.445957919 0.462244212
8: 2024-01-17 15:00:00 37.2 0.74194148 FALSE 0.334443348 0.341014503
9: 2024-01-17 16:00:00 37.2 1.00368062 FALSE 0.185534810 0.186616094
10: 2024-01-17 17:00:00 37.2 1.26541985 FALSE 0.009375501 0.009375638
      AzS      Bo0
      <num>      <num>
1: -0.99014548 183.39419
2: -0.80329847 408.35612
3: -0.58400587 584.84684
4: -0.32921922 700.84427
5: -0.04769723 748.44699
6: 0.23821864 724.41235
7: 0.50355560 630.37745
8: 0.73469016 472.74762
9: 0.93148844 262.26008
10: 1.10112996 13.25261
```

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

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

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

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

```
solI_EoT <- fSolI(sold = sold, BTi = BTi, EoT = FALSE)
show(solI_EoT)
```

```

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

Bo0
<num>
1:    0
2:    0
3:    0
4:    0
5:    0
---
284:    0
285:    0
286:    0
287:    0
288:    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.

```

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

```

```

Object of class Sol

Latitude:  37.2 degrees

Daily values:
      Dates      decl      eo      EoT
Min.   :2024-01-17  Min.   :-0.404783  Min.   :0.9675  Min.   :-0.0614793
1st Qu.:2024-04-07  1st Qu.: -0.256032  1st Qu.:0.9771  1st Qu.: -0.0289759
Median :2024-06-29  Median : -0.002305  Median :1.0007  Median : 0.0005052
Mean   :2024-07-01  Mean   : -0.001618  Mean   :1.0009  Mean   : 0.0008748
3rd Qu.:2024-09-25  3rd Qu.: 0.251172  3rd Qu.:1.0249  3rd Qu.: 0.0204515
Max.   :2024-12-13  Max.   : 0.402578  Max.   :1.0340  Max.   : 0.0689613

      ws      Bo0d
Min.   :-1.900  Min.   : 4284
1st Qu.: -1.767  1st Qu.: 5841
Median : -1.569  Median : 8297
Mean   : -1.569  Mean   : 8109
3rd Qu.: -1.370  3rd Qu.:10416
Max.   : -1.240  Max.   :11574

Intradaily values:
      Dates      w      night
Min.   :2024-01-17 00:00:00  Min.   : -3.1393050  Mode :logical

```

```

1st Qu.:2024-04-07 11:45:00 1st Qu.: -1.5692285 FALSE:145
Median :2024-06-29 11:30:00 Median : 0.0010871 TRUE :143
Mean :2024-07-01 15:30:00 Mean : 0.0009975
3rd Qu.:2024-09-26 11:15:00 3rd Qu.: 1.5716412
Max. :2024-12-13 23:00:00 Max. : 3.1413972

cosThzS ALS AzS BoO
Min. : -0.9700256 Min. : -1.325336 Min. : -3.139169 Min. : 0.000
1st Qu.: -0.5004531 1st Qu.: -0.524130 1st Qu.: -1.570722 1st Qu.: 0.000
Median : 0.0062923 Median : 0.006292 Median : 0.003834 Median : 8.748
Mean : -0.0009523 Mean : -0.001202 Mean : 0.001011 Mean : 337.752
3rd Qu.: 0.5007129 3rd Qu.: 0.524433 3rd Qu.: 1.555342 3rd Qu.: 698.153
Max. : 0.9697262 Max. : 1.324107 Max. : 3.141331 Max. : 1284.718

```

4.2. Datos meteorológicos

Para el procesamiento de datos meteorológicos, **solar2** provee una serie de funciones⁴ 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.

```

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

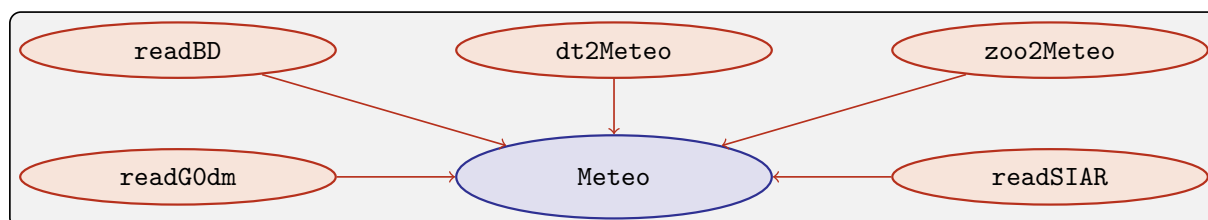


FIGURA 4.3: Los datos meteorológicos se pueden leer mediante las funciones **readG0dm**, **readBD**, **dt2Meteo**, **zoo2Meteo** y **readSIAR** las cuales procesan estos datos y los almacenan en un objeto de clase **Meteo**.

⁴Las funciones comentadas en este apartado, se recogen en la sección A.1.8

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.

```
## Se utiliza un archivo alojado en el
## github del tutor de este proyecto
myURL <- "https://raw.githubusercontent.com/oscarperpinan/R/master/data/
  aranjuez.csv"
download.file(myURL, 'data/aranjuez.csv', quiet = TRUE)
BDd <- readBDd(file = 'data/aranjuez.csv', lat = lat,
  format = '%Y-%m-%d', header = TRUE,
  fill = TRUE, dec = '.', sep = ',', dates.col = '',
  ta.col = 'TempAvg', g0.col = 'Radiation', keep.cols = TRUE)
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
Min. :2004-01-01	Min. : 0.277	Min. : -5.309	Min. : -12.980
1st Qu.:2005-12-29	1st Qu.: 9.370	1st Qu.: 7.692	1st Qu.: 1.515
Median :2008-01-09	Median :16.660	Median :13.810	Median : 7.170
Mean :2008-01-03	Mean :16.742	Mean :14.405	Mean : 6.888
3rd Qu.:2010-01-02	3rd Qu.:24.650	3rd Qu.:21.615	3rd Qu.:12.590
Max. :2011-12-31	Max. :32.740	Max. :30.680	Max. :22.710
	NA's :13		NA's :4
TempMax	HumidAvg	HumidMax	WindAvg
Min. : -2.362	Min. : 19.89	Min. : 35.88	Min. : 0.251
1st Qu.:14.530	1st Qu.: 47.04	1st Qu.: 81.60	1st Qu.:0.667
Median :21.670	Median : 62.58	Median : 90.90	Median :0.920
Mean :22.531	Mean : 62.16	Mean : 87.22	Mean :1.174
3rd Qu.:30.875	3rd Qu.: 77.38	3rd Qu.: 94.90	3rd Qu.:1.431
Max. :41.910	Max. :100.00	Max. :100.00	Max. :8.260
		NA's :13	NA's :8
WindMax	Rain	ET	
Min. : 0.000	Min. : 0.000	Min. :0.000	
1st Qu.: 3.783	1st Qu.: 0.000	1st Qu.:1.168	
Median : 5.027	Median : 0.000	Median :2.758	
Mean : 5.208	Mean : 1.094	Mean :3.091	
3rd Qu.: 6.537	3rd Qu.: 0.200	3rd Qu.:4.926	
Max. :10.000	Max. :49.730	Max. :8.564	
NA's :128	NA's :4	NA's :18	

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

```

myURL <- "https://raw.githubusercontent.com/oscarperpinan/R/master/data/
NREL-Hawaii.csv"
download.file(myURL, 'data/NREL-Hawaii.csv', quiet = TRUE)
BDi <- readBDi(file = 'data/NREL-Hawaii.csv', lat = 19,
              format = "%d/%m/%Y %H:%M", header = TRUE,
              fill = TRUE, dec = '.', sep = ',',
              dates.col = 'DATE', times.col = 'HST',
              ta.col = 'Air Temperature [deg C]',
              g0.col = 'Global Horizontal [W/m^2]',
              keep.cols = TRUE)
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
Min. :2010-01-11 06:32:00.00	Min. : 0.4769	Min. :13.42
1st Qu.:2010-03-11 17:37:45.00	1st Qu.: 147.4328	1st Qu.:22.76
Median :2010-06-11 17:32:30.00	Median : 300.6510	Median :24.15
Mean :2010-06-26 11:55:22.63	Mean : 370.5293	Mean :23.64
3rd Qu.:2010-09-11 17:34:15.00	3rd Qu.: 585.7402	3rd Qu.:25.24
Max. :2010-12-11 17:46:00.00	Max. :1172.3000	Max. :28.12
NA's :4660		
Direct Normal [W/m^2]	Diffuse Horizontal [W/m^2]	
Min. : 0.0	Min. : 0.4769	
1st Qu.: 0.0	1st Qu.: 78.4636	
Median :270.3	Median :152.9320	
Mean :356.6	Mean :171.7706	
3rd Qu.:715.2	3rd Qu.:246.3193	
Max. :943.0	Max. :586.3600	

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

```

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

```

Object of class `Meteo`

Source of meteorological information: `bd-data.frame`

Latitude of source: 40 degrees

Meteorological Data:

Dates	G0d	TempMin
Min. :2009-01-01 00:00:00.00	Min. : 325.6	Min. : -37.500
1st Qu.:2009-04-08 12:00:00.00	1st Qu.: 2523.2	1st Qu.: 1.950
Median :2009-07-07 00:00:00.00	Median : 4745.7	Median : 7.910
Mean :2009-07-04 21:29:54.93	Mean : 4812.0	Mean : 5.323
3rd Qu.:2009-10-03 12:00:00.00	3rd Qu.: 7139.5	3rd Qu.: 15.105
Max. :2009-12-31 00:00:00.00	Max. :11253.9	Max. : 24.800
TempMax	Ta	
Min. : 1.41	Min. : -23.049	


```
1st Qu.:14.41  1st Qu.: 7.008
Median :23.16  Median : 12.055
Mean :22.59   Mean : 10.944
3rd Qu.:31.06  3rd Qu.: 19.472
Max. :38.04   Max. : 28.619
```

- **zoo2Meteo**: Transforma un objeto de clase **zoo**⁵ en un objeto de clase **Meteo**.

```
library(zoo)
bd_zoo <- read.csv.zoo('data/aranjuez.csv')
BD_zoo <- zoo2Meteo(file = bd_zoo, lat = 40)
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
Min. : -5.309	Min. : -2.362	Min. : -12.980	Min. : 19.89
1st Qu.: 7.692	1st Qu.: 14.530	1st Qu.: 1.515	1st Qu.: 47.04
Median : 13.810	Median : 21.670	Median : 7.170	Median : 62.58
Mean : 14.405	Mean : 22.531	Mean : 6.888	Mean : 62.16
3rd Qu.: 21.615	3rd Qu.: 30.875	3rd Qu.: 12.590	3rd Qu.: 77.38
Max. : 30.680	Max. : 41.910	Max. : 22.710	Max. : 100.00
		NA's : 4	

HumidMax	WindAvg	WindMax	Rain
Min. : 35.88	Min. : 0.251	Min. : 0.000	Min. : 0.000
1st Qu.: 81.60	1st Qu.: 0.667	1st Qu.: 3.783	1st Qu.: 0.000
Median : 90.90	Median : 0.920	Median : 5.027	Median : 0.000
Mean : 87.22	Mean : 1.174	Mean : 5.208	Mean : 1.094
3rd Qu.: 94.90	3rd Qu.: 1.431	3rd Qu.: 6.537	3rd Qu.: 0.200
Max. : 100.00	Max. : 8.260	Max. : 10.000	Max. : 49.730
NA's : 13	NA's : 8	NA's : 128	NA's : 4

Radiation	ET
Min. : 0.277	Min. : 0.000
1st Qu.: 9.370	1st Qu.: 1.168
Median : 16.660	Median : 2.758
Mean : 16.742	Mean : 3.091
3rd Qu.: 24.650	3rd Qu.: 4.926
Max. : 32.740	Max. : 8.564
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**.

⁵Pese 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.

```
library(httr2)
library(jsonlite)
bd_SIAR <- readSIAR(Lat = 40.40596822621351, Lon = -3.70038308516172,
  ## Ubicación de la Escuela Técnica Superior
  ## de Ingeniería y Diseño Industrial (ETSIDI)
  inicio = '2023-09-01', final = '2024-08-01',
  tipo = 'Mensuales', n_est = 3)
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
Min. :2023-09-18 00:00:00	Min. :1860	Min. : 5.318	Min. :-4.6513
1st Qu.:2023-12-06 18:00:00	1st Qu.:2744	1st Qu.: 9.857	1st Qu.: -2.1466
Median :2024-02-29 00:00:00	Median :4052	Median :14.890	Median : 0.3663
Mean :2024-03-01 04:00:00	Mean :4529	Mean :15.348	Mean : 2.4225
3rd Qu.:2024-05-21 12:00:00	3rd Qu.:6616	3rd Qu.:20.047	3rd Qu.: 7.1506
Max. :2024-08-18 00:00:00	Max. :7608	Max. :27.560	Max. :12.6082

TempMax
Min. :15.34
1st Qu.:21.12
Median :31.01
Mean :29.41
3rd Qu.:35.47
Max. :40.70

Esta función tiene dos argumentos importantes:

- **tipo:** La API SIAR⁶ permite tener 4 tipos de registros: **Mensuales**, **Semanales**, **Diarios** y **Horarios**.
- **n_est:** Con este argumento, la función es capaz de localizar el número seleccionado de estaciones más 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, `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:

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

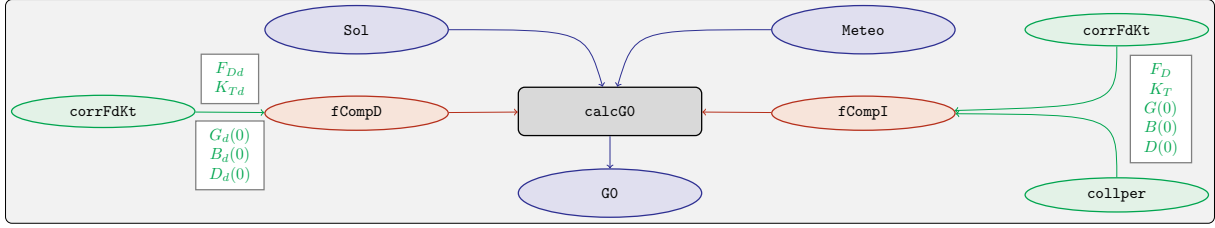


FIGURA 4.4: Cálculo de la radiación incidente en el plano horizontal mediante la función **calcGO**, la cual procesa un objeto clase **Sol** y otro clase **Meteo** mediante las funciones **fCompD** y **fCompI** resultando en un objeto clase **GO**.

- **fCompD**: La cual computa 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:

```

lat <- 37.2
BTd <- fBTd(mode = 'prom')
solD <- fSolD(lat, BTd)
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
compD_page <- fCompD(sol = solD, G0d = G0d, corr = "Page")
compD_page

```

Key: <Dates>

Dates	Fd	Kt	G0d	D0d	B0d
<POS<	<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

```

compD_lj <- fCompD(sol = solD, G0d = G0d, corr = "LJ")
compD_lj

```

Key: <Dates>

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

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

```
Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
      <POSc>    <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
```

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

```
Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
      <POSc>    <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

```

```

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

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
   <P0Sc>   <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 corrección propia.

```

f_corrd <- function(sol, G0d){
  ## Función CLIMEDd
  Kt <- Ktd(sol, G0d)
  Fd=(Kt<=0.13)*(0.952)+
    (Kt>0.13 & Kt<=0.8)*(0.868+1.335*Kt-5.782*Kt^2+3.721*Kt^3)+
    (Kt>0.8)*0.141
  return(data.table(Fd, Kt))
}
compD_user <- fCompD(sol = sol, G0d = G0d, corr = 'user', f = f_corrd)
compD_user

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
   <P0Sc>   <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 corrección.

```

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

```

```

Key: <Dates>
      Dates      Fd      Kt      G0d      D0d      B0d
   <POSc>   <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**, computa 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).

```

sol <- calcSol(lat = 37.2, BTd = fBTd(mode = 'prom'),
              sample = 'hour', keep.night = FALSE)
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
compD <- fCompD(sol = sol, G0d = G0d, corr = 'CPR')
compI <- fCompI(sol = sol, compD = compD)
show(compI)

```

```

Key: <Dates>
      Dates      Fd      Kt      G0      D0      B0
   <POSc>   <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**.

```

G0I <- compI$G0
compI_ekdh <- fCompI(sol = sol, G0I = G0I, corr = 'EKDh')
show(compI_ekdh)

```

```

Key: <Dates>
      Dates      Fd      Kt      G0      D0      B0
    <P0Sc>    <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

```

```

compI_br1 <- fCompI(sol = sol, G0I = G0I, corr = 'BRL')
show(compI_br1)

```

```

Key: <Dates>
      Dates      Fd      Kt      G0      D0      B0
    <P0Sc>    <num>    <num>    <num>    <num>    <num>
1: 2024-01-17 08:00:00 0.6573908 0.4583592 84.06042 55.26054 28.79987
2: 2024-01-17 09:00:00 0.5624767 0.5277148 215.49558 121.21125 94.28433
3: 2024-01-17 10:00:00 0.4845081 0.5821268 340.45500 164.95322 175.50179
4: 2024-01-17 11:00:00 0.4333714 0.6178887 433.04376 187.66880 245.37496
5: 2024-01-17 12:00:00 0.4120068 0.6325646 473.44106 195.06094 278.38012
---
141: 2024-12-13 12:00:00 0.5776181 0.5488870 382.71443 221.06278 161.65164
142: 2024-12-13 13:00:00 0.5917966 0.5371376 352.10710 208.37580 143.73130
143: 2024-12-13 14:00:00 0.6306611 0.5063725 276.60890 174.44649 102.16241
144: 2024-12-13 15:00:00 0.6887448 0.4586869 172.87432 119.06629 53.80803
145: 2024-12-13 16:00:00 0.7561974 0.3973283 63.15968 47.76119 15.39849

```

```

compI_climedh <- fCompI(sol = sol, G0I = G0I, corr = 'CLIMEDh')
show(compI_climedh)

```

Key: <Dates>

	Dates <POSc>	Fd <num>	Kt <num>	GO <num>	DO <num>	B0 <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.

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

Key: <Dates>

	Dates <POSc>	Fd <num>	Kt <num>	GO <num>	DO <num>	B0 <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 corrección.

```
GOI <- compI_user
compI_none <- fCompI(sol = sol, GOI = GOI, corr = 'none')
show(compI_none)
```

Key: <Dates>

	Dates <POSc>	Fd <num>	Kt <num>	GO <num>	DO <num>	B0 <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. A parte 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.

```

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

```

```

Object of class  GO

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.

```
g0_aguiar <- calcG0(lat, modeRad = 'aguiar', dataRad = prom)
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 1.096343 1.669657
2: Feb. 2024 3.491 1.588619 1.902381
3: Mar. 2024 4.494 2.007561 2.486439
4: Apr. 2024 5.912 2.354686 3.557314
5: May. 2024 6.989 2.499471 4.489529
6: Jun. 2024 7.742 2.463914 5.278086
7: Jul. 2024 7.919 2.279394 5.639606
8: Aug. 2024 7.027 2.226269 4.800731
9: Sep. 2024 5.369 1.789533 3.579467
10: Oct. 2024 3.562 1.585801 1.976199
11: Nov. 2024 2.814 1.173628 1.640372
12: Dec. 2024 2.179 1.076159 1.102841

Yearly values:
Key: <Dates>
      Dates   G0d      D0d      B0d
      <int>   <num>   <num>   <num>
1:  2024 1839.365 675.4237 1163.941
```

- Diarios.

```
bd <- g0_aguiar@G0D
g0_bd <- calcG0(lat, modeRad = 'bd', dataRad = bd)
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 1.096343 1.669657
2: Feb. 2024 3.491 1.588619 1.902381
3: Mar. 2024 4.494 2.007561 2.486439
4: Apr. 2024 5.912 2.354686 3.557314
5: May. 2024 6.989 2.499471 4.489529
6: Jun. 2024 7.742 2.463914 5.278086
7: Jul. 2024 7.919 2.279394 5.639606
8: Aug. 2024 7.027 2.226269 4.800731
9: Sep. 2024 5.369 1.789533 3.579467
10: Oct. 2024 3.562 1.585801 1.976199
```

```
11: Nov. 2024 2.814 1.173628 1.640372
12: Dec. 2024 2.179 1.076159 1.102841
```

Yearly values:

Key: <Dates>

Dates	G0d	D0d	B0d
<int>	<num>	<num>	<num>
1: 2024	1839.365	675.4237	1163.941

■ Intradiarios

```
bdI <- g0_aguiar@G0I
g0_bdI <- calcG0(lat, modeRad = 'bdI', dataRad = bdI)
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	2.766	1.081850	1.684150
2: Feb. 2024	3.491	1.640091	1.850909
3: Mar. 2024	4.494	2.158400	2.335600
4: Apr. 2024	5.912	2.573242	3.338758
5: May. 2024	6.989	2.509373	4.479627
6: Jun. 2024	7.742	2.423260	5.318740
7: Jul. 2024	7.919	1.912492	6.006508
8: Aug. 2024	7.027	1.884232	5.142768
9: Sep. 2024	5.369	1.616619	3.752381
10: Oct. 2024	3.562	1.704965	1.857035
11: Nov. 2024	2.814	1.161031	1.652969
12: Dec. 2024	2.179	1.103065	1.075935

Yearly values:

Key: <Dates>

Dates	G0d	D0d	B0d
<int>	<num>	<num>	<num>
1: 2024	1839.365	663.7729	1175.592

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 (β) y la orientación (α), computa el ángulo de inclinación en cada instante (β), el ángulo azimutal (ψ_s) y el coseno del ángulo

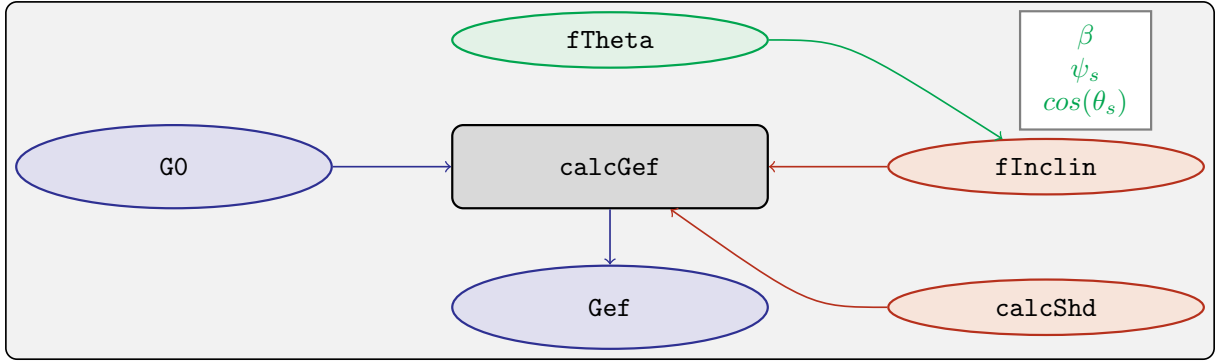


FIGURA 4.5: Cálculo de la radiación efectiva incidente en el plano del generador mediante la función **calcGef**, la cual emplea la función **fInclin** para el computo de las componentes efectivas, la función **fTheta** que provee a la función anterior los ángulos necesarios para su computo y la función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

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.

```

BTd <- fBTd(mode = 'prom')[6]
sol <- calcSol(lat, BTd = BTd, keep.night = FALSE)
beta <- lat - 10
alfa <- 0
angGen_fixed <- fTheta(sol = sol, beta = beta, alfa = alfa,
                      modeTrk = 'fixed')
show(angGen_fixed)

```

	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.

```

angGen_two <- fTheta(sol = sol, beta = beta, alfa = alfa,
                    modeTrk = 'two')
show(angGen_two)

```

	Dates <POS>	Beta <num>	Alfa <num>	cosTheta <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.

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

	Dates <POS>	Beta <num>	Alfa <num>	cosTheta <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.

```
struct <- list(L = 1)
distances <- data.table(Lew = 2)
angGen_BT <- fTheta(sol = sol, beta = beta, alfa = alfa,
                  modeTrk = 'horiz', BT = TRUE,
                  struct = struct, dist = distances)
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 computa la irradiancia efectiva. Por defecto tiene valor 2 (grado de suciedad bajo).

```

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

```

	Dates	Bo	Bn	G	D	Di	Dc
	<POS>	<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
2:	2024-01-17 09:00:00	792.6980	464.2106	366.6704	103.96230	52.12242	51.83988
3:	2024-01-17 10:00:00	1010.9063	541.3602	536.6247	145.69981	68.60264	77.09717
4:	2024-01-17 11:00:00	1154.3223	592.0663	662.0048	173.72247	77.44190	96.28057
5:	2024-01-17 12:00:00	1213.1770	612.8750	716.5974	185.35767	80.61172	104.74595

141:	2024-12-13 12:00:00	1181.1554	470.2512	578.4583	180.82966	95.85462	84.97504
142:	2024-12-13 13:00:00	1129.5610	453.5904	536.8668	170.08970	91.70559	78.38411
143:	2024-12-13 14:00:00	994.4636	409.9651	434.0673	142.25355	79.88147	62.37208
144:	2024-12-13 15:00:00	785.0640	342.3463	292.1950	99.92831	58.81096	41.11735
145:	2024-12-13 16:00:00	515.6229	255.3390	140.8937	46.94651	26.80445	20.14206

	B	R	FTb	FTd	FTr	Dief	Dcef
	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	133.18653	0.9295706	0.115032290	0.05043622	0.2503398	24.51843	23.47122
2:	260.32510	2.3830282	0.034235799	0.05043622	0.2503398	49.49356	50.06510
3:	387.15997	3.7648749	0.012139104	0.05043622	0.2503398	65.14258	76.16128
4:	483.49354	4.7887550	0.005426675	0.05043622	0.2503398	73.53602	95.75809
5:	526.00427	5.2354830	0.003640433	0.05043622	0.2503398	76.54597	104.36463

141:	393.39650	4.2321949	0.004516349	0.05043622	0.2503398	91.02007	84.59127
142:	362.88341	3.8937280	0.006269898	0.05043622	0.2503398	87.08031	77.89265
143:	288.75488	3.0588416	0.013120704	0.05043622	0.2503398	75.85255	61.55372
144:	190.35496	1.9117069	0.035287438	0.05043622	0.2503398	55.84476	39.66642
145:	93.24874	0.6984426	0.114223038	0.05043622	0.2503398	25.45254	17.84137
	Gef	Def	Bef	Ref			
	<num>	<num>	<num>	<num>			

```

1: 166.5523 47.98966 117.86578 0.6968621
2: 352.7578 99.55866 251.41266 1.7864615
3: 526.5864 141.30386 382.46020 2.8223770
4: 653.7538 169.29411 480.86978 3.5899392
5: 708.9248 180.91060 524.08939 3.9248333
---
141: 570.4038 175.61134 391.61978 3.1727082
142: 528.5001 164.97296 360.60816 2.9189730
143: 424.6656 137.40626 284.96622 2.2930919
144: 280.5821 95.51118 183.63782 1.4331306
145: 126.4151 43.29391 82.59758 0.5235947

```

```

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

```

```

          Dates          Bo          Bn          G          D          Di          Dc
        <POSc>        <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
2: 2024-01-17 09:00:00 792.6980 464.2106 366.6704 103.96230 52.12242 51.83988
3: 2024-01-17 10:00:00 1010.9063 541.3602 536.6247 145.69981 68.60264 77.09717
4: 2024-01-17 11:00:00 1154.3223 592.0663 662.0048 173.72247 77.44190 96.28057
5: 2024-01-17 12:00:00 1213.1770 612.8750 716.5974 185.35767 80.61172 104.74595
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 578.4583 180.82966 95.85462 84.97504
142: 2024-12-13 13:00:00 1129.5610 453.5904 536.8668 170.08970 91.70559 78.38411
143: 2024-12-13 14:00:00 994.4636 409.9651 434.0673 142.25355 79.88147 62.37208
144: 2024-12-13 15:00:00 785.0640 342.3463 292.1950 99.92831 58.81096 41.11735
145: 2024-12-13 16:00:00 515.6229 255.3390 140.8937 46.94651 26.80445 20.14206

          B          R          FTb          FTd          FTr          Dief          Dcef
        <num>        <num>        <num>        <num>        <num>        <num>        <num>
1: 133.18653 0.9295706 0.24100175 0.09714708 0.3918962 21.44734 18.51982
2: 260.32510 2.3830282 0.10321543 0.09714708 0.3918962 43.29416 42.77007
3: 387.15997 3.7648749 0.04727214 0.09714708 0.3918962 56.98305 67.57641
4: 483.49354 4.7887550 0.02455379 0.09714708 0.3918962 64.32515 86.40320
5: 526.00427 5.2354830 0.01743586 0.09714708 0.3918962 66.95809 94.68605
---
141: 393.39650 4.2321949 0.02100686 0.09714708 0.3918962 79.61921 76.53478
142: 362.88341 3.8937280 0.02771140 0.09714708 0.3918962 76.17293 70.11502
143: 288.75488 3.0588416 0.05023795 0.09714708 0.3918962 66.35152 54.49955
144: 190.35496 1.9117069 0.10550059 0.09714708 0.3918962 48.84983 33.83709
145: 93.24874 0.6984426 0.23984890 0.09714708 0.3918962 22.26444 14.08613

          Gef          Def          Bef          Ref
        <num>        <num>        <num>        <num>
1: 133.4885 39.96716 93.00127 0.5200533
2: 302.1765 86.06424 214.77909 1.3331982
3: 466.0152 124.55946 339.34944 2.1062799
4: 587.2996 150.72835 433.89218 2.6790952
5: 640.0594 161.64413 475.48630 2.9290196
---
141: 512.8436 156.15400 354.32187 2.3677246
142: 473.0675 146.28795 324.60121 2.1783674
143: 374.8709 120.85106 252.30856 1.7112857
144: 240.4070 82.68692 156.65061 1.0695149
145: 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).

4. DESARROLLO DEL CÓDIGO

```
inclin_alb0 <- fInclin(compI = compI, angGen = angGen, alb = 0)
show(inclin_alb0)
```

```

      Dates      Bo      Bn      G      D      Di      Dc
      <POS<      <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
2: 2024-01-17 09:00:00 792.6980 464.2106 364.2874 103.96230 52.12242 51.83988
3: 2024-01-17 10:00:00 1010.9063 541.3602 532.8598 145.69981 68.60264 77.09717
4: 2024-01-17 11:00:00 1154.3223 592.0663 657.2160 173.72247 77.44190 96.28057
5: 2024-01-17 12:00:00 1213.1770 612.8750 711.3619 185.35767 80.61172 104.74595
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 574.2262 180.82966 95.85462 84.97504
142: 2024-12-13 13:00:00 1129.5610 453.5904 532.9731 170.08970 91.70559 78.38411
143: 2024-12-13 14:00:00 994.4636 409.9651 431.0084 142.25355 79.88147 62.37208
144: 2024-12-13 15:00:00 785.0640 342.3463 290.2833 99.92831 58.81096 41.11735
145: 2024-12-13 16:00:00 515.6229 255.3390 140.1953 46.94651 26.80445 20.14206

      B      R      FTb      FTd      FTr      Dief      Dcef      Gef
      <num> <num>    <num>    <num>    <num>    <num>    <num>    <num>
1: 133.18653 0 0.156321477 0.06473603 0.2994808 23.66622 21.92862 155.7141
2: 260.32510 0 0.054197292 0.06473603 0.2994808 47.77325 48.04970 337.1148
3: 387.15997 0 0.021399057 0.06473603 0.2994808 62.87835 73.93841 508.1144
4: 483.49354 0 0.010185772 0.06473603 0.2994808 70.98005 93.39388 633.3713
5: 526.00427 0 0.006996517 0.06473603 0.2994808 73.88537 101.93283 687.6958
---
141: 393.39650 0 0.008575046 0.06473603 0.2994808 87.85638 82.56145 552.6405
142: 362.88341 0 0.011653979 0.06473603 0.2994808 84.05356 75.92121 511.4560
143: 288.75488 0 0.022965930 0.06473603 0.2994808 73.21605 59.72086 409.4178
144: 190.35496 0 0.055666181 0.06473603 0.2994808 53.90370 38.05193 268.1191
145: 93.24874 0 0.155368802 0.06473603 0.2994808 24.56786 16.67236 118.4258

      Def      Bef      Ref
      <num>    <num>    <num>
1: 45.59484 110.11928 0
2: 95.82295 241.29186 0
3: 136.81676 371.29761 0
4: 164.37393 468.99741 0
5: 175.81821 511.87759 0
---
141: 170.41783 382.22264 0
142: 159.97477 351.48128 0
143: 132.93691 276.48089 0
144: 91.95563 176.16345 0
145: 41.24021 77.18558 0

```

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

```

      Dates      Bo      Bn      G      D      Di      Dc
      <POS<      <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
2: 2024-01-17 09:00:00 792.6980 464.2106 376.2025 103.96230 52.12242 51.83988
3: 2024-01-17 10:00:00 1010.9063 541.3602 551.6842 145.69981 68.60264 77.09717
4: 2024-01-17 11:00:00 1154.3223 592.0663 681.1598 173.72247 77.44190 96.28057
5: 2024-01-17 12:00:00 1213.1770 612.8750 737.5394 185.35767 80.61172 104.74595
---
141: 2024-12-13 12:00:00 1181.1554 470.2512 595.3871 180.82966 95.85462 84.97504
142: 2024-12-13 13:00:00 1129.5610 453.5904 552.4417 170.08970 91.70559 78.38411
143: 2024-12-13 14:00:00 994.4636 409.9651 446.3026 142.25355 79.88147 62.37208
144: 2024-12-13 15:00:00 785.0640 342.3463 299.8418 99.92831 58.81096 41.11735
145: 2024-12-13 16:00:00 515.6229 255.3390 143.6875 46.94651 26.80445 20.14206

```


	B	R	FTb	FTd	FTr	Dief	Dcef
	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	133.18653	4.647853	0.156321477	0.06473603	0.2994808	23.66622	21.92862
2:	260.32510	11.915141	0.054197292	0.06473603	0.2994808	47.77325	48.04970
3:	387.15997	18.824375	0.021399057	0.06473603	0.2994808	62.87835	73.93841
4:	483.49354	23.943775	0.010185772	0.06473603	0.2994808	70.98005	93.39388
5:	526.00427	26.177415	0.006996517	0.06473603	0.2994808	73.88537	101.93283

141:	393.39650	21.160975	0.008575046	0.06473603	0.2994808	87.85638	82.56145
142:	362.88341	19.468640	0.011653979	0.06473603	0.2994808	84.05356	75.92121
143:	288.75488	15.294208	0.022965930	0.06473603	0.2994808	73.21605	59.72086
144:	190.35496	9.558535	0.055666181	0.06473603	0.2994808	53.90370	38.05193
145:	93.24874	3.492213	0.155368802	0.06473603	0.2994808	24.56786	16.67236
	Gef	Def	Bef	Ref			
	<num>	<num>	<num>	<num>			
1:	158.9049	45.59484	110.11928	3.190792			
2:	345.2947	95.82295	241.29186	8.179849			
3:	521.0375	136.81676	371.29761	12.923098			
4:	649.8089	164.37393	468.99741	16.437612			
5:	705.6668	175.81821	511.87759	17.971025			

141:	567.1677	170.41783	382.22264	14.527195			
142:	524.8214	159.97477	351.48128	13.365392			
143:	419.9174	132.93691	276.48089	10.499608			
144:	274.6811	91.95563	176.16345	6.562018			
145:	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**⁷ (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.

```
inclin_horizBright <- fInclin(compI = compI, angGen = angGen,
                             horizBright = FALSE)
show(inclin_horizBright)
```

	Dates	Bo	Bn	G	D	Di	Dc
	<POS>	<num>	<num>	<num>	<num>	<num>	<num>
1:	2024-01-17 08:00:00	514.5612	365.8727	186.2091	52.09303	25.57090	26.52212
2:	2024-01-17 09:00:00	792.6980	464.2106	366.1413	103.43314	51.59325	51.83988
3:	2024-01-17 10:00:00	1010.9063	541.3602	535.9087	144.98390	67.88673	77.09717
4:	2024-01-17 11:00:00	1154.3223	592.0663	661.1846	172.90227	76.62170	96.28057
5:	2024-01-17 12:00:00	1213.1770	612.8750	715.7390	184.49921	79.75326	104.74595

141:	2024-12-13 12:00:00	1181.1554	470.2512	577.4973	179.86860	94.89356	84.97504
142:	2024-12-13 13:00:00	1129.5610	453.5904	535.9539	169.17679	90.79268	78.38411
143:	2024-12-13 14:00:00	994.4636	409.9651	433.2885	141.47476	79.10268	62.37208
144:	2024-12-13 15:00:00	785.0640	342.3463	291.6442	99.37758	58.26023	41.11735
145:	2024-12-13 16:00:00	515.6229	255.3390	140.6606	46.71344	26.57138	20.14206
	B	R	FTb	FTd	FTr	Dief	Dcef
	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	133.18653	0.9295706	0.156321477	0.06473603	0.2994808	23.43723	21.92862
2:	260.32510	2.3830282	0.054197292	0.06473603	0.2994808	47.28824	48.04970
3:	387.15997	3.7648749	0.021399057	0.06473603	0.2994808	62.22217	73.93841
4:	483.49354	4.7887550	0.010185772	0.06473603	0.2994808	70.22829	93.39388

⁷la 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

```

5: 526.00427 5.2354830 0.006996517 0.06473603 0.2994808 73.09855 101.93283
---
141: 393.39650 4.2321949 0.008575046 0.06473603 0.2994808 86.97552 82.56145
142: 362.88341 3.8937280 0.011653979 0.06473603 0.2994808 83.21682 75.92121
143: 288.75488 3.0588416 0.022965930 0.06473603 0.2994808 72.50225 59.72086
144: 190.35496 1.9117069 0.055666181 0.06473603 0.2994808 53.39892 38.05193
145: 93.24874 0.6984426 0.155368802 0.06473603 0.2994808 24.35423 16.67236
      Gef      Def      Bef      Ref
      <num>    <num>    <num>    <num>
1: 156.1233  45.36586 110.11928 0.6381583
2: 338.2658  95.33794 241.29186 1.6359698
3: 510.0428 136.16059 371.29761 2.5846197
4: 635.9071 163.62217 468.99741 3.2875223
5: 690.5032 175.03138 511.87759 3.5942050
---
141: 554.6650 169.53697 382.22264 2.9054390
142: 513.2924 159.13803 351.48128 2.6730784
143: 410.8039 132.22311 276.48089 2.0999216
144: 268.9267 91.45086 176.16345 1.3124036
145: 118.6917 41.02659 77.18558 0.4794870

```

```

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

```

```

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

```

```

143:      0 276.48089      0
144:      0 176.16345      0
145:      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**.

```

gef_prom <- calcGef(lat = lat, modeTrk = 'two', modeRad = 'prom',
                   dataRad = prom,
                   beta = lat-10, alfa = 0,
                   iS = 2, alb = 0.2,
                   horizBright = TRUE, HCPV = FALSE)

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

Befd
<num>

1: 4.825736
2: 4.933601
3: 5.060439
4: 6.280444
5: 7.465149
6: 8.920381
7: 9.397421
8: 8.468140
9: 6.564518
10: 4.455104
11: 4.545523
12: 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.

```
struct <- list(W=23.11, L=9.8, Nrow=2, Ncol=8)
distances <- data.table(Lew=40, Lns=30, H=0)
gef_shd <- calcShd(radEf = gef_prom, modeShd = 'prom',
                  struct = struct, distances = distances)
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
	<char>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1: Jan. 2024	6.348801	1.384087	4.825736	6.522313	1.440413	4.924221	6.104126	
2: Feb. 2024	6.680139	1.599929	4.933601	6.875052	1.672079	5.034287	6.406274	
3: Mar. 2024	7.104641	1.902356	5.060439	7.329138	1.998110	5.163713	6.788630	
4: Apr. 2024	8.578222	2.158071	6.280444	8.843422	2.265896	6.408617	8.295340	
5: May. 2024	9.885240	2.284334	7.465149	10.178196	2.394606	7.617499	9.688308	
6: Jun. 2024	11.293417	2.230338	8.920381	11.606533	2.329653	9.102430	11.115054	
7: Jul. 2024	11.495648	1.948530	9.397421	11.801533	2.029150	9.589205	11.308971	
8: Aug. 2024	10.493150	1.869393	8.468140	10.777404	1.947410	8.640959	10.196758	
9: Sep. 2024	8.584604	1.864962	6.564518	8.831006	1.948075	6.698488	8.228309	
10: Oct. 2024	6.226290	1.631551	4.455104	6.418653	1.711039	4.546024	6.018374	
11: Nov. 2024	6.076159	1.393353	4.545523	6.247341	1.452953	4.638289	5.875732	
12: Dec. 2024	4.685547	1.198824	3.370992	4.825181	1.254616	3.439788	4.575893	
	Defd	Befd						
	<num>	<num>						
1:	1.343455	4.621693						
2:	1.553670	4.705996						
3:	1.848127	4.798657						
4:	2.112064	6.043569						
5:	2.253942	7.298609						
6:	2.205314	8.767042						
7:	1.924962	9.234312						
8:	1.830334	8.210807						
9:	1.810198	6.262986						
10:	1.595528	4.283212						
11:	1.359514	4.378935						
12:	1.179346	3.280817						

Yearly values:

	Dates	Gef0d	Def0d	Bef0d	Gd	Dd	Bd	Gefd	Defd
	<int>	<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
		Befd							
		<num>							
1:		2193.621							

```
-----
Mode of tracking: two
Inclination limit: 90
```

```
gef_shd2 <- calcGef(lat = lat, modeTrk = 'two', dataRad = prom,
                    modeShd = 'prom', struct = struct, distances = distances)
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
      <char>      <num>      <num>      <num>      <num>      <num>      <num>      <num>
1: Jan. 2024  6.348801  1.384087  4.825736  6.522313  1.440413  4.924221  6.104126
2: Feb. 2024  6.680139  1.599929  4.933601  6.875052  1.672079  5.034287  6.406274
3: Mar. 2024  7.104641  1.902356  5.060439  7.329138  1.998110  5.163713  6.788630
4: Apr. 2024  8.578222  2.158071  6.280444  8.843422  2.265896  6.408617  8.295340
5: May. 2024  9.885240  2.284334  7.465149  10.178196  2.394606  7.617499  9.688308
6: Jun. 2024  11.293417  2.230338  8.920381  11.606533  2.329653  9.102430  11.115054
7: Jul. 2024  11.495648  1.948530  9.397421  11.801533  2.029150  9.589205  11.308971
8: Aug. 2024  10.493150  1.869393  8.468140  10.777404  1.947410  8.640959  10.196758
9: Sep. 2024  8.584604  1.864962  6.564518  8.831006  1.948075  6.698488  8.228309
10: Oct. 2024  6.226290  1.631551  4.455104  6.418653  1.711039  4.546024  6.018374
11: Nov. 2024  6.076159  1.393353  4.545523  6.247341  1.452953  4.638289  5.875732
12: Dec. 2024  4.685547  1.198824  3.370992  4.825181  1.254616  3.439788  4.575893
      Defd      Befd
      <num>      <num>
1: 1.343455  4.621693
2: 1.553670  4.705996
3: 1.848127  4.798657
4: 2.112064  6.043569
5: 2.253942  7.298609
6: 2.205314  8.767042
7: 1.924962  9.234312
8: 1.830334  8.210807
9: 1.810198  6.262986
10: 1.595528  4.283212
11: 1.359514  4.378935
12: 1.179346  3.280817

Yearly values:
      Dates      Gef0d      Def0d      Bef0d      Gd      Dd      Bd      Gefd      Defd
      <int>      <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
      Befd
      <num>
1: 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 circumsolar y directa.

```
gef_shdarea <- calcGef(lat, modeTrk = 'two', dataRad = prom,
                      modeShd = 'area',
                      struct = struct, distances = distances)
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
	<char>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	Jan. 2024	6.348801	1.384087	4.825736	6.522313	1.440413	4.924221	5.877879
2:	Feb. 2024	6.680139	1.599929	4.933601	6.875052	1.672079	5.034287	6.291348
3:	Mar. 2024	7.104641	1.902356	5.060439	7.329138	1.998110	5.163713	6.743478
4:	Apr. 2024	8.578222	2.158071	6.280444	8.843422	2.265896	6.408617	8.254928
5:	May. 2024	9.885240	2.284334	7.465149	10.178196	2.394606	7.617499	9.660175
6:	Jun. 2024	11.293417	2.230338	8.920381	11.606533	2.329653	9.102430	11.089573
7:	Jul. 2024	11.495648	1.948530	9.397421	11.801533	2.029150	9.589205	11.282303
8:	Aug. 2024	10.493150	1.869393	8.468140	10.777404	1.947410	8.640959	10.154416
9:	Sep. 2024	8.584604	1.864962	6.564518	8.831006	1.948075	6.698488	8.177410
10:	Oct. 2024	6.226290	1.631551	4.455104	6.418653	1.711039	4.546024	5.950189
11:	Nov. 2024	6.076159	1.393353	4.545523	6.247341	1.452953	4.638289	5.705306
12:	Dec. 2024	4.685547	1.198824	3.370992	4.825181	1.254616	3.439788	4.440179

	Defd	Befd
	<num>	<num>
1:	1.305883	4.433019
2:	1.534257	4.610483
3:	1.840379	4.761253
4:	2.105491	6.009730
5:	2.249601	7.274817
6:	2.201739	8.745137
7:	1.921596	9.211011
8:	1.824754	8.174045
9:	1.802375	6.219910
10:	1.583714	4.226840
11:	1.330740	4.237284
12:	1.155239	3.169210

Yearly values:

	Dates	Gef0d	Def0d	Bef0d	Gd	Dd	Bd	Gefd	Defd
	<int>	<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

	Befd
	<num>
1:	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.

```
gef_shdprom <- calcGef(lat, modeTrk = 'two', dataRad = prom,
                      modeShd = c('area', 'prom'),
                      struct = struct, distances = distances)
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
	<char>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	Jan. 2024	6.348801	1.384087	4.825736	6.522313	1.440413	4.924221	6.104126
2:	Feb. 2024	6.680139	1.599929	4.933601	6.875052	1.672079	5.034287	6.406274
3:	Mar. 2024	7.104641	1.902356	5.060439	7.329138	1.998110	5.163713	6.788630
4:	Apr. 2024	8.578222	2.158071	6.280444	8.843422	2.265896	6.408617	8.295340
5:	May. 2024	9.885240	2.284334	7.465149	10.178196	2.394606	7.617499	9.688308
6:	Jun. 2024	11.293417	2.230338	8.920381	11.606533	2.329653	9.102430	11.115054
7:	Jul. 2024	11.495648	1.948530	9.397421	11.801533	2.029150	9.589205	11.308971
8:	Aug. 2024	10.493150	1.869393	8.468140	10.777404	1.947410	8.640959	10.196758
9:	Sep. 2024	8.584604	1.864962	6.564518	8.831006	1.948075	6.698488	8.228309
10:	Oct. 2024	6.226290	1.631551	4.455104	6.418653	1.711039	4.546024	6.018374
11:	Nov. 2024	6.076159	1.393353	4.545523	6.247341	1.452953	4.638289	5.875732
12:	Dec. 2024	4.685547	1.198824	3.370992	4.825181	1.254616	3.439788	4.575893

	Defd	Befd
	<num>	<num>
1:	1.343455	4.621693
2:	1.553670	4.705996
3:	1.848127	4.798657
4:	2.112064	6.043569
5:	2.253942	7.298609
6:	2.205314	8.767042
7:	1.924962	9.234312
8:	1.830334	8.210807
9:	1.810198	6.262986
10:	1.595528	4.283212
11:	1.359514	4.378935
12:	1.179346	3.280817

Yearly values:

	Dates	Gef0d	Def0d	Bef0d	Gd	Dd	Bd	Gefd	Defd
	<int>	<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

	Befd
	<num>
1:	2193.621

Mode of tracking: two

Inclination limit: 90

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

```
gef_shdhoriz <- calcGef(lat, modeTrk = 'horiz', dataRad = prom,
```

```

modeShd = 'area',
struct = struct, distances = distances)

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
	<char>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	Jan. 2024	4.274445	1.0909303	3.118987	4.528022	1.166334	3.285391	3.826940
2:	Feb. 2024	5.173537	1.3974587	3.699745	5.414413	1.484046	3.839622	4.709780
3:	Mar. 2024	6.270377	1.8008592	4.379272	6.512568	1.906181	4.498391	5.856407
4:	Apr. 2024	8.160354	2.1103041	5.938446	8.429640	2.222836	6.072611	7.744288
5:	May. 2024	9.639011	2.2544315	7.260788	9.932830	2.366831	7.416258	9.158384
6:	Jun. 2024	11.005388	2.1942042	8.675874	11.320680	2.294944	8.861907	10.355140
7:	Jul. 2024	11.220872	1.9183453	9.163290	11.527430	2.000253	9.358648	10.747413
8:	Aug. 2024	10.066277	1.8239013	8.112148	10.352216	1.904515	8.290847	9.601132
9:	Sep. 2024	7.732062	1.7621525	5.864625	7.991813	1.852070	6.013507	7.317424
10:	Oct. 2024	5.023316	1.4757157	3.471271	5.250215	1.568278	3.591050	4.691499
11:	Nov. 2024	4.211801	1.1318865	3.014748	4.452659	1.209397	3.166130	3.846165
12:	Dec. 2024	3.024846	0.9640813	2.008270	3.237139	1.039367	2.135901	2.849995

	Defd	Befd
	<num>	<num>
1:	1.0166151	2.745797
2:	1.3191237	3.314324
3:	1.7298195	4.036342
4:	2.0426359	5.590049
5:	2.1802588	6.854334
6:	2.1029750	8.116855
7:	1.8585724	8.749603
8:	1.7626031	7.708301
9:	1.6984219	5.513717
10:	1.4182254	3.196944
11:	1.0701542	2.710845
12:	0.9330218	1.864479

Yearly values:

	Dates	Gef0d	Def0d	Bef0d	Gd	Dd	Bd	Gefd	Defd
	<int>	<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

	Befd
	<num>
1:	1843.889

Mode of tracking: horiz
Inclination limit: 90

```

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

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

      Befd
      <num>
1: 2.724604
2: 3.287780
3: 4.020914
4: 5.560571
5: 6.833933
6: 8.059004
7: 8.712900
8: 7.678732
9: 5.487114
10: 3.173452
11: 2.681776
12: 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 bajo diferentes condiciones de irradiancia y temperatura. Tiene los siguientes argumentos:

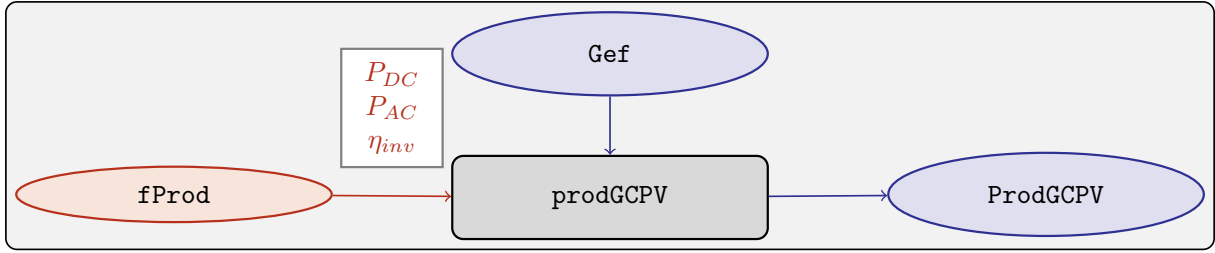


FIGURA 4.6: Estimación de la producción eléctrica de un SFCR mediante la función **prodGCPV**, la cual emplea la función **fProd** para el compute de la potencia a la entrada (P_{DC}), a la salida (P_{AC}) y el rendimiento (η_{inv}) del inversor.

- **inclin**: puede ser un objeto de clase **Gef** o un **data.frame** (o **data.table**). En el segundo caso deberá contener columnas llamadas **Gef** y **Ta**.
- **module**: lista de valores numéricos con la información sobre el módulo fovoltaico.
- **generator**: lista de valores numéricos con la información sobre el generador.
- **inverter**: lista de valores numéricos con la información sobre el inversor DC/AC.
- **effSys**: lista de valores numéricos con la información sobre las pérdidas del sistema.

```

inclin <- calcGef(lat, dataRad = prom, keep.night = FALSE)
## valores por defecto
module <- list(Vocn = 57.6,      #tension de circuito abierto en STC
               Iscn = 4.7,       #corriente de cortocircuito en STC
               Vmn = 46.08,      #tension en el MPP en STC
               Imn = 4.35,       #corriente en el MPP en STC
               Ncs = 96,         #numero de celulas en serie
               Ncp = 1,          #numero de celulas en paralelo
               CoefVT = 0.0023,  #coeficiente de disminuci3n de la
               tension con la temperatura
               TONC = 47)        #temperatura de operaci3n nominal de
               celula
generator <- list(Nms = 12, #numero de modulos en serie
                  Nmp = 11) #numero de modulos en paralelo
inverter <- list(Ki = c(0.01, 0.025, 0.05), #coeficientes de la
                 eficiencia de la curva del inversor
                 Pinv = 25000,               #potencia nominal del
                 inversor
                 Vmin = 420,                 #tension minima del rango
                 de MPP del inversor
                 Vmax = 750,                 #tension maxima del rango
                 de MPP del inversor
                 Gumb = 20)                  #irradiancia umbral para el
                 arranque del inversor
effSys <- list(ModQual = 3, #tolerancia de los modulos
               ModDisp = 2, #perdidas por dispersi3n
               OhmDc = 1.5, #perdidas por efecto Joule antes del
               inversor
               OhmAC = 1.5, #perdidas por efecto Joule despu3s del
               inversor
               MPP = 1,      #error promedio del algoritmo de MPP
               TrafoMT = 1,  #perdidas por el transformador MT
               Disp = 0.5)   #perdidas por las paradas del sistema
prod <- fProd(inclin, module = module, generator = generator,
              inverter = inverter, effSys = effSys)
  
```

```
show(prod)
```

```

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

```

La función **prodGCPV** toma el resultado de **fProd** para obtener un objeto de clase **ProdGCPV**.

```
prod2x <- prodGCPV(lat, modeTrk = 'two', dataRad = prom)
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

```

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**: computa 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**:

```

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

```

```

      Qn stages Qmax Pmn      a      b      c      g      h      i      j
<int> <int> <num> <int> <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
      k      l
<num> <num>
1: 0.095 0.2013

```

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

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

Obtiene como resultado los siguientes valores y funciones:

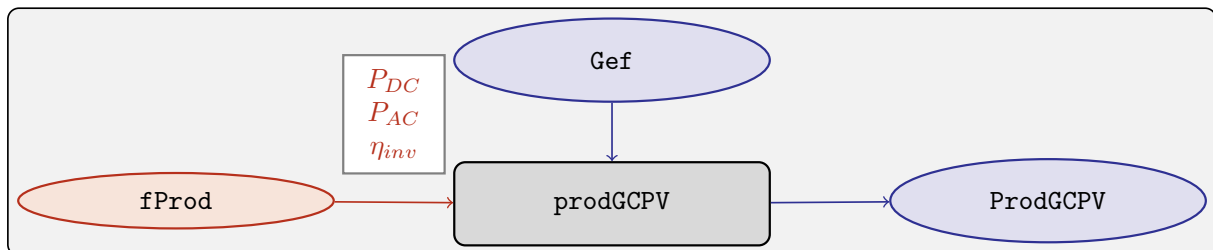


FIGURA 4.7: Estimación de la producción eléctrica de un SFB mediante la función **prodPVPS**, la cual emplea la función **fPump** para el computo del rendimiento de las diferentes parte de una bomba centrífuga alimentada por un convertidor de frecuencia.

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

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

```
[1] 190.100 4084.218
```

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

```
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: 0x000002b9e973e7b8>
<environment: 0x000002b9e725bd18>
```

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

```
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: 0x000002b9e973e7b8>
<environment: 0x000002b9e70b3220>
```

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

```
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: 0x000002b9e973e7b8>
<environment: 0x000002b9e660ddf0>
```

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

```
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: 0x000002b9e973e7b8>
<environment: 0x000002b9e724fd88>
```

Se pueden realizar operaciones con este objeto:

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

Q	Pac	Pb	Ph	etam	etab	f
<num>	<num>	<num>	<num>	<num>	<num>	<num>

```

1: 0.3133325 190.1000 124.8346 34.15325 0.6566786 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

```

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

```

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

```
struct2x <- list(W = 23.11, L = 9.8, Nrow = 2, Ncol = 3)
dist2x <- list(Lew = c(30, 45), Lns = c(20, 40))
ShdM2x <- optimShd(lat, dataRad = prom, modeTrk = 'two',
                  modeShd = c('area', 'prom'),
                  distances = dist2x, struct = struct2x,
                  res = 5)
show(ShdM2x)
```

```
|
|
|
|====| 0%
|=====| 5%
|=====| 10%
|=====| 14%
|=====| 19%
|=====| 24%
|=====| 29%
|=====| 33%
|=====| 38%
|=====| 43%
|=====| 48%
|=====| 52%
|=====| 57%
|=====| 62%
|=====| 67%
|=====| 71%
|=====| 76%
|=====| 81%
|=====| 86%
|
```



```

=====| 90%
|
=====| 95%
|
=====| 100%
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
Min. :30.00	Min. :20	Min. :0	Min. :0.01509	Min. :2.649
1st Qu.:33.75	1st Qu.:25	1st Qu.:0	1st Qu.:0.02223	1st Qu.:3.946
Median :37.50	Median :30	Median :0	Median :0.02870	Median :4.802
Mean :37.50	Mean :30	Mean :0	Mean :0.03463	Mean :4.967
3rd Qu.:41.25	3rd Qu.:35	3rd Qu.:0	3rd Qu.:0.03945	3rd Qu.:6.016
Max. :45.00	Max. :40	Max. :0	Max. :0.07769	Max. :7.948

Yf

Min. :2104
 1st Qu.:2192
 Median :2216
 Mean :2203
 3rd Qu.:2231
 Max. :2247

Además, con el argumento **prog**, podemos indicar si queremos que indique con una barra de progreso el estado de la secuencia.

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

```

structFixed = list(L = 5)
distFixed = list(D = structFixed$L*c(1,3))
Shd12Fixed <- optimShd(lat = lat, dataRad = prom,
                      modeTrk = 'fixed',
                      distances = distFixed, res = 2,
                      struct = structFixed,
                      modeShd = 'area',
                      prog = FALSE) #Se quita la barra de progreso
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** cuanta con una serie de métodos que ayudan a la compresión de los datos obtenidos.

4.8.1. Geometría solar

Para la clase **Sol** no hay métodos de visualización directos, pero se puede utilizar el método **as.data.table** [sección A.4.1] para mostrar resultados.

```

lat <- 37.2
BTd <- fBTd(mode = 'serie')
sol <- calcSol(lat, BTd = BTd[100])

```

```
show(sol)
```

Object of class Sol

Latitude: 37.2 degrees

Daily values:

Dates	decl	eo	EoT
Min. :2024-04-09	Min. :0.1373	Min. :0.9961	Min. :-0.004637
1st Qu.:2024-04-09	1st Qu.:0.1373	1st Qu.:0.9961	1st Qu.: -0.004637
Median :2024-04-09	Median :0.1373	Median :0.9961	Median : -0.004637
Mean :2024-04-09	Mean :0.1373	Mean :0.9961	Mean : -0.004637
3rd Qu.:2024-04-09	3rd Qu.:0.1373	3rd Qu.:0.9961	3rd Qu.: -0.004637
Max. :2024-04-09	Max. :0.1373	Max. :0.9961	Max. : -0.004637

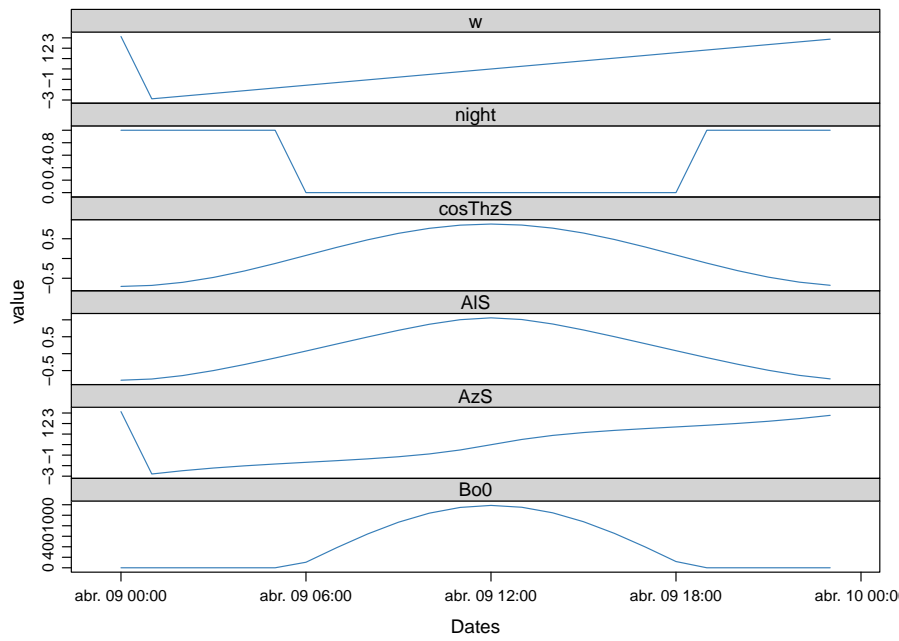
ws	BoOd
Min. :-1.676	Min. :9605
1st Qu.: -1.676	1st Qu.:9605
Median : -1.676	Median :9605
Mean : -1.676	Mean :9605
3rd Qu.: -1.676	3rd Qu.:9605
Max. : -1.676	Max. :9605

Intradaily values:

Dates	w	night
Min. :2024-04-09 00:00:00	Min. :-2.8867	Mode :logical
1st Qu.:2024-04-09 05:45:00	1st Qu.: -1.3810	FALSE:13
Median :2024-04-09 11:30:00	Median : 0.1246	TRUE :11
Mean :2024-04-09 11:30:00	Mean : 0.1245	
3rd Qu.:2024-04-09 17:15:00	3rd Qu.: 1.6302	
Max. :2024-04-09 23:00:00	Max. : 3.1347	

cosThzS	AlS	AzS	BoO
Min. :-0.70627	Min. :-0.78421	Min. :-2.7936	Min. : 0.0
1st Qu.: -0.47365	1st Qu.: -0.49344	1st Qu.: -1.5662	1st Qu.: 0.0
Median : 0.08252	Median : 0.08261	Median : 0.2381	Median : 112.4
Mean : 0.08260	Mean : 0.10467	Mean : 0.1245	Mean : 401.0
3rd Qu.: 0.63882	3rd Qu.: 0.69297	3rd Qu.: 1.7167	3rd Qu.: 869.9
Max. : 0.87177	Max. : 1.05880	Max. : 3.1319	Max. :1187.1

```
xyplot(as.data.tableI(sol))
```



4.8.2. Datos meteorológicos

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

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

Object of class `Meteo`

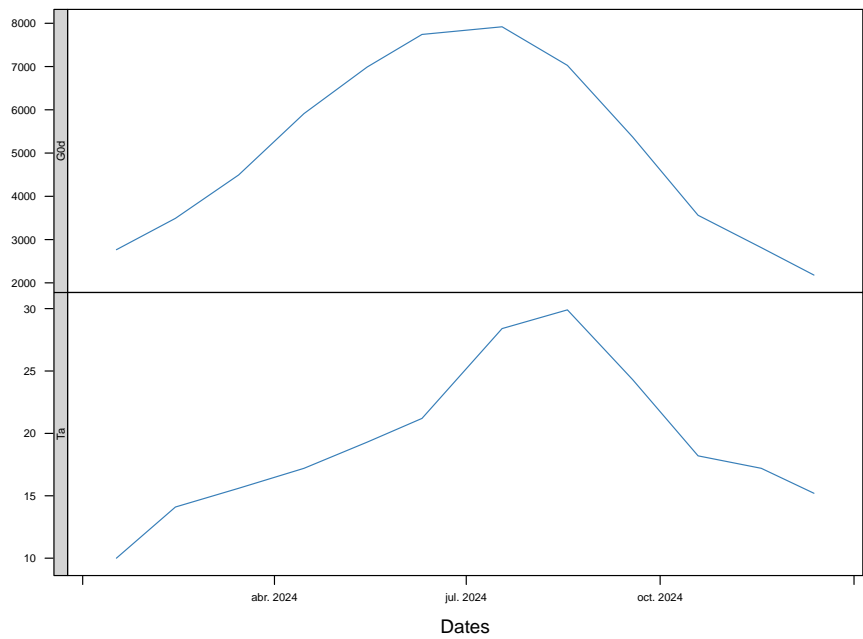
Source of meteorological information: prom-

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

```
xyplot(BD)
```



4.8.3. Radiación en el plano horizontal

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

```
g0 <- calcG0(lat, dataRad = list(G0dm = G0dm, Ta = Ta))
show(g0)
```

Object of class G0

Source of meteorological information: prom-

Latitude of source: 37.2 degrees

Latitude for calculations: 37.2 degrees

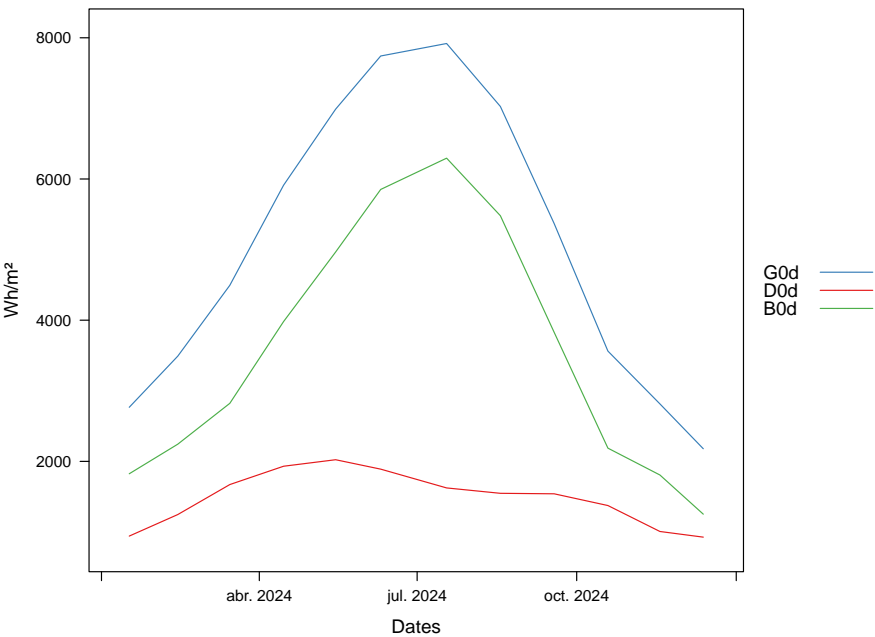
Monthly avarages:

	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

```
xyplot(g0)
```



Y con un método `compare`.

4.8.4. Radiación efectiva en el plano del generador

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

```
gef <- calcGef(lat, dataRad = list(G0dm = G0dm, Ta = Ta))
show(gef)
```

Object of class Gef

Source of meteorological information:prom-

Latitude of source: 37.2 degrees

Latitude for calculations: 37.2 degrees

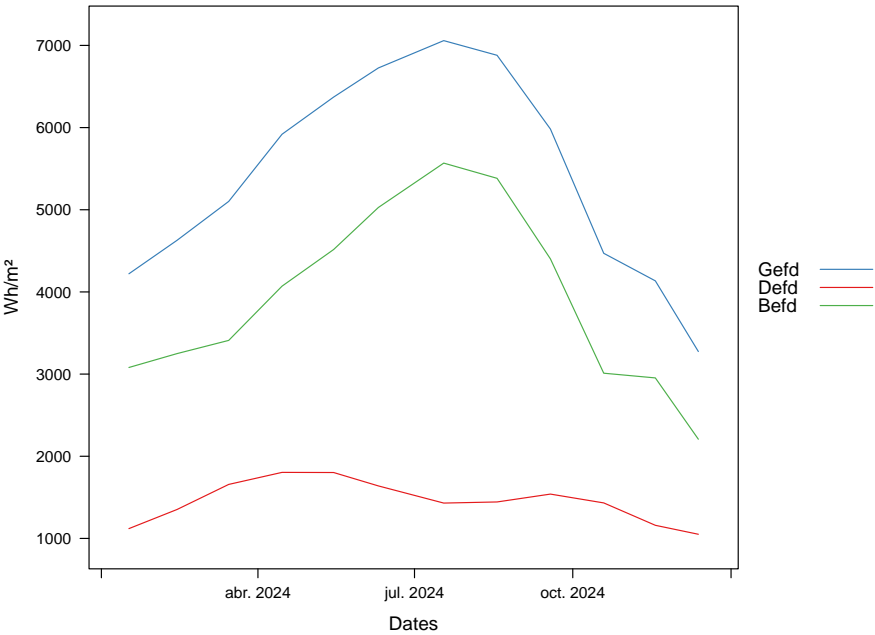
Monthly avarages:

Dates	Bod	Bnd	Gd	Dd	Bd	Gefd	Defd
<char>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1: Jan. 2024	8.724907	4.924221	4.489744	1.200992	3.258164	4.220907	1.119517
2: Feb. 2024	9.592013	5.034287	4.919206	1.451954	3.428647	4.628492	1.352529
3: Mar. 2024	10.281308	5.163713	5.413543	1.779951	3.583896	5.101556	1.657369
4: Apr. 2024	10.527227	6.408617	6.282631	1.936897	4.280357	5.918787	1.803811
5: May. 2024	10.431853	7.617499	6.784202	1.937331	4.769584	6.371295	1.802060
6: Jun. 2024	10.291163	9.102430	7.173475	1.762326	5.325535	6.725684	1.639192
7: Jul. 2024	10.305302	10.037233	7.511733	1.533887	5.890275	7.058263	1.430322
8: Aug. 2024	10.394682	8.640959	7.295543	1.545089	5.672747	6.879777	1.443952
9: Sep. 2024	10.233884	6.698488	6.335591	1.647975	4.628244	5.982520	1.539552
10: Oct. 2024	9.659077	4.546024	4.746760	1.538325	3.169044	4.470026	1.432213
11: Nov. 2024	8.798687	4.638289	4.393712	1.244217	3.118376	4.134590	1.159756


```
12: Dec. 2024  8.176298  3.439788  3.478125  1.128381  2.325648  3.274677  1.050626
      Befd
      <num>
1: 3.080392
2: 3.249460
3: 3.410070
4: 4.070094
5: 4.516177
6: 5.027718
7: 5.567823
8: 5.382478
9: 4.402209
10: 3.010771
11: 2.953471
12: 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
```

```
xyplot(gef)
```



4.8.5. Producción eléctrica de un SFCR

La clase ==

Ejemplo práctico de aplicación

Como demostración se va a realizar un caso práctico...

5.1. solaR

...

5.2. PVsyst

...

5.3. solaR

...

5.4. Comparación entre los tres

Código completo

Todo el código que se muestra a continuación está disponible...

A.1. Constructores

A.1.1. calcSol

```
calcSol <- function(lat, BTd,
                    sample = 'hour', BTi,
                    EoT = TRUE,
                    keep.night = TRUE,
                    method = 'michalsky')
{
  if(missing(BTd)) BTd <- truncDay(BTi)
  sold <- fSold(lat, BTd, method = method) #daily values
  soli <- fSoli(sold = sold, sample = sample, #intradaily values
               BTi = BTi, keep.night = keep.night,
               EoT = EoT, method = method)

  if(!missing(BTi)){
    sample <- soli$Dates[2]-soli$Dates[1]
    sample <- format(sample)
  }

  sold[, lat := NULL]
  soli[, lat := NULL]
  result <- new('Sol',
               lat = lat,
               sold = sold,
               soli = soli,
               sample = sample,
               method = method)

  return(result)
}
```

EXTRACTO DE CÓDIGO A.1: *calcSol*

A.1.2. calcG0

```
calcG0 <- function(lat,
```

```

        modeRad='prom',
        dataRad,
        sample='hour',
        keep.night=TRUE,
        sunGeometry='michalsky',
        corr, f, ...)
{

  if (missing(lat)) stop('lat missing. You must provide a latitude value.')

  stopifnot(modeRad %in% c('prom', 'aguilar','bd', 'bdI'))

###Datos de Radiacion
  if (missing(corr)){
    corr = switch(modeRad,
      values
      bd = 'CPR', #Correlation between Fd and Kt for daily
      values
      aguilar = 'CPR', #Correlation between Fd and Kt for daily
      values
      prom = 'Page', #Correlation between Fd and Kt for monthly
      averages
      bdI = 'BRL' #Correlation between fd and kt for
      intraday values
    )
  }

  if(is(dataRad, 'Meteo')){BD <- dataRad}
  else{
    BD <- switch(modeRad,
      bd = {
        if (!is.list(dataRad)) dataRad <- list(file=dataRad)
        switch(class(dataRad$file)[1],
          character={
            bd.default=list(file='', lat=lat)
            bd=modifyList(bd.default, dataRad)
            res <- do.call('readBDd', bd)
            res
          },
          data.table= ,
          data.frame={
            bd.default=list(file='', lat=lat)
            bd=modifyList(bd.default, dataRad)
            res <- do.call('dt2Meteo', bd)
            res
          },
          zoo={
            bd.default=list(file='', lat=lat, source='')
            bd=modifyList(bd.default, dataRad)
            res <- do.call('zoo2Meteo', bd)
            res
          }
        })
      }, #End of bd
    prom = {
      if (!is.list(dataRad)) dataRad <- list(G0dm=dataRad)
      prom.default <- list(G0dm=numeric(), lat=lat)
      prom = modifyList(prom.default, dataRad)
      res <- do.call('readG0dm', prom)
    }
  }
}

```

```

    }, #End of prom
    aguiar = {
      if (is.list(dataRad)) dataRad <- dataRad$G0dm
      BTd <- fBTd(mode='serie')
      sold <- fSold(lat, BTd)
      G0d <- markovG0(dataRad, sold)
      res <- dt2Meteo(G0d, lat=lat, source='aguiar')
    }, #End of aguiar
    bdI = {
      if (!is.list(dataRad)) dataRad <- list(file=dataRad)
      switch(class(dataRad$file)[1],
        character = {
          bdI.default <- list(file='', lat=lat)
          bdI <- modifyList(bdI.default, dataRad)
          res <- do.call('readBDi', bdI)
          res
        },
        data.table = ,
        data.frame = {
          bdI.default <- list(file='', lat=lat)
          bdI <- modifyList(bdI.default, dataRad)
          res <- do.call('dt2Meteo', bdI)
          res
        },
        zoo = {
          bdI.default <- list(file='', lat=lat, source='')
          bdI <- modifyList(bdI.default, dataRad)
          res <- do.call('zoo2Meteo', bdI)
          res
        },
        stop('dataRad$file should be a character, a data.
table, a data.frame or a zoo.')
      )} #End of btI
    ) #End of general switch
  }

### Angulos solares y componentes de irradiancia
if (modeRad=='bdI') {
  sol <- calcSol(lat, sample = sample,
    BTi = indexD(BD), keep.night=keep.night, method=
sunGeometry)
  compI <- fCompI(sol=sol, G0I=BD, corr=corr, f=f, ...)
  compD <- compI[, lapply(.SD, P2E, sol@sample),
    .SDcols = c('G0', 'D0', 'B0'),
    by = truncDay(Dates)]
  names(compD)[1] <- 'Dates'
  names(compD)[-1] <- paste(names(compD)[-1], 'd', sep = '')
  compD$Fd <- compD$D0d/compD$G0d
  compD$Kt <- compD$G0d/sol@sold$Bo0d
} else { ##modeRad!='bdI'
  sol <- calcSol(lat, indexD(BD), sample = sample,
    keep.night = keep.night, method = sunGeometry)
  compD<-fCompD(sol=sol, G0d=BD, corr=corr, f, ...)
  compI<-fCompI(sol=sol, compD=compD, ...)
}

###Temperature

```

```

Ta=switch(modeRad,
  bd={
    if (all(c("TempMax", "TempMin") %in% names(BD@data))) {
      fTemp(sol, BD)
    } else {
      if ("Ta" %in% names(BD@data)) {
        data.table(Dates = indexD(sol),
          Ta = BD@data$Ta)
      } else {
        warning('No temperature information available!')
      }
    }
  },
  bdI={
    if ("Ta" %in% names(BD@data)) {
      data.table(Dates = indexI(sol),
        Ta = BD@data$Ta)
    } else {
      warning('No temperature information available!')
    }
  },
  prom={
    if ("Ta" %in% names(BD@data)) {
      data.table(Dates = indexD(sol),
        Ta = BD@data$Ta)
    } else {
      warning('No temperature information available!')
    }
  },
  aguiar={
    data.table(Dates = indexI(sol),
      Ta = BD@data$Ta)
  }
)

###Medias mensuales y anuales
nms <- c('G0d', 'D0d', 'B0d')
G0dm <- compD[, lapply(.SD/1000, mean, na.rm = TRUE),
  .SDcols = nms,
  by = .(month(Dates), year(Dates))]

if(modeRad == 'prom'){
  G0dm[, DayOfMonth := DOM(G0dm)]
  G0y <- G0dm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
    .SDcols = nms,
    by = .(Dates = year)]
  G0dm[, DayOfMonth := NULL]
} else{
  G0y <- compD[, lapply(.SD/1000, sum, na.rm = TRUE),
    .SDcols = nms,
    by = .(Dates = year(Dates))]
}
G0dm[, Dates := paste(month.abb[month], year, sep = '. ')]
G0dm[, c('month', 'year') := NULL]
setcolorder(G0dm, 'Dates')

###Result

```



```

result <- new(Class='G0',
              BD,          #G0 contains "Meteo"
              sol,         #G0 contains 'Sol'
              GOD=compD,   #results of fCompD
              G0dm=G0dm,   #monthly means
              G0y=G0y,     #yearly values
              G0I=compI,   #results of fCompD
              Ta=Ta        #ambient temperature
              )
return(result)
}

```

EXTRACTO DE CÓDIGO A.2: *calcG0*

A.1.3. *calcGef*

```

calcGef<-function(lat,
                  modeTrk='fixed',      #c('two','horiz','fixed')
                  modeRad='prom',
                  dataRad,
                  sample='hour',
                  keep.night=TRUE,
                  sunGeometry='michalsky',
                  corr, f,
                  betaLim=90, beta=abs(lat)-10, alfa=0,
                  iS=2, alb=0.2, horizBright=TRUE, HCPV=FALSE,
                  modeShd='',          #modeShd=c('area','bt','prom')
                  struct=list(), #list(W=23.11, L=9.8, Nrow=2, Ncol=8),
                  distances=data.frame(), #data.table(Lew=40, Lns=30, H=0)){
  ...){

  stopifnot(is.list(struct), is.data.frame(distances))

  if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
    modeShd[which(modeShd=='bt')]='area'
    warning('backtracking is only implemented for modeTrk=horiz')}

  if (modeRad!='prev'){ #not use a prev calculation
    radHoriz <- calcG0(lat=lat, modeRad=modeRad,
                      dataRad=dataRad,
                      sample=sample, keep.night=keep.night,
                      sunGeometry=sunGeometry,
                      corr=corr, f=f, ...)
  } else {
    radHoriz <- as(dataRad, 'G0') #use a prev calculation
  }

  ### Inclined and effective radiation
  BT=("'bt'" %in% modeShd)
  angGen <- fTheta(radHoriz, beta, alfa, modeTrk, betaLim, BT, struct,
                  distances)
  inclin <- fInclin(radHoriz, angGen, iS, alb, horizBright, HCPV)

  ### Daily, monthly and yearly values
  by <- radHoriz@sample
  nms <- c('Bo', 'Bn', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
  nmsd <- paste(nms, 'd', sep = '')

```

```

if(radHoriz@type == 'prom'){
  Gefdm <- inclin[, lapply(.SD/1000, P2E, by),
                        .SDcols = nms,
                        by = .(month(Dates), year(Dates)))]
  names(Gefdm)[-c(1,2)] <- nmsd
  GefD <- Gefdm[, .SD*1000,
                  .SDcols = nmsd,
                  by = .(Dates = indexD(radHoriz)))]

  Gefdm[, DayOfMonth := DOM(Gefdm)]
  Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                  .SDcols = nmsd,
                  by = .(Dates = year)]
  Gefdm[, DayOfMonth := NULL]
} else{
  GefD <- inclin[, lapply(.SD, P2E, by),
                    .SDcols = nms,
                    by = .(Dates = truncDay(Dates)))]
  names(GefD)[-1] <- nmsd

  Gefdm <- GefD[, lapply(.SD/1000, mean, na.rm = TRUE),
                  .SDcols = nmsd,
                  by = .(month(indexD(radHoriz)), year(indexD(radHoriz))))]
  Gefy <- GefD[, lapply(.SD/1000, sum, na.rm = TRUE),
                  .SDcols = nmsd,
                  by = .(Dates = year(indexD(radHoriz))))]
}

Gefdm[, Dates := paste(month.abb[month], year, sep = '. ')]
Gefdm[, c('month', 'year') := NULL]
setcolororder(Gefdm, 'Dates')

###Resultado antes de sombras
result0=new('Gef',
            radHoriz,
            Theta=angGen,
            GefD=GefD,
            Gefdm=Gefdm,
            Gefy=Gefy,
            GefI=inclin,
            iS=iS,
            alb=alb,
            modeTrk=modeTrk,
            modeShd=modeShd,
            angGen=list(alfa=alfa, beta=beta, betaLim=betaLim),
            struct=struct,
            distances=distances
            )

###Shadows
if (isTRUE(modeShd == "") || #If modeShd==' ' there is no shadow
calculation
('bt' %in% modeShd)) { #nor if there is backtracking
  return(result0)
} else {
  result <- calcShd(result0, modeTrk, modeShd, struct, distances)
  return(result)
}

```

}

EXTRACTO DE CÓDIGO A.3: *calcGef*A.1.4. **prodGCPV**

```

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

  stopifnot(is.list(module),
    is.list(generator),
    is.list(inverter),
    is.list(effSys),
    is.list(struct),
    is.data.table(distances))

  if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
    modeShd[which(modeShd=='bt')]='area'
    warning('backtracking is only implemented for modeTrk=horiz')}

  if (modeRad!='prev'){ #We do not use a previous calculation

    radEf<-calcGef(lat=lat, modeTrk=modeTrk, modeRad=modeRad,
      dataRad=dataRad,
      sample=sample, keep.night=keep.night,
      sunGeometry=sunGeometry,
      corr=corr, f=f,
      betaLim=betaLim, beta=beta, alfa=alfa,
      iS=iS, alb=alb, horizBright=horizBright, HCPV=HCPV,
      modeShd=modeShd, struct=struct, distances=distances, ...)

  } else { #We use a previous calcG0, calcGef or prodGCPV calculation.

    stopifnot(class(dataRad) %in% c('G0', 'Gef', 'ProdGCPV'))
    radEf <- switch(class(dataRad),
      G0=calcGef(lat=lat,
        modeTrk=modeTrk, modeRad='prev',
        dataRad=dataRad,
        betaLim=betaLim, beta=beta, alfa=alfa,
        iS=iS, alb=alb, horizBright=horizBright, HCPV=
HCPV,
        modeShd=modeShd, struct=struct, distances=
distances, ...),

```

```

        Gef=dataRad,
        ProdGCPV=as(dataRad, 'Gef')
    )
}

##Production
prodI<-fProd(radEf,module,generator,inverter,effSys)
module=attr(prodI, 'module')
generator=attr(prodI, 'generator')
inverter=attr(prodI, 'inverter')
effSys=attr(prodI, 'effSys')

##Calculation of daily, monthly and annual values
Pg=generator$Pg #Wp

by <- radEf@sample
nms1 <- c('Pac', 'Pdc')
nms2 <- c('Eac', 'Edc', 'Yf')

if(radEf@type == 'prom'){
  prodDm <- prodI[, lapply(.SD/1000, P2E, by),
                        .SDcols = nms1,
                        by = .(month(Dates), year(Dates))]
  names(prodDm)[-c(1,2)] <- nms2[-3]
  prodDm[, Yf := Eac/(Pg/1000)]
  prodD <- prodDm[, .SD*1000,
                  .SDcols = nms2,
                  by = .(Dates = indexD(radEf))]
  prodD[, Yf := Yf/1000]

  prodDm[, DayOfMonth := DOM(prodDm)]
  prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                  .SDcols = nms2,
                  by = .(Dates = year)]
  prodDm[, DayOfMonth := NULL]
} else {
  prodD <- prodI[, lapply(.SD, P2E, by),
                  .SDcols = nms1,
                  by = .(Dates = truncDay(Dates))]
  names(prodD)[-1] <- nms2[-3]
  prodD[, Yf := Eac/Pg]

  prodDm <- prodD[, lapply(.SD/1000, mean, na.rm = TRUE),
                  .SDcols = nms2,
                  by = .(month(Dates), year(Dates))]
  prodDm[, Yf := Yf * 1000]
  prody <- prodD[, lapply(.SD/1000, sum, na.rm = TRUE),
                  .SDcols = nms2,
                  by = .(Dates = year(Dates))]
  prody[, Yf := Yf * 1000]
}

prodDm[, Dates := paste(month.abb[month], year, sep = '. ')]
prodDm[, c('month', 'year') := NULL]
setcolorder(prodDm, 'Dates')

```

```

result <- new('ProdGCPV',
             radEf,                #contains 'Gef'
             prodD=prodD,
             prodDm=prodDm,
             prody=prody,
             prodI=prodI,
             module=module,
             generator=generator,
             inverter=inverter,
             effSys=effSys
             )
}

```

EXTRACTO DE CÓDIGO A.4: *prodGCPV*

A.1.5. prodPVPS

```

prodPVPS<-function(lat,
                   modeTrk='fixed',
                   modeRad='prom',
                   dataRad,
                   sample='hour',
                   keep.night=TRUE,
                   sunGeometry='michalsky',
                   corr, f,
                   betaLim=90, beta=abs(lat)-10, alfa=0,
                   iS=2, alb=0.2, horizBright=TRUE, HCPV=FALSE,
                   pump , H,
                   Pg, converter= list(), #Pnom=Pg, Ki=c(0.01,0.025,0.05)),
                   effSys=list(),
                   ...){

  stopifnot(is.list(converter),
            is.list(effSys))

  if (modeRad!='prev'){ #We do not use a previous calculation

    radEf<-calcGef(lat=lat, modeTrk=modeTrk, modeRad=modeRad,
                  dataRad=dataRad,
                  sample=sample, keep.night=keep.night,
                  sunGeometry=sunGeometry,
                  corr=corr, f=f,
                  betaLim=betaLim, beta=beta, alfa=alfa,
                  iS=iS, alb=alb, horizBright=horizBright, HCPV=HCPV,
                  modeShd='', ...)

  } else { #We use a previous calculation of calcG0, calcGef or prodPVPS
    stopifnot(class(dataRad) %in% c('G0', 'Gef', 'ProdPVPS'))
    radEf <- switch(class(dataRad),
                   G0=calcGef(lat=lat,
                              modeTrk=modeTrk, modeRad='prev',
                              dataRad=dataRad,
                              betaLim=betaLim, beta=beta, alfa=alfa,
                              iS=iS, alb=alb, horizBright=horizBright, HCPV=HCPV,
                              modeShd='', ...),
                   Gef=dataRad,
                   ProdPVPS=as(dataRad, 'Gef')
                   )
  }
}

```

```

}

###Electric production
converter.default=list(Ki = c(0.01,0.025,0.05), Pnom=Pg)
converter=modifyList(converter.default, converter)

effSys.default=list(ModQual=3,ModDisp=2,OhmDC=1.5,OhmAC=1.5,MPP=1,TrafoMT=1,
Disp=0.5)
effSys=modifyList(effSys.default, effSys)

TONC=47
Ct=(TONC-20)/800
lambda=0.0045
Gef=radEf@GefI$Gef
night=radEf@solI$night
Ta=radEf@Ta$Ta

Tc=Ta+Ct*Gef
Pdc=Pg*Gef/1000*(1-lambda*(Tc-25))
Pdc[is.na(Pdc)]=0 #Necessary for the functions provided by fPump
PdcN=with(effSys,
          Pdc/converter$Pnom*(1-ModQual/100)*(1-ModDisp/100)*(1-OhmDC/100)
          )
PacN=with(converter,{
  A=Ki[3]
  B=Ki[2]+1
  C=Ki[1]-(PdcN)
  ##AC power normalized to the inverter
  result=(-B+sqrt(B^2-4*A*C))/(2*A)
})
PacN[PacN<0]<-0

Pac=with(converter,
          PacN*Pnom*(1-effSys$OhmAC/100))
Pdc=PdcN*converter$Pnom*(Pac>0)

###Pump
fun<-fPump(pump=pump, H=H)
##I limit power to the pump operating range.
rango=with(fun,Pac>=lim[1] & Pac<=lim[2])
Pac[!rango]<-0
Pdc[!rango]<-0
prodI=data.table(Pac=Pac,Pdc=Pdc,Q=0,Pb=0,Ph=0,f=0)
prodI=within(prodI,{
  Q[rango]<-fun$fQ(Pac[rango])
  Pb[rango]<-fun$fPb(Pac[rango])
  Ph[rango]<-fun$fPh(Pac[rango])
  f[rango]<-fun$fFreq(Pac[rango])
  etam=Pb/Pac
  etab=Ph/Pb
})

prodI[night,]<-NA
prodI[, Dates := indexI(radEf)]
setcolorder(prodI, c('Dates', names(prodI)[-length(prodI)]))

###daily, monthly and yearly values

```

```

by <- radEf@sample

if(radEf@type == 'prom'){
  prodDm <- prodI[, .(Eac = P2E(Pac, by)/1000,
                        Qd = P2E(Q, by)),
                    by = .(month(Dates), year(Dates))]
  prodDm[, Yf := Eac/(Pg/1000)]

  prodD <- prodDm[, .(Eac = Eac*1000,
                      Qd,
                      Yf),
                  by = .(Dates = indexD(radEf))]

  prodDm[, DayOfMonth := DOM(prodDm)]

  prody <- prodDm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                  .SDcols = c('Eac', 'Qd', 'Yf'),
                  by = .(Dates = year)]
  prodDm[, DayOfMonth := NULL]
} else {
  prodD <- prodI[, .(Eac = P2E(Pac, by)/1000,
                    Qd = P2E(Q, by)),
                  by = .(Dates = truncDay(Dates))]
  prodD[, Yf := Eac/Pg*1000]

  prodDm <- prodD[, lapply(.SD, mean, na.rm = TRUE),
                  .SDcols = c('Eac', 'Qd', 'Yf'),
                  by = .(month(Dates), year(Dates))]
  prody <- prodD[, lapply(.SD, sum, na.rm = TRUE),
                  .SDcols = c('Eac', 'Qd', 'Yf'),
                  by = .(Dates = year(Dates))]

}

prodDm[, Dates := paste(month.abb[month], year, sep = '. ')]
prodDm[, c('month', 'year') := NULL]
setcolorder(prodDm, 'Dates')

result <- new('ProdPVPS',
             radEf,
             prodD=prodD,
             prodDm=prodDm,
             prody=prody,
             prodI=prodI,
             pump=pump,
             H=H,
             Pg=Pg,
             converter=converter,
             effSys=effSys
            )
}

```

EXTRACTO DE CÓDIGO A.5: *prodGCPV*

A.1.6. calcShd

```
calcShd<-function(radEf,##class='Gef'
```

```

        modeTrk='fixed',      #c('two','horiz','fixed')
        modeShd='prom',      #modeShd=c('area','bt','prom')
        struct=list(), #list(W=23.11, L=9.8, Nrow=2, Ncol=8),
        distances=data.frame() #data.table(Lew=40, Lns=30, H=0)){
}

stopifnot(is.list(struct), is.data.frame(distances))

##For now I only use modeShd = 'area'
##With different modeShd (to be defined) I will be able to calculate Gef in
a different way
##See macagnan thesis
prom=("prom" %in% modeShd)
prev <- as.data.tableI(radEf, complete=TRUE)
## shadow calculations
sol <- data.table(AzS = prev$AzS,
                  Als = prev$Als)
theta <- radEf@Theta
AngGen <- data.table(theta, sol)
FS <- fSombra(AngGen, distances, struct, modeTrk, prom)
## irradiance calculation
gef0 <- radEf@GefI
Bef0 <- gef0$Bef
Dcef0 <- gef0$Dcef
Gef0 <- gef0$Gef
Dief0 <- gef0$Dief
Ref0 <- gef0$Ref
## calculation
Bef <- Bef0*(1-FS)
Dcef <- Dcef0*(1-FS)
Def <- Dief0+Dcef
Gef <- Dief0+Ref0+Bef+Dcef #Including shadows
##Change names
nms <- c('Gef', 'Def', 'Dcef', 'Bef')
nmsIndex <- which(names(gef0) %in% nms)
names(gef0)[nmsIndex] <- paste(names(gef0)[nmsIndex], '0', sep='')
GefShd <- gef0
GefShd[, c(nms, 'FS') := .(Gef, Def, Dcef, Bef, FS)]

## daily, monthly and yearly values
by <- radEf@sample
nms <- c('Gef0', 'Def0', 'Bef0', 'G', 'D', 'B', 'Gef', 'Def', 'Bef')
nmsd <- paste(nms, 'd', sep = '')

Gefdm <- GefShd[, lapply(.SD/1000, P2E, by),
                 by = .(month(truncDay(Dates)), year(truncDay(Dates))),
                 .SDcols = nms]
names(Gefdm)[-c(1, 2)] <- nmsd

if(radEf@type == 'prom'){
  GefD <- Gefdm[, .SD[, -c(1, 2)] * 1000,
                 .SDcols = nmsd,
                 by = .(Dates = indexD(radEf))]

  Gefdm[, DayOfMonth := DOM(Gefdm)]

  Gefy <- Gefdm[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                 .SDcols = nmsd,

```



```

        by = .(Dates = year)]
    Gefdm[, DayOfMonth := NULL]
  } else{
    GefD <- GefShd[, lapply(.SD/1000, P2E, by),
                        .SDcols = nms,
                        by = .(Dates = truncDay(Dates)))]
    names(GefD)[-1] <- nmsd

    Gefy <- GefD[, lapply(.SD[, -1], sum, na.rm = TRUE),
                  .SDcols = nmsd,
                  by = .(Dates = year(Dates)))]
  }

  Gefdm[, Dates := paste(month.abb[month], year, sep = '. ')]
  Gefdm[, c('month', 'year') := NULL]
  setcolorder(Gefdm, c('Dates', names(Gefdm)[-length(Gefdm)]))

  ## Object of class Gef
  ## modifying the 'modeShd', 'GefI', 'GefD', 'Gefdm', and 'Gefy' slots
  ## from the original radEf object
  radEf@modeShd=modeShd
  radEf@GefI=GefShd
  radEf@GefD=GefD
  radEf@Gefdm=Gefdm
  radEf@Gefy=Gefy
  return(radEf)
}

```

EXTRACTO DE CÓDIGO A.6: *calcShd*

A.1.7. optimShd

```

optimShd<-function(lat,
  modeTrk='fixed',
  modeRad='prom',
  dataRad,
  sample='hour',
  keep.night=TRUE,
  sunGeometry='michalsky',
  betaLim=90, beta=abs(lat)-10, alfa=0,
  iS=2, alb=0.2, HCPV=FALSE,
  module=list(),
  generator=list(),
  inverter=list(),
  effSys=list(),
  modeShd='',
  struct=list(),
  distances=data.table(),
  res=2,      #resolution, distance spacing
  prog=TRUE){ #Drawing progress bar

  if (('bt' %in% modeShd) & (modeTrk!='horiz')) {
    modeShd[which(modeShd=='bt')]='area'
    warning('backtracking is only implemented for modeTrk=horiz')}

  ##I save function arguments for later use

  listArgs<-list(lat=lat, modeTrk=modeTrk, modeRad=modeRad,

```

```

        dataRad=dataRad,
        sample=sample, keep.night=keep.night,
        sunGeometry=sunGeometry,
        betaLim=betaLim, beta=beta, alfa=alfa,
        iS=iS, alb=alb, HCPV=HCPV,
        module=module, generator=generator,
        inverter=inverter, effSys=effSys,
        modeShd=modeShd, struct=struct,
        distances=data.table(Lew=NA, Lns=NA, D=NA))

##I think network on which I will do the calculations
Red=switch(modeTrk,
    horiz=with(distances,
        data.table(Lew=seq(Lew[1],Lew[2],by=res),
            H=0)),
    two=with(distances,
        data.table(
            expand.grid(Lew=seq(Lew[1],Lew[2],by=res),
                Lns=seq(Lns[1],Lns[2],by=res),
                H=0))),
    fixed=with(distances,
        data.table(D=seq(D[1],D[2],by=res),
            H=0))
)

casos<-dim(Red)[1] #Number of possibilities to study

##I prepare the progress bar
if (prog) {pb <- txtProgressBar(min = 0, max = casos+1, style = 3)
    setTxtProgressBar(pb, 0)}

###Calculations
##Reference: No shadows
listArgs0 <- modifyList(listArgs,
    list(modeShd='', struct=NULL, distances=NULL) )
Prod0<-do.call(prodGCPV, listArgs0)
YfAnual0=mean(Prod0@prody$Yf) #I use mean in case there are several years
if (prog) {setTxtProgressBar(pb, 1)}

##The loop begins

##I create an empty vector of the same length as the cases to be studied
YfAnual<-numeric(casos)

BT=('bt' %in% modeShd)
if (BT) { ##There is backtracking, then I must start from horizontal
radiation.
    RadBT <- as(Prod0, 'GO')
    for (i in seq_len(casos)){
        listArgsBT <- modifyList(listArgs,
            list(modeRad='prev', dataRad=RadBT,
                distances=Red[i,]))
        prod.i <- do.call(prodGCPV, listArgsBT)
        YfAnual[i]=mean(prod.i@prody$Yf)
        if (prog) {setTxtProgressBar(pb, i+1)}
    }
} else {

```

```

    prom=('prom' %in% modeShd)
    for (i in seq_len(casos)){
      Gef0=as(Prod0, 'Gef')
      GefShd=calcShd(Gef0, modeTrk=modeTrk, modeShd=modeShd,
                     struct=struct, distances=Red[i,])
      listArgsShd <- modifyList(listArgs,
                               list(modeRad='prev', dataRad=GefShd)
                               )
      prod.i <- do.call(prodGCPV, listArgsShd)
      YfAnual[i]=mean(prod.i@prody$Yf)
      if (prog) {setTxtProgressBar(pb, i+1)}
    }
  }
  if (prog) {close(pb)}

###Results
FS=1-YfAnual/YfAnual0
GRR=switch(modeTrk,
            two=with(Red,Lew*Lns)/with(struct,L*W),
            fixed=Red$D/struct$L,
            horiz=Red$Lew/struct$L)
SombrADF=data.table(Red,GRR,FS,Yf=YfAnual)
FS.loess=switch(modeTrk,
                 two=loess(FS~Lew*Lns,data=SombrADF),
                 horiz=loess(FS~Lew,data=SombrADF),
                 fixed=loess(FS~D,data=SombrADF))
Yf.loess=switch(modeTrk,
                 two=loess(Yf~Lew*Lns,data=SombrADF),
                 horiz=loess(Yf~Lew,data=SombrADF),
                 fixed=loess(Yf~D,data=SombrADF))
result <- new('Shade',
              Prod0, ##contains ProdGCPV
              FS=FS,
              GRR=GRR,
              Yf=YfAnual,
              FS.loess=FS.loess,
              Yf.loess=Yf.loess,
              modeShd=modeShd,
              struct=struct,
              distances=Red,
              res=res
              )
  result
}

```

EXTRACTO DE CÓDIGO A.7: *optimShd*

A.1.8. meteoReaders

```

#### monthly means of irradiation ####
readG0dm <- function(G0dm, Ta = 25, lat = 0,
                     year = as.POSIXlt(Sys.Date())$year + 1900,
                     promDays = c(17, 14, 15, 15, 15, 10, 18, 18, 18, 19, 18,
                                   13),
                     source = '')
{
  if(missing(lat)){lat <- 0}

```

```

Dates <- as.IDate(paste(year, 1:12, promDays, sep = '-'), tz = 'UTC')
G0dm.dt <- data.table(Dates = Dates,
                      G0d = G0dm,
                      Ta = Ta)
setkey(G0dm.dt, 'Dates')
results <- new(Class = 'Meteo',
               latm = lat,
               data = G0dm.dt,
               type = 'prom',
               source = source)
}

#### file to Meteo (daily) ####
readBDd <- function(file, lat,
                    format = "%d/%m/%Y", header = TRUE,
                    fill = TRUE, dec = '.', sep = ';',
                    dates.col = 'Dates', ta.col = 'Ta',
                    g0.col = 'G0', keep.cols = FALSE)
{
  #stops if the arguments are not characters or numerics
  stopifnot(is.character(dates.col) || is.numeric(dates.col))
  stopifnot(is.character(ta.col) || is.numeric(ta.col))
  stopifnot(is.character(g0.col) || is.numeric(g0.col))

  #read from file and set it in a data.table
  bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep)

  #check the columns
  if(!(dates.col %in% names(bd))) stop(paste('The column', dates.col, 'is not
  in the file'))
  if(!(g0.col %in% names(bd))) stop(paste('The column', g0.col, 'is not in
  the file'))
  if(!(ta.col %in% names(bd))) stop(paste('The column', ta.col, 'is not in
  the file'))

  #name the dates column by Dates
  Dates <- bd[[dates.col]]
  bd[, (dates.col) := NULL]
  bd[, Dates := as.IDate(Dates, format = format)]

  #name the g0 column by G0
  G0 <- bd[[g0.col]]
  bd[, (g0.col) := NULL]
  bd[, G0 := as.numeric(G0)]

  #name the ta column by Ta
  Ta <- bd[[ta.col]]
  bd[, (ta.col) := NULL]
  bd[, Ta := as.numeric(Ta)]

  names0 <- NULL
  if(all(c('D0', 'B0') %in% names(bd))){
    names0 <- c(names0, 'D0', 'B0')
  }

  names0 <- c(names0, 'Ta')

  if(all(c('TempMin', 'TempMax') %in% names(bd))){

```

```

    names0 <- c(names0, 'TempMin', 'TempMax')
  }
  if(keep.cols)
  {
    #keep the rest of the columns but reorder the columns
    setcolorder(bd, c('Dates', 'G0', names0))
  }
  else
  {
    #erase the rest of the columns
    cols <- c('Dates', 'G0', names0)
    bd <- bd[, ..cols]
  }

  setkey(bd, 'Dates')
  result <- new(Class = 'Meteo',
                latm = lat,
                data = bd,
                type = 'bd',
                source = file)
}

#### file to Meteo (intradaily) ####
readBDi <- function(file, lat,
                    format = "%d/%m/%Y %H:%M:%S",
                    header = TRUE, fill = TRUE, dec = '.',
                    sep = ';', dates.col = 'dates', times.col,
                    ta.col = 'Ta', g0.col = 'G0', keep.cols = FALSE)
{
  #stops if the arguments are not characters or numerics
  stopifnot(is.character(dates.col) || is.numeric(dates.col))
  stopifnot(is.character(ta.col) || is.numeric(ta.col))
  stopifnot(is.character(g0.col) || is.numeric(g0.col))

  #read from file and set it in a data.table
  bd <- fread(file, header = header, fill = fill, dec = dec, sep = sep)

  #check the columns
  if(!(dates.col %in% names(bd))) stop(paste('The column', dates.col, 'is not
in the file'))
  if(!(g0.col %in% names(bd))) stop(paste('The column', g0.col, 'is not in
the file'))
  if(!(ta.col %in% names(bd))) stop(paste('The column', ta.col, 'is not in
the file'))

  if(!missing(times.col)){
    stopifnot(is.character(times.col) || is.numeric(times.col))
    if(!(times.col %in% names(bd))) stop(paste('The column', times.col, 'is
not in the file'))

    #name the dates column by Dates
    format <- strsplit(format, ' ')
    dd <- as.IDate(bd[[dates.col]], format = format[[1]][1])
    tt <- as.ITime(bd[[times.col]], format = format[[1]][2])
    bd[, (dates.col) := NULL]
    bd[, (times.col) := NULL]
    bd[, Dates := as.POSIXct(dd, tt, tz = 'UTC')]
  }
}

```

```

else
{
  dd <- as.POSIXct(bd[[dates.col]], format = format, tz = 'UTC')
  bd[, (dates.col) := NULL]
  bd[, Dates := dd]
}

#name the g0 column by GO
GO <- bd[[g0.col]]
bd[, (g0.col) := NULL]
bd[, GO := as.numeric(GO)]

#name the ta column by Ta
Ta <- bd[[ta.col]]
bd[, (ta.col) := NULL]
bd[, Ta := as.numeric(Ta)]

names0 <- NULL
if(all(c('D0', 'B0') %in% names(bd))){
  names0 <- c(names0, 'D0', 'B0')
}

names0 <- c(names0, 'Ta')

if(keep.cols)
{
  #keep the rest of the columns but reorder the columns
  setcolorder(bd, c('Dates', 'GO', names0))
}
else
{
  #erase the rest of the columns
  cols <- c('Dates', 'GO', names0)
  bd <- bd[, ..cols]
}

setkey(bd, 'Dates')
result <- new(Class = 'Meteo',
              latm = lat,
              data = bd,
              type = 'bdI',
              source = file)
}

dt2Meteo <- function(file, lat, source = '', type){
  ## Make sure its a data.table
  bd <- data.table(file)

  ## Dates is an as.POSIX element
  bd[, Dates := as.POSIXct(Dates, tz = 'UTC')]

  ## type
  if(missing(type)){
    sample <- median(diff(file$Dates))
    IsDaily <- as.numeric(sample, units = 'days')
    if(is.na(IsDaily)) IsDaily <- ifelse('G0d' %in% names(bd),

```

```

1, 0)

if(IsDaily >= 30) type <- 'prom'
else{
  type <- ifelse(IsDaily >= 1, 'bd', 'bdI')
}

}
if(!('Ta' %in% names(bd))){
  if(all(c('Tempmin', 'TempMax') %in% names(bd))){
    bd[, Ta := mean(c(Tempmin, TempMax))]
  } else bd[, Ta := 25]
}

## Columns of the data.table
nms0 <- switch(type,
  bd = ,
  prom = {
    nms0 <- 'G0d'
    if(all(c('D0d', 'B0d') %in% names(bd))){
      nms0 <- c(nms0, 'D0d', 'B0d')
    }
    nms0 <- c(nms0, 'Ta')
    if(all(c('TempMin', 'TempMax') %in% names(bd))){
      nms0 <- c(nms0, 'TempMin', 'TempMax')
    }
    nms0
  },
  bdI = {
    nms0 <- 'G0'
    if(all(c('D0', 'B0') %in% names(bd))){
      nms0 <- c(nms0, 'D0', 'B0')
    }
    if('Ta' %in% names(bd)){
      nms0 <- c(nms0, 'Ta')
    }
    nms0
  })

## Columns order and set key
setcolorder(bd, c('Dates', nms0))
setkey(bd, 'Dates')
## Result
result <- new(Class = 'Meteo',
  latm = lat,
  data = bd,
  type = type,
  source = source)
}

#### Liu and Jordan, Collares-Pereira and Rabl proposals ####
collper <- function(sol, compD)
{
  Dates <- indexI(sol)
  x <- as.Date(Dates)
  ind.rep <- cumsum(c(1, diff(x) != 0))
  solI <- as.data.tableI(sol, complete = T)
  ws <- solI$ws
  w <- solI$w

```

```

a <- 0.409-0.5016*sin(ws+pi/3)
b <- 0.6609+0.4767*sin(ws+pi/3)

rd <- solI[, Bo0/Bo0d]
rg <- rd * (a + b * cos(w))

# Daily irradiation components
G0d <- compD$G0d[ind.rep]
B0d <- compD$B0d[ind.rep]
D0d <- compD$D0d[ind.rep]

# Daily profile
G0 <- G0d * rg
D0 <- D0d * rd

# This method may produce diffuse irradiance higher than
# global irradiance
G0 <- pmax(G0, D0, na.rm = TRUE)
B0 <- G0 - D0

# Negative values are set to NA
neg <- (B0 < 0) | (D0 < 0) | (G0 < 0)
is.na(G0) <- neg
is.na(B0) <- neg
is.na(D0) <- neg

# Daily profiles are scaled to keep daily irradiation values
day <- truncDay(indexI(sol))
sample <- sol@sample

G0dCP <- ave(G0, day, FUN=function(x) P2E(x, sample))
B0dCP <- ave(B0, day, FUN=function(x) P2E(x, sample))
D0dCP <- ave(D0, day, FUN=function(x) P2E(x, sample))

G0 <- G0 * G0d/G0dCP
B0 <- B0 * B0d/B0dCP
D0 <- D0 * D0d/D0dCP

res <- data.table(G0, B0, D0)
return(res)
}

#### intradaily Meteo to daily Meteo ####
Meteoi2Meteod <- function(G0i)
{
  lat <- G0i@latm
  source <- G0i@source

  dt0 <- getData(G0i)
  dt <- dt0[, lapply(.SD, sum),
             .SDcols = names(dt0)[!names(dt0) %in% c('Dates', 'Ta')],
             by = .(Dates = as.IDate(Dates))]
  if('Ta' %in% names(dt0)){
    Ta <- dt0[, .(Ta = mean(Ta),
                      TempMin = min(Ta),
                      TempMax = max(Ta)),
              by = .(Dates = as.IDate(Dates))]
  }
}

```



```

    if(all(Ta$Ta == c(Ta$TempMin, Ta$TempMax))) Ta[, c('TempMin', 'TempMax')]
    := NULL]
    dt <- merge(dt, Ta)
  }
  if('G0' %in% names(dt)){
    names(dt)[names(dt) == 'G0'] <- 'G0d'
  }
  if('D0' %in% names(dt)){
    names(dt)[names(dt) == 'D0'] <- 'D0d'
  }
  if('B0' %in% names(dt)){
    names(dt)[names(dt) == 'B0'] <- 'B0d'
  }
  G0d <- dt2Meteo(dt, lat, source, type = 'bd')
  return(G0d)
}

#### daily Meteo to monthly Meteo ####
Meteod2Meteom <- function(G0d)
{
  lat <- G0d@latm
  source <- G0d@source

  dt <- getData(G0d)
  nms <- names(dt)[-1]
  dt <- dt[, lapply(.SD, mean),
    .SDcols = nms,
    by = .(month(Dates), year(Dates))]
  dt[, Dates := fBTd()]
  dt <- dt[, c('month', 'year') := NULL]

  setcolorder(dt, 'Dates')

  G0m <- dt2Meteo(dt, lat, source, type = 'prom')
  return(G0m)
}

zoo2Meteo <- function(file, lat, source = '')
{
  sample <- median(diff(index(file)))
  IsDaily <- as.numeric(sample, units = 'days')>=1
  type <- ifelse(IsDaily, 'bd', 'bdI')
  result <- new(Class = 'Meteo',
    latm = lat,
    data = file,
    type = type,
    source = source)
}

siarGET <- function(id, inicio, final, tipo = 'Mensuales', ambito = 'Estacion'){
  if(!(tipo %in% c('Horarios', 'Diarios', 'Semanales', 'Mensuales'))){
    stop('argument \'tipo\' must be: Horarios, Diarios, Semanales or
Mensuales')
  }
  if(!(ambito %in% c('CCAA', 'Provincia', 'Estacion'))){
    stop('argument \'ambito\' must be: CCAA, Provincia or Estacion')
  }
}

```

```

mainURL <- "https://servicio.mapama.gob.es"

path <- paste('/apisiar/API/v1/Datos', tipo, ambito, sep = '/')

## prepare the APISiar
req <- request(mainURL) |>
  req_url_path(path) |>
  req_url_query(Id = id,
                FechaInicial = inicio,
                FechaFinal = final,
                ClaveAPI = '_Q8L_niYFBBmBs-
vB3UomUqdUYy98FTRX1aYbrZ8n2FXuHYGTV')
## execute it
resp <- req_perform(req)

##JSON to R
respJSON <- resp_body_json(resp, simplifyVector = TRUE)

if(!is.null(respJSON$MensajeRespuesta)){
  stop(respJSON$MensajeRespuesta)
}

res0 <- data.table(respJSON$Datos)

res <- switch(tipo,
  Horarios = {
    res0[, HoraMin := as.ITime(sprintf('%04d', HoraMin),
                                   format = '%H%M')]
    res0[, Fecha := as.IDate(Fecha, format = '%Y-%m-%d')]
    res0[, Fecha := as.IDate(ifelse(HoraMin == as.ITime(0),
                                   Fecha+1, Fecha))]
    res0[, Dates := as.POSIXct(HoraMin, Fecha,
                               tz = 'Europe/Madrid')]
    res0 <- res0[, .(Dates,
                     GO = Radiacion,
                     Ta = TempMedia)]
    return(res0)
  },
  Diarios = {
    res0[, Dates := as.IDate(Fecha)]
    res0 <- res0[, .(Dates,
                     GOd = Radiacion * 277.78,
                     Ta = TempMedia,
                     TempMin,
                     TempMax)]
    return(res0)
  },
  Semanales = res0,
  Mensuales = {
    promDays<-c(17,14,15,15,15,10,18,18,18,19,18,13)
    names(res0)[1] <- 'Year'
    res0[, Dates := as.IDate(paste(Year, Mes,
                                   promDays[Mes],
                                   sep = '-'))]
    res0 <- res0[, .(Dates,
                     GOd = Radiacion * 277.78,
                     Ta = TempMedia,
                     TempMin,

```

```

                                TempMax]]
                                })

    return(res)
}

haversine <- function(lat1, lon1, lat2, lon2) {
  R <- 6371 # Radius of the Earth in kilometers
  dLat <- (lat2 - lat1) * pi / 180
  dLon <- (lon2 - lon1) * pi / 180
  a <- sin(dLat / 2) * sin(dLat / 2) + cos(lat1 * pi / 180) *
    cos(lat2 * pi / 180) * sin(dLon / 2) * sin(dLon / 2)
  c <- 2 * atan2(sqrt(a), sqrt(1 - a))
  d <- R * c
  return(d)
}

readSIAR <- function(Lon = 0, Lat = 0,
                     inicio = paste(year(Sys.Date())-1, '01-01', sep = '-'),
                     final = paste(year(Sys.Date())-1, '12-31', sep = '-'),
                     tipo = 'Mensuales', n_est = 3){
  inicio <- as.Date(inicio)
  final <- as.Date(final)

  n_reg <- switch(tipo,
    Horarios = {
      tt <- difftime(final, inicio, units = 'days')
      tt <- (as.numeric(tt)+1)*48
      tt <- tt*n_est
      tt
    },
    Diarios = {
      tt <- difftime(final, inicio, units = 'days')
      tt <- as.numeric(tt)+1
      tt <- tt*n_est
      tt
    },
    Semanales = {
      tt <- difftime(final, inicio, units = 'weeks')
      tt <- as.numeric(tt)
      tt <- tt*n_est
      tt
    },
    Mensuales = {
      tt <- difftime(final, inicio, units = 'weeks')
      tt <- as.numeric(tt)/4.34524
      tt <- ceiling(tt)
      tt <- tt*n_est
      tt
    }
  )
  if(n_reg > 100) stop(paste('Number of requested records (', n_reg,
                             ') exceeds the maximum allowed (100)', sep = ''))
  )
  ## Obtain the nearest stations
  siar <- est_SIAR[
    Fecha_Instalacion <= final & (is.na(Fecha_Baja) | Fecha_Baja >= inicio)
  ]
}

```

```

## Weights for the interpolation
siar[, dist := haversine(Latitud, Longitud, Lat, Lon)]
siar <- siar[order(dist)][1:n_est]
siar[, peso := 1/dist]
siar[, peso := peso/sum(peso)]
## Is the given location within the polygon formed by the stations?
siar <- siar[, .(Estacion,Codigo, dist, peso)]

## List for the data.tables of siarGET
siar_list <- list()
for(codigo in siar$Codigo){
  siar_list[[codigo]] <- siarGET(id = codigo,
                                inicio = as.character(inicio),
                                final = as.character(final),
                                tipo = tipo)
  siar_list[[codigo]]$peso <- siar[Codigo == codigo, peso]
}

## Bind the data.tables
s_comb <- rbindlist(siar_list, use.names = TRUE, fill = TRUE)

nms <- names(s_comb)
nms <- nms[-c(1, length(nms))]

## Interpole
res <- s_comb[, lapply(.SD * peso, sum, na.rm = TRUE),
               .SDcols = nms,
               by = Dates]

## Source
mainURL <- "https://servicio.mapama.gob.es"
Estaciones <- siar[, paste(Estacion, '(', Codigo, ')', sep = '')]
Estaciones <- paste(Estaciones, collapse = ', ')
source <- paste(mainURL, '\n -Estaciones:', Estaciones, sep = ' ')

res <- switch(tipo,
              Horarios = {dt2Meteo(res, lat = Lat, source = mainURL, type =
'bdI')},
              Diarios = {dt2Meteo(res, lat = Lat, source = mainURL, type =
'bd')},
              Semanales = {res},
              Mensuales = {dt2Meteo(res, lat = Lat, source = source, type =
'prom')})
return(res)
}

```

EXTRACTO DE CÓDIGO A.8: *meteoReaders*

A.2. Clases

A.2.1. Sol

```

setClass(
  Class='Sol', ##Solar angles
  slots = c(
    lat='numeric',#latitud in degrees, >0 if North
    sold='data.table',#daily angles
    soli='data.table',#intradaily angles

```

```

        sample='character',#sample of time
        method='character'#method used for geometry calculations
    ),
    validity=function(object) {return(TRUE)}
)

```

EXTRACTO DE CÓDIGO A.9: Clase *Sol*

A.2.2. Meteo

```

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

```

EXTRACTO DE CÓDIGO A.10: Clase *Meteo*

A.2.3. G0

```

setClass(
  Class = 'G0',
  slots = c(
    GOD = 'data.table', #result of fCompD
    Godm = 'data.table', #monthly means
    GOy = 'data.table', #yearly values
    GOI = 'data.table', #result of fCompI
    Ta = 'data.table' #Ambient temperature
  ),
  contains = c('Sol', 'Meteo'),
  validity = function(object) {return(TRUE)}
)

```

EXTRACTO DE CÓDIGO A.11: Clase *G0*

A.2.4. Gef

```

setClass(
  Class='Gef',
  slots = c(
    GefD='data.table', #daily values
    Gefdm='data.table', #monthly means
    Gefy='data.table', #yearly values
    GefI='data.table', #result of fInclin
    Theta='data.table', #result of fTheta
    iS='numeric', #dirt index
    alb='numeric', #albedo
    modeTrk='character', #tracking mode
    modeShd='character', #shadow mode
    angGen='list', #includes alpha, beta and betaLim
    struct='list', #structure dimensions
    distances='data.frame' #distances between structures
  )
)

```

```
    ),  
    contains='GO',  
    validity=function(object) {return(TRUE)}  
)
```

EXTRACTO DE CÓDIGO A.12: Clase *Gef*

A.2.5. ProdGCPV

```
setClass(  
  Class='ProdGCPV',  
  slots = c(  
    prodD='data.table', #daily values  
    prodDm='data.table', #monthly means  
    prody='data.table', #yearly values  
    prodI='data.table', #results of fProd  
    module='list', #module characteristics  
    generator='list', #generator characteristics  
    inverter='list', #inverter characteristics  
    effSys='list' #efficiency values of the system  
  ),  
  contains='Gef',  
  validity=function(object) {return(TRUE)}  
)
```

EXTRACTO DE CÓDIGO A.13: Clase *ProdGCPV*

A.2.6. ProdPVPS

```
setClass(  
  Class='ProdPVPS',  
  slots = c(  
    prodD='data.table', #daily values  
    prodDm='data.table', #monthly means  
    prody='data.table', #yearly values  
    prodI='data.table', #results of fPump  
    Pg='numeric', #generator power  
    H='numeric', #manometric head  
    pump='list', #parameters of the pump  
    converter='list', #inverter characteristics  
    effSys='list' #efficiency values of the system  
  ),  
  contains='Gef',  
  validity=function(object) {return(TRUE)}  
)
```

EXTRACTO DE CÓDIGO A.14: Clase *ProdPVPS*

A.2.7. Shade

```
setClass(  
  Class='Shade',  
  slots = c(  
    FS='numeric', #shadows factor values  
    GRR='numeric', #Ground Requirement Ratio  
    Yf='numeric', #final productivity  
    FS.loess='loess', #local fitting of FS with loess  
    Yf.loess='loess', #local fitting of Yf with loess  
  )  
)
```

```

        modeShd='character', #mode of shadow
        struct='list',      #dimensions of the structures
        distances='data.frame', #distances between structures
        res='numeric'       #difference between the different steps of
the calculations
    ),
    contains='ProdGCPV',##Resultado de prodGCPV sin sombras (Prod0)
    validity=function(object) {return(TRUE)}
)

```

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

A.3. Funciones

A.3.1. corrFdKt

```

#### monthly Kt ####
Ktm <- function(sol, G0dm){
  solf <- sol@solD[, .(Dates, Bo0d)]
  solf[, c('month', 'year') := .(month(Dates), year(Dates))]
  solf[, Bo0m := mean(Bo0d), by = .(month, year)]
  G0df <- G0dm@data[, .(Dates, G0d)]
  G0df[, c('month', 'year') := .(month(Dates), year(Dates))]
  G0df[, G0d := mean(G0d), by = .(month, year)]
  Ktm <- G0df$G0d/solf$Bo0m
  return(Ktm)
}

#### daily Kt ####
Ktd <- function(sol, G0d){
  Bo0d <- sol@solD$Bo0d
  G0d <- getG0(G0d)
  Ktd <- G0d/Bo0d
  return(Ktd)
}

### intradaily
Kti <- function(sol, G0i){
  Bo0 <- sol@solI$Bo0
  G0i <- getG0(G0i)
  Kti <- G0i/Bo0
  return(Kti)
}

#### monthly correlations ####

### Page ###
FdKtPage <- function(sol, G0dm){
  Kt <- Ktm(sol, G0dm)
  Fd=1-1.13*Kt
  return(data.table(Fd, Kt))
}

### Liu and Jordan ###
FdKtLJ <- function(sol, G0dm){
  Kt <- Ktm(sol, G0dm)
  Fd=(Kt<0.3)*0.595774 +

```

```

        (Kt>=0.3 & Kt<=0.7)*(1.39-4.027*Kt+5.531*Kt^2-3.108*Kt^3)+
        (Kt>0.7)*0.215246
    return(data.table(Fd, Kt))
}

#### daily correlations ####

### Collares-Pereira and Rabl
FdKtCPR <- function(sol, G0d){
    Kt <- Ktd(sol, G0d)
    Fd=(0.99*(Kt<=0.17))+(Kt>0.17 & Kt<0.8)*
        (1.188-2.272*Kt+9.473*Kt^2-21.856*Kt^3+14.648*Kt^4)+
        (Kt>=0.8)*0.2426688
    return(data.table(Fd, Kt))
}

### Erbs, Klein and Duffie ###
FdKtEKDd <- function(sol, G0d){
    ws <- sol@sold$ws
    Kt <- Ktd(sol, G0d)

    WS1=(abs(ws)<1.4208)
    Fd=WS1*((Kt<0.715)*(1-0.2727*Kt+2.4495*Kt^2-11.9514*Kt^3+9.3879*Kt^4)+
        (Kt>=0.715)*(0.143))+
        !WS1*((Kt<0.722)*(1+0.2832*Kt-2.5557*Kt^2+0.8448*Kt^3)+
        (Kt>=0.722)*(0.175))
    return(data.table(Fd, Kt))
}

### CLIMED1 ###
FdKtCLIMEDd <- function(sol, G0d){
    Kt <- Ktd(sol, G0d)
    Fd=(Kt<=0.13)*(0.952)+
        (Kt>0.13 & Kt<=0.8)*(0.868+1.335*Kt-5.782*Kt^2+3.721*Kt^3)+
        (Kt>0.8)*0.141
    return(data.table(Fd, Kt))
}

#### intradaily correlations ####

### intradaily EKD ###
FdKtEKDh <- function(sol, G0i){
    Kt <- Kti(sol, G0i)
    Fd=(Kt<=0.22)*(1-0.09*Kt)+
        (Kt>0.22 & Kt<=0.8)*(0.9511-0.1604*Kt+4.388*Kt^2-16.638*Kt^3+12.336*Kt^4)+
        (Kt>0.8)*0.165
    return(data.table(Fd, Kt))
}

### intradaily CLIMED
FdKtCLIMEDh <- function(sol, G0i){
    Kt <- Kti(sol, G0i)
    Fd=(Kt<=0.21)*(0.995-0.081*Kt)+
        (Kt>0.21 & Kt<=0.76)*(0.724+2.738*Kt-8.32*Kt^2+4.967*Kt^3)+
        (Kt>0.76)*0.180
    return(data.table(Fd, Kt))
}

```



```

### intradaily Boland, Ridley and Lauret ###
FdKtBRL <- function(sol, GOi){
  Kt <- Kti(sol, GOi)
  sample <- sol@sample

  solI <- as.data.tableI(sol, complete = TRUE)
  w <- solI$w
  night <- solI$night
  AlS <- solI$AlS

  GOd <- Meteoi2Meteod(GOi)
  ktd <- Ktd(sol, GOd)

  ##persistence
  pers <- persistence(sol, ktd)

  ##indexRep for ktd and pers
  Dates <- indexI(sol)
  x <- as.Date(Dates)
  ind.rep <- cumsum(c(1, diff(x) != 0))
  ktd <- ktd[ind.rep]
  pers <- pers[ind.rep]

  ##fd calculation
  Fd=(1+exp(-5.38+6.63*Kt+0.006*r2h(w)-0.007*r2d(AlS)+1.75*ktd+1.31*pers))
  ^(-1)

  return(data.table(Fd, Kt))
}

persistence <- function(sol, Ktd){
  kt <- data.table(indexD(sol), Ktd)
  ktNA <- na.omit(kt)
  iDay <- truncDay(ktNA[[1]])

  x <- rle(as.numeric(iDay))$lengths
  xLast <- cumsum(x)

  lag1 <- shift(ktNA$Ktd, -1, fill = NA)
  for (i in xLast){
    if ((i-1) != 0){lag1[i] <- ktNA$Ktd[i-1]}
  }

  lag2 <- shift(ktNA$Ktd, 1, fill = NA)
  for (i in xLast){
    if ((i+1) <= length(ktNA$Ktd)){lag2[i] <- ktNA$Ktd[i+1]}
  }
  pers <- data.table(lag1, lag2)
  pers[, mean := 1/2 * (lag1+lag2)]
  pers[, mean]
}

```

EXTRACTO DE CÓDIGO A.16: *corrFdKt*

A.3.2. fBTd

```
fBTd<-function(mode='prom',
```

```

        year= as.POSIXlt(Sys.Date())$year+1900,
        start=paste('01-01-',year,sep=''),
        end=paste('31-12-',year,sep=''),
        format='%d-%m-%Y'){
promDays<-c(17,14,15,15,15,10,18,18,18,19,18,13)
BTd=switch(mode,
  serie={
    start.<-as.POSIXct(start, format=format, tz='UTC')
    end.<-as.POSIXct(end, format=format, tz='UTC')
    res<-seq(start., end., by="1 day")
  },
  prom=as.POSIXct(paste(year, 1:12, promDays, sep='-'), tz='UTC')
)
BTd
}

```

EXTRACTO DE CÓDIGO A.17: *fBTd*A.3.3. *fBTi*

```

intervalo <- function(day, sample){
  intervalo <- seq.POSIXt(from = as.POSIXct(paste(day, '00:00:00'), tz = 'UTC'
),
                        to = as.POSIXct(paste(day, '23:59:59'), tz = 'UTC'),
                        by = sample)
  return(intervalo)
}

fBTi <- function(d, sample = 'hour'){
  BTi <- lapply(d, intervalo, sample)
  BTi <- do.call(c, BTi)
  return(BTi)
}

```

EXTRACTO DE CÓDIGO A.18: *fBTi*A.3.4. *fCompD*

```

fCompD <- function(sol, G0d, corr = 'CPR', f)
{
  if(!(corr %in% c('CPR', 'Page', 'LJ', 'EKDd', 'CLIMEDd', 'user', 'none'))){
    warning('Wrong descriptor of correlation Fd-Ktd. Set CPR.')
    corr <- 'CPR'
  }
  if(class(sol)[1] != 'Sol'){
    sol <- sol[, calcSol(lat = unique(lat), BTi = Dates)]
  }
  if(class(G0d)[1] != 'Meteo'){
    dt <- copy(data.table(G0d))
    if(!('Dates' %in% names(dt))){
      dt[, Dates := indexD(sol)]
      setcolorder(dt, 'Dates')
      setkey(dt, 'Dates')
    }
    if('lat' %in% names(dt)){
      latg <- unique(dt$lat)
      dt[, lat := NULL]
    }else{latg <- getLat(sol)}
  }
}

```

```

    G0d <- dt2Meteo(dt, latg)
  }

  stopifnot(indexD(sol) == indexD(G0d))
  Bo0d <- sol@solD$Bo0d
  G0 <- getData(G0d)$G0

  is.na(G0) <- (G0>Bo0d)

  ### the Direct and Difuse data is not given
  if(corr != 'none'){
    Fd <- switch(corr,
      CPR = FdKtCPR(sol, G0d),
      Page = FdKtPage(sol, G0d),
      LJ = FdKtLJ(sol, G0d),
      CLIMEDd = FdKtCLIMEDd(sol, G0d),
      user = f(sol, G0d))

    Kt <- Fd$Kt
    Fd <- Fd$Fd
    D0d <- Fd * G0
    B0d <- G0 - D0d
  }
  ### the Direct and Difuse data is given
  else {
    G0 <- getData(G0d)$G0
    D0d <- getData(G0d)[['D0']]
    B0d <- getData(G0d)[['B0']]
    Fd <- D0d/G0
    Kt <- G0/Bo0d
  }

  result <- data.table(Dates = indexD(sol), Fd, Kt, G0d = G0, D0d, B0d)
  setkey(result, 'Dates')
  result
}

```

EXTRACTO DE CÓDIGO A.19: *fCompD*

A.3.5. fCompI

```

fCompI <- function(sol, compD, GOI,
  corr = 'EKDh', f,
  filterGO = TRUE){
  if(!(corr %in% c('EKDh', 'CLIMEDh', 'BRL', 'user', 'none'))){
    warning('Wrong descriptor of correlation Fd-Ktd. Set EKDh.')
    corr <- 'EKDh'
  }

  if(class(sol)[1] != 'Sol'){
    sol <- sol[, calcSol(lat = unique(lat), BTi = Dates)]
  }

  lat <- sol@lat
  sample <- sol@sample
  night <- sol@solI$night
  Bo0 <- sol@solI$Bo0
  Dates <- indexI(sol)

```

```

## If instantaneous values are not provided, compD is used instead.
if (missing(GOI)) {

  GOI <- collper(sol, compD)
  GO <- GOI$G0
  B0 <- GOI$B0
  D0 <- GOI$D0

  Fd <- D0/G0
  Kt <- G0/Bo0

} else { ## Use instantaneous values if provided through GOI

  if(class(GOI)[1] != 'Meteo'){
    dt <- copy(GOI)
    if(!('Dates' %in% names(GOI))){
      dt[, Dates := indexI(sol)]
      setcolorder(dt, 'Dates')
      setkey(dt, 'Dates')
    }
    if('lat' %in% names(GOI)){latg <- unique(GOI$lat)}
    else{latg <- lat}
    GOI <- dt2Meteo(dt, latg)
  }

  if (corr!='none'){
    GO <- getG0(GOI)
    ## Filter values: surface irradiation must be lower than
    ## extraterrestrial;
    if (filterG0) {is.na(GO) <- (GO > Bo0)}

    ## Fd-Kt correlation
    Fd <- switch(corr,
      EKDh = FdKtEKDh(sol, GOI),
      CLIMEDh = FdKtCLIMEDh(sol, GOI),
      BRL = FdKtBRL(sol, GOI),
      user = f(sol, GOI))

    Kt <- Fd$Kt
    Fd <- Fd$Fd
    D0 <- Fd * GO
    B0 <- G0 - D0

  } else {
    GO <- getG0(GOI)
    D0 <- getData(GOI)[['D0']]
    B0 <- getData(GOI)[['B0']]
    ## Filter values: surface irradiation must be lower than
    ## extraterrestrial;
    if (isTRUE(filterG0)) is.na(GO) <- is.na(D0) <- is.na(B0) <- (GO >
Bo0)

    Fd <- D0/G0
    Kt <- G0/Bo0
  }
}

## Values outside sunrise-sunset are set to zero
GO[night] <- D0[night] <- B0[night] <- Kt[night] <- Fd[night] <- 0

```

```

result <- data.table(Dates, Fd, Kt, G0, D0, B0)
setkey(result, 'Dates')
result
}

```

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

```

fInclin <- function(compI, angGen, iS = 2, alb = 0.2, horizBright = TRUE, HCPV =
  FALSE){
  ##compI es class='G0'

  ##Arguments
  stopifnot(iS %in% 1:4)
  Beta <- angGen$Beta
  Alfa <- angGen$Alfa
  cosTheta <- angGen$cosTheta

  comp <- as.data.tableI(compI, complete=TRUE)
  night <- comp$night
  B0 <- comp$B0
  Bo0 <- comp$Bo0
  D0 <- comp$D0
  G0 <- comp$G0
  cosThzS <- comp$cosThzS
  is.na(cosThzS) <- night

  ##N.Martin method for dirt and non-perpendicular incidence
  Suc <- rbind(c(1, 0.17, -0.069),
               c(0.98,.2,-0.054),
               c(0.97,0.21,-0.049),
               c(0.92,0.27,-0.023))
  FTb <- (exp(-cosTheta/Suc[iS,2]) - exp(-1/Suc[iS,2]))/(1 - exp(-1/Suc[iS,2]))
  )
  FTd <- exp(-1/Suc[iS,2] * (4/(3*pi) * (sin(Beta) + (pi - Beta - sin(Beta))/
    (1 + cos(Beta))) +
          Suc[iS,3] * (sin(Beta) + (pi - Beta - sin(Beta))/
    (1 + cos(Beta)))^2))
  FTr <- exp(-1/Suc[iS,2] * (4/(3*pi) * (sin(Beta) + (Beta - sin(Beta))/(1 -
    cos(Beta))) +
          Suc[iS,3] * (sin(Beta) + (Beta - sin(Beta))/(1 -
    cos(Beta)))^2))

  ##Hay and Davies method for diffuse treatment
  B <- B0 * cosTheta/cosThzS * (cosThzS>0.007) #The factor cosThzS>0.007 is
  needed to eliminate erroneous results near dawn
  k1 <- B0/(Bo0)
  Di <- D0 * (1-k1) * (1+cos(Beta))/2
  if (horizBright) Di <- Di * (1+sqrt(B0/G0) * sin(Beta/2)^3)
  Dc <- D0 * k1 * cosTheta/cosThzS * (cosThzS>0.007)
  R <- alb * G0 * (1-cos(Beta))/2
  D <- (Di + Dc)
  ##Extraterrestrial irradiance on the inclined plane
  Bo <- Bo0 * cosTheta/cosThzS * (cosThzS>0.007)
  ##Normal direct irradiance (DNI)
  Bn <- B0/cosThzS

```

```

##Sum of components
G <- B + D + R
Ref <- R * Suc[iS,1] * (1-FTr) * (!HCPV)
Ref[is.nan(FTr)] <- 0 #When cos(Beta)=1, FTr=NaN. Cancel Ref.
Dief <- Di * Suc[iS,1] * (1 - FTd) * (!HCPV)
Dcef <- Dc * Suc[iS,1] * (1 - FTb) * (!HCPV)
Def <- Dief + Dcef
Bef <- B * Suc[iS,1] * (1 - FTb)
Gef <- Bef + Def + Ref

result <- data.table(Bo, Bn,
                    G, D, Di, Dc, B, R,
                    FTb, FTd, FTr,
                    Dief, Dcef, Gef, Def, Bef, Ref)

## Use 0 instead of NA for irradiance values
result[night] <- 0
result[, Dates := indexI(compI)]
result[, .SD, by = Dates]
setcolorder(result, c('Dates', names(result)[-length(result)]))
result
}

```

EXTRACTO DE CÓDIGO A.21: *fInclin*A.3.7. *fProd*

```

## voc, iscn, vmpp, impp : *cell* values
## Voc, Isc, Vmpp, Imp: *module/generator* values

## Compute Current - Voltage characteristic of a solar *cell* with Gef
## and Ta
iv <- function(vocn, iscn, vmn, imn,
              TONC, CoefVT = 2.3e-3,
              Ta, Gef,
              vmin = NULL, vmax = NULL)
{
  ##Cell Constants
  Gstc <- 1000
  Ct <- (TONC - 20) / 800
  Vtn <- 0.025 * (273 + 25) / 300
  m <- 1.3

  ##Cell temperature
  Tc <- Ta + Ct * Gef
  Vt <- 0.025 * (Tc + 273)/300

  ## Series resistance
  Rs <- (vocn - vmn + m * Vtn * log(1 - imn/iscn)) / imn

  ## Voc and Isc at ambient conditions
  voc <- vocn - CoefVT * (Tc - 25)
  isc <- iscn * Gef/Gstc

  ## Ruiz method for computing voltage and current characteristic of a *cell*
  rs <- Rs * isc/voc
  koc <- voc/(m * Vt)
}

```

```

## Maximum Power Point
Dm0 <- (koc - 1)/(koc - log(koc))
Dm <- Dm0 + 2 * rs * Dm0^2

impp <- isc * (1 - Dm/koc)
vmpp <- voc * (1 - log(koc/Dm)/koc - rs * (1 - Dm/koc))

vdc <- vmpp
idc <- impp

## When the MPP is below/above the inverter voltage limits, it
## sets the voltage point at the corresponding limit.

## Auxiliary functions for computing the current at a defined
## voltage.
ilimit <- function(v, koc, rs)
{
  if (is.na(koc))
    result <- NA
  else
  {
    ## The IV characteristic is an implicit equation. The starting
    ## point is the voltage of the cell (imposed by the inverter
    ## limit).

    izero <- function(i, v, koc, rs)
    {
      vp <- v + i * rs
      Is <- 1/(1 - exp(-koc * (1 - rs)))
      result <- i - (1 - Is * (exp(-koc * (1 - vp)) - exp(-koc * (1 -
rs))))
    }

    result <- uniroot(f = izero,
                      interval = c(0,1),
                      v = v,
                      koc = koc,
                      rs = rs)$root
  }
  result
}

## Inverter minimum voltage
if (!is.null(vmin))
{
  if (any(vmpp < vmin, na.rm = TRUE))
  {
    indMIN <- which(vmpp < vmin)
    imin <- sapply(indMIN, function(i)
    {
      vocMIN <- voc[i]
      kocMIN <- koc[i]
      rsMIN <- rs[i]
      vmin <- vmin/vocMIN
      ##v debe estar entre 0 y 1
      vmin[vmin < 0] <- 0
      vmin[vmin > 1] <- 1
      ilimit(vmin, kocMIN, rsMIN)
    })
  }
}

```

```

    })
    iscMIN <- isc[indMIN]
    idc[indMIN] <- imin * iscMIN
    vdc[indMIN] <- vmin
    warning('Minimum MPP voltage of the inverter has been reached')}
}

if (!is.null(vmax))
{
  if (any(vmpp > vmax, na.rm = TRUE))
  {
    indMAX <- which(vmpp > vmax)
    imax <- sapply(indMAX, function(i)
    {
      vocMAX <- voc[i]
      kocMAX <- koc[i]
      rsMAX <- rs[i]
      vmax <- vmax / vocMAX
      ##v debe estar entre 0 y 1
      vmax[vmax < 0] <- 0
      vmax[vmax > 1] <- 1
      ilimit(vmax, kocMAX, rsMAX)
    })
    iscMAX <- isc[indMAX]
    idc[indMAX] <- imax * iscMAX
    vdc[indMAX] <- vmax
    warning('Maximum MPP voltage of the inverter has been reached')
  }
}
data.table(Ta, Tc, Gef, voc, isc, vmpp, impp, vdc, idc)
}

fProd <- function(inclin,
  module=list(),
  generator=list(),
  inverter=list(),
  effSys=list()
)
{
  stopifnot(is.list(module),
    is.list(generator),
    is.list(inverter),
    is.list(effSys)
  )
  ## Extract data from objects
  if (class(inclin)[1]=='Gef') {
    indInclin <- indexI(inclin)
    gefI <- as.data.tableI(inclin, complete = TRUE)
    Gef <- gefI$Gef
    Ta <- gefI$Ta
  } else {
    Gef <- inclin$Gef
    Ta <- inclin$Ta
  }

  ## Module, generator, and inverter parameters
  module.default <- list(Vocn = 57.6,

```



```

        Iscn = 4.7,
        Vmn = 46.08,
        Imn = 4.35,
        Ncs = 96,
        Ncp = 1,
        CoefVT = 0.0023,
        TONC = 47)
module <- modifyList(module.default, module)
## Make these parameters visible because they will be used often.
Ncs <- module$Ncs
Ncp <- module$Ncp

generator.default <- list(Nms = 12,
                          Nmp = 11)
generator <- modifyList(generator.default, generator)
generator$Pg <- (module$Vmn * generator$Nms) *
               (module$Imn * generator$Nmp)
Nms <- generator$Nms
Nmp <- generator$Nmp

inverter.default <- list(Ki = c(0.01,0.025,0.05),
                        Pinv = 25000,
                        Vmin = 420,
                        Vmax = 750,
                        Gumb = 20)
inverter <- modifyList(inverter.default, inverter)
Pinv <- inverter$Pinv

effSys.default <- list(ModQual = 3,
                      ModDisp = 2,
                      OhmDC = 1.5,
                      OhmAC = 1.5,
                      MPP = 1,
                      TrafoMT = 1,
                      Disp = 0.5)
effSys <- modifyList(effSys.default, effSys)

## Solar Cell i-v
vocn <- with(module, Vocn / Ncs)
iscn <- with(module, Iscn / Ncp)
vmn <- with(module, Vmn / Ncs)
imn <- with(module, Imn / Ncp)
vmin <- with(inverter, Vmin / (Ncs * Nms))
vmax <- with(inverter, Vmax / (Ncs * Nms))

cell <- iv(vocn, iscn,
          vmn, imn,
          module$TONC, module$CoefVT,
          Ta, Gef,
          vmin, vmax)

## Generator voltage and current
Idc <- Nmp * Ncp * cell$idc
Isc <- Nmp * Ncp * cell$isc
Impp <- Nmp * Ncp * cell$impp
Vdc <- Nms * Ncs * cell$vdc
Voc <- Nms * Ncs * cell$voc
Vmpp <- Nms * Ncs * cell$vmpp

```

```

##DC power (normalization with nominal power of inverter)
##including losses
PdcN <- with(effSys, (Idc * Vdc) / Pinv *
                  (1 - ModQual / 100) *
                  (1 - ModDisp / 100) *
                  (1 - MPP / 100) *
                  (1 - OhmDC / 100)
                )

##Normalized AC power to the inverter
Ki <- inverter$Ki
if (is.matrix(Ki)) { #Ki is a matrix of nine coefficients-->dependence with
tension
  VP <- cbind(Vdc, PdcN)
  PacN <- apply(VP, 1, solvePac, Ki)
} else { #Ki is a vector of three coefficients-->without dependence on
voltage
  A <- Ki[3]
  B <- Ki[2] + 1
  C <- Ki[1] - (PdcN)
  PacN <- (-B + sqrt(B^2 - 4 * A * C))/(2 * A)
}
EffI <- PacN / PdcN
pacNeg <- PacN <= 0
PacN[pacNeg] <- PdcN[pacNeg] <- EffI[pacNeg] <- 0

##AC and DC power without normalization
Pac <- with(effSys, PacN * Pinv *
            (Gef > inverter$Gumb) *
            (1 - OhmAC / 100) *
            (1 - TrafoMT / 100) *
            (1 - Disp / 100))
Pdc <- PdcN * Pinv * (Pac > 0)

## Result
resProd <- data.table(Tc = cell$Tc,
                      Voc, Isc,
                      Vmpp, Impp,
                      Vdc, Idc,
                      Pac, Pdc,
                      EffI)
if (class(inclin)[1] %in% 'Gef'){
  result <- resProd[, .SD,
                    by=.(Dates = indInclin)]
  attr(result, 'generator') <- generator
  attr(result, 'module') <- module
  attr(result, 'inverter') <- inverter
  attr(result, 'effSys') <- effSys
  return(result)
} else {
  result <- cbind(inclin, resProd)
  return(result)
}
}

```

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

```

fPump <- function(pump, H){

  w1=3000 ##synchronous rpm frequency
  wm=2870 ##rpm frequency with slip when applying voltage at 50 Hz
  s=(w1-wm)/w1
  fen=50 ##Nominal electrical frequency
  fmin=sqrt(H/pump$a)
  fmax=with(pump, (-b*Qmax+sqrt(b^2*Qmax^2-4*a*(c*Qmax^2-H)))/(2*a))
  ##fb is rotation frequency (Hz) of the pump,
  ##fe is the electrical frequency applied to the motor
  ##which makes it rotate at a frequency fb (and therefore also the pump).
  fb=seq(fmin,min(60,fmax),length=1000) #The maximum frequency is 60
  fe=fb/(1-s)

  ###Flow
  Q=with(pump, (-b*fb-sqrt(b^2*fb^2-4*c*(a*fb^2-H)))/(2*c))
  Qmin=0.1*pump$Qn*fb/50
  Q=Q+(Qmin-Q)*(Q<Qmin)

  ###Hydraulic power
  Ph=2.725*Q*H

  ###Mechanical power
  Q50=50*Q/fb
  H50=H*(50/fb)^2
  etab=with(pump, j*Q50^2+k*Q50+1)
  Pb50=2.725*H50*Q50/etab
  Pb=Pb50*(fb/50)^3

  ###Electrical power
  Pbc=Pb*50/fe
  etam=with(pump, g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
  Pmc=Pbc/etam
  Pm=Pmc*fe/50
  Pac=Pm
  ##Pdc=Pm/(etac*(1-cab))

  ###I build functions for flow, frequency and powers
  ###to adjust the AC power.
  fQ<-splinefun(Pac,Q)
  fFreq<-splinefun(Pac,fe)
  fPb<-splinefun(Pac,Pb)
  fPh<-splinefun(Pac,Ph)
  lim=c(min(Pac),max(Pac))
  ##lim marks the operating range of the pump
  result<-list(lim = lim,
               fQ = fQ,
               fPb = fPb,
               fPh = fPh,
               fFreq = fFreq)
}

```

EXTRACTO DE CÓDIGO A.23: *fPump*

A.3.9. fSolD

```
fSolD <- function(lat, BTd, method = 'michalsky'){
  if (abs(lat) > 90){
    lat <- sign(lat) * 90
    warning(paste('Latitude outside acceptable values. Set to', lat))
  }
  sun <- data.table(Dates = unique(as.IDate(BTd)),
                    lat = lat)

  ##### solarAngles #####

  ##Declination
  sun[, decl := declination(Dates, method = method)]
  ##Eccentricity
  sun[, eo := eccentricity(Dates, method = method)]
  ##Equation of time
  sun[, EoT := eot(Dates)]
  ##Solar time
  sun[, ws := sunrise(Dates, lat, method = method,
                      decl = decl)]
  ##Extraterrestrial irradiance
  sun[, Bo0d := bo0d(Dates, lat, method = method,
                     decl = decl,
                     eo = eo,
                     ws = ws
                     )]

  setkey(sun, Dates)
  return(sun)
}
```

EXTRACTO DE CÓDIGO A.24: *fSolD*

A.3.10. fSolI

```
fSolI <- function(sold, sample = 'hour', BTi,
                  EoT = TRUE, keep.night = TRUE, method = 'michalsky')
{
  #Solar constant
  Bo <- 1367

  if(missing(BTi)){
    d <- sold$Dates
    BTi <- fBTi(d, sample)
  }
  sun <- data.table(Dates = as.IDate(BTi),
                    Times = as.ITime(BTi))
  sun <- merge(sold, sun, by = 'Dates')
  sun[, eqtime := EoT]
  sun[, EoT := NULL]

  #sun hour angle
  sun[, w := sunHour(Dates, BTi, EoT = EoT, method = method, eqtime = eqtime)]

  #classify night elements
  sun[, night := abs(w) >= abs(ws)]

  #zenith angle
  sun[, cosThzS := zenith(Dates, lat, BTi,
```

```

                                method = method,
                                decl = decl,
                                w = w
                                )]

#solar altitude angle
sun[, AlS := asin(cosThzS)]

#azimuth
sun[, AzS := azimuth(Dates, lat, BTi, sample,
                    method = method,
                    decl = decl,
                    w = w,
                    cosThzS = cosThzS)]

#Extraterrestrial irradiance
sun[, Bo0 := Bo * eo * cosThzS]

#When it is night there is no irradiance
sun[night == TRUE, Bo0 := 0]

#Erase columns that are in sold
sun[, decl := NULL]
sun[, eo := NULL]
sun[, eqtime := NULL]
sun[, ws := NULL]
sun[, Bo0d := NULL]

#Column Dates with Times
sun[, Dates := as.POSIXct(Dates, Times, tz = 'UTC')]
sun[, Times := NULL]

#keep night
if(!keep.night){
  sun <- sun[night == FALSE]
}

return(sun)
}

```

EXTRACTO DE CÓDIGO A.25: *fSolI*

A.3.11. fSombra

```

fSombra<-function(angGen, distances, struct, modeTrk='fixed',prom=TRUE){

  stopifnot(modeTrk %in% c('two','horiz','fixed'))
  res=switch(modeTrk,
             two={fSombra6(angGen, distances, struct, prom)},
             horiz={fSombraHoriz(angGen, distances, struct)},
             fixed= {fSombraEst(angGen, distances, struct)}
             )
  return(res)
}

```

EXTRACTO DE CÓDIGO A.26: *fSombra*

```
fSombra2X<-function(angGen,distances,struct)
```

```

{
  stopifnot(is.list(struct), is.data.frame(distances))
  ##I prepare starting data
  P=with(struct,distances/W)
  b=with(struct,L/W)
  AzS=angGen$AzS
  Beta=angGen$Beta
  AlS=angGen$AlS

  d1=abs(P$Lew*cos(AzS)-P$Lns*sin(AzS))
  d2=abs(P$Lew*sin(AzS)+P$Lns*cos(AzS))
  FC=sin(AlS)/sin(Beta+AlS)
  s=b*cos(Beta)+(b*sin(Beta)+P$H)/tan(AlS)
  FS1=1-d1
  FS2=s-d2
  SombraCond=(FS1>0)*(FS2>0)*(P$Lew*Azs>=0)
  SombraCond[is.na(SombraCond)]<-FALSE #NAs are of no use to me in a logical
vector. I replace them with FALSE
  ## Result
  FS=SombraCond*(FS1*FS2*FC)/b
  FS[FS>1]<-1
  return(FS)
}

```

EXTRACTO DE CÓDIGO A.27: *fSombra2X*

```

fSombra6<-function(angGen, distances, struct, prom=TRUE)
{
  stopifnot(is.list(struct),
            is.data.frame(distances))
  ##distances only has three distances, so I generate a grid
  if (dim(distances)[1]==1){
    Red <- distances[, .(Lew = c(-Lew, 0, Lew, -Lew, Lew),
                          Lns = c(Lns, Lns, Lns, 0, 0),
                          H=H)]
  } else { #distances is an array, so there is no need to generate the grid
    Red<-distances[1:5,]} #I only need the first 5 rows...necessary in case
a wrong data.frame is delivered

  ## I calculate the shadow due to each of the 5 followers
  SombraGrupo<-matrix(ncol=5,nrow=dim(angGen)[1]) ###VECTORIZE
  for (i in 1:5) {SombraGrupo[,i]<-fSombra2X(angGen,Red[i,],struct)}
  ##To calculate the Average Shadow, I need the number of followers in each
position (distrib)
  distrib=with(struct,c(1,Ncol-2,1,Nrow-1,(Ncol-2)*(Nrow-1),Nrow-1))
  vProm=c(sum(distrib[c(5,6)]),
          sum(distrib[c(4,5,6)]),
          sum(distrib[c(4,5)]),
          sum(distrib[c(2,3,5,6)]),
          sum(distrib[c(1,2,4,5)]))
  Nseg=sum(distrib) ##Total number of followers
  ##With the SWEEP function I multiply the Shadow Factor of each type (
ShadowGroup columns) by the vProm result

  if (prom==TRUE){
    ## Average Shadow Factor in the group of SIX followers taking into
account distribution
    FS=rowSums(sweep(SombraGrupo,2,vProm,'*'))/Nseg
    FS[FS>1]<-1
  }
}

```

```

} else {
  ## Shadow factor on follower #5 due to the other 5 followers
  FS=rowSums(SombraGrupo)
  FS[FS>1]<-1}
return(FS)
}

```

EXTRACTO DE CÓDIGO A.28: *fSombra6*

```

fSombraEst<-function(angGen, distances, struct)
{
  stopifnot(is.list(struct),is.data.frame(distances))
  ## I prepare starting data
  dist <- with(struct, distances/L)
  Alfa <- angGen$Alfa
  Beta <- angGen$Beta
  AlS <- angGen$AlS
  AzS <- angGen$AzS
  cosTheta <- angGen$cosTheta
  h <- dist$H #It must be previously normalized
  d <- dist$D
  ## Calculations
  s=cos(Beta)+cos(Alfa-AzS)*(sin(Beta)+h)/tan(AlS)
  FC=sin(AlS)/sin(Beta+AlS)
  SombraCond=(s-d>0)
  FS=(s-d)*SombraCond*FC*(cosTheta>0)
  ## Result
  FS=FS*(FS>0)
  FS[FS>1]<-1
  return(FS)
}

```

EXTRACTO DE CÓDIGO A.29: *fSombraEst*

```

fSombraHoriz<-function(angGen, distances, struct)
{
  stopifnot(is.list(struct),is.data.frame(distances))
  ## I prepare starting data
  d <- with(struct, distances/L)
  AzS <- angGen$AzS
  AlS <- angGen$AlS
  Beta <- angGen$Beta
  lew <- d$Lew #It must be previously normalized
  ## Calculations
  Beta0=atan(abs(sin(AzS)/tan(AlS)))
  FS=1-lew*cos(Beta0)/cos(Beta-Beta0)
  SombraCond=(FS>0)
  ## Result
  FS=FS*SombraCond
  FS[FS>1]<-1
  return(FS)
}

```

EXTRACTO DE CÓDIGO A.30: *fSombraHoriz*

A.3.12. fTemp

```

fTemp<-function(sol, BD)

```

```

{
  ##sol is an object with class='Sol'
  ##BD is an object with class='Meteo', whose 'data' slot contains two columns
  called "TempMax" and "TempMin"

  stopifnot(class(sol)=='Sol')
  stopifnot(class(BD)=='Meteo')

  checkIndexD(indexD(sol), indexD(BD))

  Dates<-indexI(sol)
  x <- as.Date(Dates)
  ind.rep <- cumsum(c(1, diff(x) != 0))

  TempMax <- BD@data$TempMax[ind.rep]
  TempMin <- BD@data$TempMin[ind.rep]
  ws <- sol@sold$ws[ind.rep]
  w <- sol@solI$w

  ##Generate temperature sequence from database Maxima and Minima

  Tm=(TempMin+TempMax)/2
  Tr=(TempMax-TempMin)/2

  wp=pi/4

  a1=pi*12*(ws-w)/(21*pi+12*ws)
  a2=pi*(3*pi-12*w)/(3*pi-12*ws)
  a3=pi*(24*pi+12*(ws-w))/(21*pi+12*ws)

  T1=Tm-Tr*cos(a1)
  T2=Tm+Tr*cos(a2)
  T3=Tm-Tr*cos(a3)

  Ta=T1*(w<=ws)+T2*(w>ws&w<=wp)+T3*(w>wp)

  ##Result
  result<-data.table(Dates, Ta)
}

```

EXTRACTO DE CÓDIGO A.31: *fTemp***A.3.13. fTheta**

```

fTheta<-function(sol, beta, alfa=0, modeTrk='fixed', betaLim=90,
  BT=FALSE, struct, dist)
{
  stopifnot(modeTrk %in% c('two','horiz','fixed'))
  if (!missing(struct)) {stopifnot(is.list(struct))}
  if (!missing(dist)) {stopifnot(is.data.frame(dist))}

  betaLim=d2r(betaLim)
  lat=getLat(sol, 'rad')
  signLat=ifelse(sign(lat)==0, 1, sign(lat)) ##When lat=0, sign(lat)=0. I
  change it to sign(lat)=1

  solI<-as.data.tableI(sol, complete=TRUE, day = TRUE)
  AIS=solI$AIS

```



```

AzS=solI$AzS
decl=solI$decl
w<-solI$w

night<-solI$night

Beta<-switch(modeTrk,
  two = {Beta2x=pi/2-AlS
    Beta=Beta2x+(betaLim-Beta2x)*(Beta2x>betaLim)},
  fixed = rep(d2r(beta), length(w)),
  horiz={BetaHoriz0=atan(abs(sin(AzS)/tan(AlS)))
    if (BT){lew=dist$Lew/struct$L
      Longitud=lew*cos(BetaHoriz0)
      Cond=(Longitud>=1)
      Longitud[Cond]=1
      ## When Cond==TRUE Length=1
      ## and therefore asin(Length)=pi/2,
      ## so that BetaHoriz=BetaHoriz0
      BetaHoriz=BetaHoriz0+asin(Longitud)-pi/2
    } else {
      BetaHoriz=BetaHoriz0
      rm(BetaHoriz0)}
    Beta=ifelse(BetaHoriz>betaLim,betaLim,BetaHoriz)}
)
is.na(Beta) <- night

Alfa<-switch(modeTrk,
  two = AzS,
  fixed = rep(d2r(alfa), length(w)),
  horiz=pi/2*sign(AzS))
is.na(Alfa) <- night

cosTheta<-switch(modeTrk,
  two=cos(Beta-(pi/2-AlS)),
  horiz={
    t1=sin(decl)*sin(lat)*cos(Beta)
    t2=cos(decl)*cos(w)*cos(lat)*cos(Beta)
    t3=cos(decl)*abs(sin(w))*sin(Beta)
    cosTheta=t1+t2+t3
    rm(t1,t2,t3)
    cosTheta
  },
  fixed={
    t1=sin(decl)*sin(lat)*cos(Beta)
    t2=-signLat*sin(decl)*cos(lat)*sin(Beta)*cos(Alfa)
    t3=cos(decl)*cos(w)*cos(lat)*cos(Beta)
    t4=signLat*cos(decl)*cos(w)*sin(lat)*sin(Beta)*cos(Alfa)
    t5=cos(decl)*sin(w)*sin(Alfa)*sin(Beta)
    cosTheta=t1+t2+t3+t4+t5
    rm(t1,t2,t3,t4,t5)
    cosTheta
  }
)
is.na(cosTheta) <- night
cosTheta=cosTheta*(cosTheta>0) #when cosTheta<0, Theta is greater than 90°,
and therefore the Sun is behind the panel.

```

```

    result <- data.table(Dates = indexI(sol),
                        Beta, Alfa, cosTheta)
    return(result)
}

```

EXTRACTO DE CÓDIGO A.32: f_{θ} **A.3.14. HQCurve**

```

## HQCurve: no visible binding for global variable 'fb'
## HQCurve: no visible binding for global variable 'Q'
## HQCurve: no visible binding for global variable 'x'
## HQCurve: no visible binding for global variable 'y'
## HQCurve: no visible binding for global variable 'group.value'

if(getRversion() >= "2.15.1") globalVariables(c('fb', 'Q', 'x', 'y', 'group.
value'))

HQCurve<-function(pump){
  w1=3000 #synchronous rpm frequency
  wm=2870 #rpm frequency with slip when applying voltage at 50 Hz
  s=(w1-wm)/w1
  fen=50 #Nominal electrical frequency

  f=seq(35,50,by=5)
  Hn=with(pump,a*50^2+b*50*Qn+c*Qn^2) #height corresponding to flow rate and
    nominal frequency

  kiso=Hn/pump$Qn^2 #To paint the isoyield curve I take into account the laws of
    similarity
  Qiso=with(pump,seq(0.1*Qn,Qmax,l=10))
  Hiso=kiso*Qiso^2 #Isoperformance curve

  Curva<-expand.grid(fb=f,Q=Qiso)

  Curva<-within(Curva,{
    fe=fb/(1-s)
    H=with(pump,a*fb^2+b*fb*Q+c*Q^2)

    is.na(H) <- (H<0)
    Q50=50*Q/fe
    H50=H*(50/fe)^2
    etab=with(pump,j*Q50^2+k*Q50+1)
    Pb50=2.725*H50*Q50/etab
    Pb=Pb50*(fb/50)^3

    Pbc=Pb*50/fe
    etam=with(pump,g*(Pbc/Pmn)^2+h*(Pbc/Pmn)+i)
    Pmc=Pbc/etam
    Pm=Pmc*fe/50

    etac=0.95 #Variable frequency drive performance
    cab=0.05 #Cable losses
    Pdc=Pm/(etac*(1-cab))
    rm(etac,cab,Pmc,Pbc,Pb50,Q50,H50)
  })

###H-Q curve at different frequencies

```

```

##I check if I have the lattice package available, which should have been
loaded in .First.lib
lattice.disp<-"lattice" %in% .packages()
latticeExtra.disp<-"latticeExtra" %in% .packages()
if (lattice.disp && latticeExtra.disp) {
  p<-xyplot(H~Q,groups=factor(fb),data=Curva, type='l',
    par.settings=custom.theme.2(),
    panel=function(x,y,groups,...){
      panel.superpose(x,y,groups,...)
      panel.xyplot(Qiso,Hiso,col='black',...)
      panel.text(Qiso[1], Hiso[1], 'ISO', pos=3)}
  )
  p=p+glayer(panel.text(x[1], y[1], group.value, pos=3))
  print(p)
  result<-list(result=Curva, plot=p)
} else {
  warning('lattice and/or latticeExtra packages are not available. Thus, the
plot could not be created')
  result<-Curva}
}

```

EXTRACTO DE CÓDIGO A.33: *HQCurve*

A.3.15. local2Solar

```

local2Solar <- function(x, lon=NULL){
  tz=attr(x, 'tzone')
  if (tz==' ' || is.null(tz)) {tz='UTC'}
  ##Daylight savings time
  A0=3600*dst(x)
  A0neg=(A0<0)
  if (any(A0neg)) {
    A0[A0neg]=0
    warning('Some Daylight Savings Time unknown. Set to zero.')
  }
  ##Difference between local longitude and time zone longitude LH
  LH=lonHH(tz)
  if (is.null(lon))
    {deltaL=0
    } else
    {deltaL=d2r(lon)-LH
    }
  ##Local time corrected to UTC
  tt <- format(x, tz=tz)
  result <- as.POSIXct(tt, tz='UTC')-A0+r2sec(deltaL)
  result
}

```

EXTRACTO DE CÓDIGO A.34: *local2Solar*

A.3.16. markovG0

```

## Objects loaded at startup from data/MTM.RData
if(getRversion() >= "2.15.1") globalVariables(c(
  'MTM', ## Markov Transition Matrices
  'Ktmtm', ## Kt limits to choose a matrix from MTM
  'Ktlim' ## Daily kt range of each matrix.
))

```

```

markovG0 <- function(G0dm, sold){
  sold <- copy(sold)
  timeIndex <- sold$Dates
  Bo0d <- sold$Bo0d
  Bo0dm <- sold[, mean(Bo0d), by = .(month(Dates), year(Dates))][[3]]
  ktm <- G0dm/Bo0dm

  ##Calculates which matrix to work with for each month
  whichMatrix <- findInterval(ktm, Ktmtm, all.inside = TRUE)

  ktd <- state <- numeric(length(timeIndex))
  state[1] <- 1
  ktd[1] <- ktm[state[1]]
  for (i in 2:length(timeIndex)){
    iMonth <- month(timeIndex[i])
    colMonth <- whichMatrix[iMonth]
    rng <- Ktlim[, colMonth]
    classes <- seq(rng[1], rng[2], length=11)
    matMonth <- MTM[(10*colMonth-9):(10*colMonth),]
    ## http://www-rohan.sdsu.edu/~babailey/stat575/mcsim.r
    state[i] <- sample(1:10, size=1, prob=matMonth[state[i-1],])
    ktd[i] <- runif(1, min=classes[state[i]], max=classes[state[i]+1])
  }
  G0dmMarkov <- data.table(ktd, Bo0d)
  G0dmMarkov <- G0dmMarkov[, mean(ktd*Bo0d), by = .(month(timeIndex), year(
timeIndex))][[3]]
  fix <- G0dm/G0dmMarkov
  indRep <- month(timeIndex)
  fix <- fix[indRep]
  G0d <- data.table(Dates = timeIndex, G0d = ktd * Bo0d * fix)
  G0d
}

```

EXTRACTO DE CÓDIGO A.35: *markovG0*

A.3.17. NmgPVPS

```

## NmgPVPS: no visible binding for global variable 'Pnom'
## NmgPVPS: no visible binding for global variable 'group.value'

if(getRversion() >= "2.15.1") globalVariables(c('Pnom', 'group.value'))

NmgPVPS <- function(pump, Pg, H, Gd, Ta=30,
                    lambda=0.0045, TONC=47,
                    eta=0.95, Gmax=1200, t0=6, Nm=6,
                    title='', theme=custom.theme.2()){

  ##I build the type day by IEC procedure
  t=seq(-t0,t0,l=2*t0*Nm);
  d=Gd/(Gmax*2*t0)
  s=(d*pi/2-1)/(1-pi/4)
  G=Gmax*cos(t/t0*pi/2)*(1+s*(1-cos(t/t0*pi/2)))
  G[G<0]<-0
  G=G/(sum(G,na.rm=1)/Nm)*Gd
  Red<-expand.grid(G=G,Pnom=Pg,H=H,Ta=Ta)
  Red<-within(Red,{Tcm<-Ta+G*(TONC-20)/800
                    Pdc=Pnom*G/1000*(1-lambda*(Tcm-25)) #Available DC power

```

```

Pac=Pdc*eta})) #Inverter yield

res=data.table(Red,Q=0)

for (i in seq_along(H)){
  fun=fPump(pump, H[i])
  Cond=res$H==H[i]
  x=res$Pac[Cond]
  z=res$Pdc[Cond]
  rango=with(fun,x>=lim[1] & x<=lim[2]) #I limit the power to the
operating range of the pump.
  x[!rango]<-0
  z[!rango]<-0
  y=res$Q[Cond]
  y[rango]<-fun$fQ(x[rango])
  res$Q[Cond]=y
  res$Pac[Cond]=x
  res$Pdc[Cond]=z
}

resumen <- res[, lapply(.SD, function(x)sum(x, na.rm = 1)/Nm),
by = .(Pnom, H)]
param=list(pump=pump, Pg=Pg, H=H, Gd=Gd, Ta=Ta,
lambda=lambda, TONC=TONC, eta=eta,
Gmax=Gmax, t0=t0, Nm=Nm)

###Abacus with common X-axes

##I check if I have the lattice package available, which should have been
loaded in .First.lib
lattice.disp<-"lattice" %in% .packages()
latticeExtra.disp<-"latticeExtra" %in% .packages()
if (lattice.disp && latticeExtra.disp){
  tema<-theme
  tema1 <- modifyList(tema, list(layout.width = list(panel=1,
ylab = 2, axis.left=1.0,
left.padding=1, ylab.axis.padding=1,
axis.panel=1)))
  tema2 <- modifyList(tema, list(layout.width = list(panel=1,
ylab = 2, axis.left=1.0, left.padding=1,
ylab.axis.padding=1, axis.panel=1)))
  temaT <- modifyList(tema, list(layout.heights = list(panel = c(1, 1))))
  p1 <- xyplot(Q~Pdc, groups=H, data=resumen,
ylab="Qd (m\u00b3/d)",type=c('l','g'),
par.settings = tema1)

  p1lab<-p1+glayer(panel.text(x[1], y[1], group.value, pos=2, cex=0.7))

  ##I paint the linear regression because Pnom~Pdc depends on the height.
  p2 <- xyplot(Pnom~Pdc, groups=H, data=resumen,
ylab="Pg",type=c('l','g'), #type=c('smooth','g'),
par.settings = tema2)
  p2lab<-p2+glayer(panel.text(x[1], y[1], group.value, pos=2, cex=0.7))

  p<-update(c(p1lab, p2lab, x.same = TRUE),
main=paste(title, '\nSP', pump$Qn, 'A', pump$stages, ' ',
'Gd ', Gd/1000," kWh/m\u00b3",sep=''),

```

```

        layout = c(1, 2),
        scales=list(x=list(draw=FALSE)),
        xlab='',
        ylab = list(c("Qd (m\u00b3/d)", "Pg (Wp)"), y = c(1/4, 3/4)),
        par.settings = temaT
    )

    print(p)
    result<-list(I=res,D=resumen, plot=p, param=param)
} else {
    warning('lattice, latticeExtra packages are not all available. Thus, the
    plot could not be created')
    result<-list(I=res,D=resumen, param=param)
}
}

```

EXTRACTO DE CÓDIGO A.36: *NmgPVPS*

A.3.18. solarAngles

```

#### Declination ####
declination <- function(d, method = 'michalsky')
{
    ##Method check
    if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
        warning("'method' must be: michalsky, cooper, strous or spencer. Set
        michalsky")
        method = 'michalsky'
    }

    ## x is an IDate
    d <- as.IDate(d)
    ## Day of year
    dn <- yday(d)
    ## Days from 2000-01-01
    origin <- as.IDate('2000-01-01')
    jd <- as.numeric(d - origin)
    X <- 2 * pi * (dn - 1) / 365

    switch(method,
        michalsky = {
            meanLong <- (280.460 + 0.9856474 * jd)%%360
            meanAnomaly <- (357.528 + 0.9856003 * jd)%%360
            eclipLong <- (meanLong + 1.915 * sin(d2r(meanAnomaly)) +
                0.02 * sin(d2r(2 * meanAnomaly)))%%360
            excen <- 23.439 - 0.0000004 * jd
            sinEclip <- sin(d2r(eclipLong))
            sinExcen <- sin(d2r(excen))
            asin(sinEclip * sinExcen)
        },
        cooper = {
            ##P.I. Cooper. "The Absorption of Solar Radiation in Solar Stills
            ". Solar Energy 12 (1969).
            d2r(23.45) * sin(2 * pi * (dn + 284) / 365)
        },
        strous = {
            meanAnomaly <- (357.5291 + 0.98560028 * jd)%%360
            coefC <- c(1.9148, 0.02, 0.0003)
            sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))
        }
    )
}

```

```

        C <- colSums(coefC * sinC)
        trueAnomaly <- (meanAnomaly + C) %% 360
        eclipLong <- (trueAnomaly + 282.9372) %% 360
        excen <- 23.435
        sinEclip <- sin(d2r(eclipLong))
        sinExcen <- sin(d2r(excen))
        asin(sinEclip * sinExcen)
    },
    spencer = {
        ## J.W. Spencer. "Fourier Series Representation of the Position
of the Sun". 2 (1971).
        ##URL: http://www.mail-archive.com/sundial@uni-koeln.de/msg01050.html.
        0.006918 - 0.399912 * cos(X) + 0.070257 * sin(X) -
        0.006758 * cos(2 * X) + 0.000907 * sin(2 * X) -
        0.002697 * cos(3 * X) + 0.001480 * sin(3 * X)
    })
}

#### Eccentricity ####
eccentricity <- function(d, method = 'michalsky')
{
    ##Method check
    if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
        warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
        method = 'michalsky'
    }

    ##x is an IDate
    d <- as.IDate(d)
    ##Day of year
    dn <- yday(d)
    X <- 2 * pi * (dn-1)/365

    switch(method,
        cooper = 1 + 0.033*cos(2*pi*dn/365),
        spencer = ,
        michalsky = ,
        strous = 1.000110 + 0.034221*cos(X) +
            0.001280*sin(X) + 0.000719*cos(2*X) +
            0.000077*sin(2*X)
    )
}

#### Equation of time

##Alan M.Whitman "A simple expression for the equation of time"
##EoT=ts-t, donde ts es la hora solar real y t es la hora solar
##media. Valores negativos implican que el sol real se retrasa
##respecto al medio
eot <- function(d)
{
    ## d in an IDate
    d <- as.IDate(d)
    ## Day of year

```

```

dn <- yday(d)
M <- 2 * pi/365.24 * dn
EoT <- 229.18 * (-0.0334 * sin(M) +
               0.04184 * sin(2 * M + 3.5884))
EoT <- h2r(EoT/60)
return(EoT)
}

#### Solar time ####
sunrise <- function(d, lat, method = 'michalsky',
                   decl = declination(d, method))
{
  ##Method check
  if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
    warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
    method = 'michalsky'
  }

  cosWs <- -tan(d2r(lat)) * tan(decl)
  #sunrise, negative since it is before noon
  ws <- -acos(cosWs)
  #Polar day/night
  polar <- which(is.nan(ws))
  ws[polar] <- -pi * (cosWs[polar] < -1) + 0 * (cosWs[polar] > 1)
  return(ws)
}

#### Extraterrestrial irradiation ####
bo0d <- function(d, lat, method = 'michalsky',
                decl = declination(d, method),
                eo = eccentricity(d, method),
                ws = sunrise(d, lat, method))
{
  ##Method check
  if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
    warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
    method = 'michalsky'
  }

  #solar constant
  Bo <- 1367
  latr <- d2r(lat)
  #The negative sign due to the definition of ws
  Bo0d <- -24/pi * Bo * eo * (ws * sin(latr) * sin(decl) +
                             cos(latr) * cos(decl) * sin(ws))
  return(Bo0d)
}

#### Sun hour angle ####
sunHour <- function(d, BTi, sample = '1 hour', EoT = TRUE, method = 'michalsky',
                  eqtime = eot(d))
{
  ##Method check
  if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){

```



```

    warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
    method = 'michalsky'
  }

  if(missing(BTi)){
    BTi <- fBTi(d = d, sample = sample)
  }else {
    if (inherits(BTi, 'data.table')) {
      Times <- as.ITime(BTi$Times)
      Dates <- as.IDate(BTi$Dates)
      BTi <- as.POSIXct(Dates, Times, tz = 'UTC')
    }
    else {
      BTi <- as.POSIXct(BTi, tz = 'UTC')
    }
  }
  rep <- cumsum(c(1, diff(as.Date(BTi)) != 0))
  if(EoT)
  {
    EoT <- eqtime
    if(length(EoT) != length(BTi)){EoT <- EoT[rep]}
  }else{EoT <- 0}

  jd <- as.numeric(julian(BTi, origin = '2000-01-01 12:00:00 UTC'))
  T0 <- hms(BTi)

  w=switch(method,
    cooper = h2r(T0-12)+EoT,
    spencer = h2r(T0-12)+EoT,
    michalsky = {
      meanLong <- (280.460+0.9856474*jd)%%360
      meanAnomaly <- (357.528+0.9856003*jd)%%360
      eclipLong <- (meanLong +1.915*sin(d2r(meanAnomaly))+0.02*sin(
d2r(2*meanAnomaly)))%%360
      excen <- 23.439-0.0000004*jd

      sinEclip <- sin(d2r(eclipLong))
      cosEclip <- cos(d2r(eclipLong))
      cosExcen <- cos(d2r(excen))

      ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))%%360

      ##local mean sidereal time, LMST
      ##T0 has been previously corrected with local2Solar in order
      ##to include the longitude, daylight savings, etc.
      lmst <- (h2d(6.697375 + 0.0657098242*jd + T0))%%360
      w <- (lmst-ascension)
      w <- d2r(w + 360*(w < -180) - 360*(w > 180))
    },
    strous = {
      meanAnomaly <- (357.5291 + 0.98560028*jd)%%360
      coefC <- c(1.9148, 0.02, 0.0003)
      sinC <- sin(outer(1:3, d2r(meanAnomaly), '*'))
      C <- colSums(coefC*sinC)
      trueAnomaly <- (meanAnomaly + C)%%360
      eclipLong <- (trueAnomaly + 282.9372)%%360
      excen <- 23.435
    }
  )

```

```

sinEclip <- sin(d2r(eclipLong))
cosEclip <- cos(d2r(eclipLong))
cosExcen <- cos(d2r(excen))

ascension <- r2d(atan2(sinEclip*cosExcen, cosEclip))%%360

##local mean sidereal time, LMST
##TO has been previously corrected with local2Solar in order
##to include the longitude, daylight savings, etc.
lmst <- (280.1600+360.9856235*jd)%%360
w <- (lmst-ascension)
w <- d2r(w + 360*(w< -180) - 360*(w>180))
    }
  )
  return(w)
}

#### zenith angle ####
zenith <- function(d, lat, BTi, sample = '1 hour', method = 'michalsky',
  decl = declination(d, method),
  w = sunHour(d, BTi, sample, method = method))
{
  ##Method check
  if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
    warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
    method = 'michalsky'
  }

  if(missing(BTi)){BTi <- fBTi(d, sample)}
  x <- as.Date(BTi)
  rep <- cumsum(c(1, diff(x) != 0))
  latr <- d2r(lat)
  if(length(decl) == length(BTi)){decl <- decl}
  else{decl <- decl[rep]}
  zenith <- sin(decl) * sin(latr) +
    cos(decl) * cos(w) * cos(latr)
  zenith <- ifelse(zenith > 1, 1, zenith)
  return(zenith)
}

#### azimuth ####
azimuth <- function(d, lat, BTi, sample = '1 hour', method = 'michalsky',
  decl = declination(d, method),
  w = sunHour(d, BTi, sample, method = method),
  cosThzS = zenith(d, lat, BTi, sample, method, decl, w))
{
  ##Method check
  if(!(method %in% c("michalsky", "cooper", "strous", "spencer"))){
    warning("'method' must be: michalsky, cooper, strous or spencer. Set
michalsky")
    method = 'michalsky'
  }

  signLat <- ifelse(sign(lat) == 0, 1, sign(lat)) #if the sign of lat is 0, it
changes it to 1
  if(missing(BTi)){BTi <- fBTi(d, sample)}

```

```

x <- as.Date(BTi)
rep <- cumsum(c(1, diff(x) != 0))
latr <- d2r(lat)
if(length(decl) != length(BTi)){decl <- decl[rep]}
AlS <- asin(cosThzS)
cosazimuth <- signLat * (cos(decl) * cos(w) * sin(latr) -
                        cos(latr) * sin(decl)) / cos(AlS)
cosazimuth <- ifelse(abs(cosazimuth)>1, sign(cosazimuth), cosazimuth)
azimuth <- sign(w)*acos(cosazimuth)
return(azimuth)
}

```

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

```

#degrees to radians
d2r<-function(x){x*pi/180}

#radians to degrees
r2d<-function(x){x*180/pi}

#hours to radians
h2r<-function(x){x*pi/12}

#hours to degrees
h2d<-function(x){x*180/12}

#radians to hours
r2h<-function(x){x*12/pi}

#degrees to hours
d2h<-function(x){x*12/180}

#radians to seconds
r2sec<-function(x){x*12/pi*3600}

#radians to minutes
r2min<-function(x){x*12/pi*60}

```

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

```

#complete time to hours
t2h <- function(x)
{
  hour(x)+minute(x)/60+second(x)/3600
}

#hours minutes and seconds to hours
hms <- function(x)
{
  hour(x)+minute(x)/60+second(x)/3600
}

#day of the year
doy <- function(x){

```

```

    as.numeric(format(x, '%j'))
}

#day of the month
dom <- function(x){
  as.numeric(format(x, '%d'))
}

#trunc days
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`

```

setGeneric('as.data.tableI',
  function(object, complete=FALSE, day=FALSE){standardGeneric('as.data.
    tableI')}})

setMethod('as.data.tableI',
  signature=(object='Sol'),
  definition=function(object, complete=FALSE, day=FALSE){
    sol <- copy(object)
    BTi <- indexI(sol)
    BTi <- truncDay(BTi)
    ind.rep <- cumsum(c(1, diff(BTi, units='days')!=0))
    solI <- sol@solI
    solD <- sol@solD[ind.rep]
    if(complete){
      data <- data.table(solI, solD[, Dates := NULL])
    } else{data <- solI}
    if(day){
      ind <- indexI(sol)
      data[, day := doy(ind)]
      data[, month := month(ind)]
      data[, year := year(ind)]
    }
    return(data)
  }
)

setMethod('as.data.tableI',
  signature = (object='G0'),
  definition = function(object, complete=FALSE, day=FALSE){
    g0 <- copy(object)
    BTi <- indexI(g0)
    BTi <- truncDay(BTi)
    ind.rep <- cumsum(c(1, diff(BTi)!=0))
    G0I <- g0@G0I
    solI <- g0@solI
    solD <- g0@solD[ind.rep]
    Ta <- g0@Ta
    if(length(Ta[[1]]!=length(G0I[[1]]))) Ta <- Ta[ind.rep]
    if(complete){
      data <- data.table(solI,
        G0I[, Dates := NULL],

```

```

                                sold[, Dates := NULL],
                                Ta[, Dates := NULL])
  } else{
    GOI[, Kt := NULL]
    GOI[, Fd := NULL]
    data <- GOI
  }
  if(day){
    ind <- indexI(object)
    data[, day := doy(ind)]
    data[, month := month(ind)]
    data[, year := year(ind)]
  }
  return(data)
}
)

setMethod('as.data.tableI',
signature = (object='Gef'),
definition = function(object, complete=FALSE, day=FALSE){
  gef <- copy(object)
  BTi <- indexI(gef)
  BTi <- truncDay(BTi)
  ind.rep <- cumsum(c(1, diff(BTi, units='days')!=0))
  GefI <- gef@GefI
  GOI <- gef@GOI
  solI <- gef@solI
  sold <- gef@sold[ind.rep]
  Ta <- gef@Ta
  if(length(Ta[[1]]!=length(GefI[[1]]))) Ta <- Ta[ind.rep]
  if(complete){
    data <- data.table(solI,
                        GOI[, Dates := NULL],
                        sold[, Dates := NULL],
                        Ta[, Dates := NULL],
                        GefI[, Dates := NULL])
  } else {
    data <- GefI[, c('Dates','Gef',
                    'Bef', 'Def')]
  }
  if(day){
    ind <- indexI(object)
    data[, day := doy(ind)]
    data[, month := month(ind)]
    data[, year := year(ind)]
  }
  return(data)
}
)

setMethod('as.data.tableI',
signature = (object='ProdGCPV'),
definition = function(object, complete=FALSE, day=FALSE){
  prodgcpv <- copy(object)
  BTi <- indexI(prodgcpv)
  BTi <- truncDay(BTi)
  ind.rep <- cumsum(c(1, diff(BTi, units = 'days')!=0))
  prodI <- prodgcpv@prodI

```

```

Theta <- prodgcpv@Theta
GefI <- prodgcpv@GefI
GOI <- prodgcpv@GOI
solI <- prodgcpv@solI
sold <- prodgcpv@sold[ind.rep]
Ta <- prodgcpv@Ta
if(length(Ta[[1]]!=length(prodI[[1]]))) Ta <- Ta[ind.rep]
if(complete){
  data <- data.table(solI,
                    GOI[, Dates := NULL],
                    sold[, Dates := NULL],
                    Ta[, Dates := NULL],
                    GefI[, Dates := NULL],
                    prodI[, Dates := NULL],
                    Theta[, Dates := NULL])
} else {
  data <- prodI[, c('Dates', 'Pac', 'Pdc')]
}
if(day){
  ind <- indexI(object)
  data[, day := doy(ind)]
  data[, month := month(ind)]
  data[, year := year(ind)]
}
return(data)
}
)

setMethod('as.data.tableI',
signature = (object='ProdPVPS'),
definition = function(object, complete=FALSE, day=FALSE){
  prodpvps <- copy(object)
  BTi <- indexI(prodpvps)
  BTi <- truncDay(BTi)
  ind.rep <- cumsum(c(1, diff(BTi, units='days')!=0))
  prodI <- prodpvps@prodI
  Theta <- prodpvps@Theta
  GefI <- prodpvps@GefI
  GOI <- prodpvps@GOI
  solI <- prodpvps@solI
  sold <- prodpvps@sold[ind.rep]
  Ta <- prodpvps@Ta
  if(length(Ta[[1]]!=length(prodI[[1]]))) Ta <- Ta[ind.rep]
  if(complete){
    data <- data.table(solI,
                      GOI[, Dates := NULL],
                      sold[, Dates := NULL],
                      Ta[, Dates := NULL],
                      GefI[, Dates := NULL],
                      prodI[, Dates := NULL],
                      Theta[, Dates := NULL])
  } else {
    data <- prodI[, c('Dates', 'Pac', 'Pdc')]
  }
  if(day){
    ind <- indexI(object)
    data[, day := doy(ind)]
    data[, month := month(ind)]
  }
}
)

```

```

        data[, year := year(ind)]
    }
    return(data)
}
)
```

EXTRACTO DE CÓDIGO A.40: *as.data.tableI*

A.4.2. *as.data.tableD*

```

setGeneric('as.data.tableD', function(object, complete=FALSE, day=FALSE){
  standardGeneric('as.data.tableD')})

setMethod('as.data.tableD',
  signature=(object='Sol'),
  definition=function(object, complete=FALSE, day=FALSE){
    sol <- copy(object)
    sold <- sol@solD
    data <- sold
    if(day){
      ind <- indexD(object)
      data[, day := doy(ind)]
      data[, month := month(ind)]
      data[, year := year(ind)]
    }
    return(data)
  }
)

setMethod('as.data.tableD',
  signature = (object='G0'),
  definition = function(object, complete=FALSE, day=FALSE){
    g0 <- copy(object)
    GOD <- g0@GOD
    sold <- g0@solD
    if(complete){
      data <- data.table(GOD, sold[, Dates := NULL])
    } else {
      GOD[, Fd := NULL]
      GOD[, Kt := NULL]
      data <- GOD
    }
    if(day){
      ind <- indexD(object)
      data[, day := doy(ind)]
      data[, month := month(ind)]
      data[, year := year(ind)]
    }
    return(data)
  })

setMethod('as.data.tableD',
  signature = (object='Gef'),
  definition = function(object, complete=FALSE, day=FALSE){
    gef <- copy(object)
    GefD <- gef@GefD
    GOD <- gef@GOD
    sold <- gef@solD
  })

```

```

        if(complete){
            data <- data.table(GefD,
                               GOD[, Dates := NULL],
                               sold[, Dates := NULL])
        } else {data <- GefD[, c('Dates', 'Gefd',
                                'Defd', 'Befd')]}

        if(day){
            ind <- indexD(object)
            data[, day := doy(ind)]
            data[, month := month(ind)]
            data[, year := year(ind)]
        }
        return(data)
    }
)

setMethod('as.data.tableD',
signature = (object='ProdGCPV'),
definition = function(object, complete=FALSE, day=FALSE){
    prodgcpv <- copy(object)
    prodD <- prodgcpv@prodD
    GefD <- prodgcpv@GefD
    GOD <- prodgcpv@GOD
    sold <- prodgcpv@sold
    if(complete){
        data <- data.table(prodD,
                            GefD[, Dates := NULL],
                            GOD[, Dates := NULL],
                            sold[, Dates := NULL]
                            )
    } else { data <- prodD[, c('Dates', 'Eac',
                                'Edc', 'Yf')]}

    if(day){
        ind <- indexD(object)
        data[, day := doy(ind)]
        data[, month := month(ind)]
        data[, year := year(ind)]
    }
    return(data)
}
)

setMethod('as.data.tableD',
signature = (object='ProdPVPS'),
definition = function(object, complete=FALSE, day=FALSE){
    prodpvps <- copy(object)
    prodD <- prodpvps@prodD
    GefD <- prodpvps@GefD
    GOD <- prodpvps@GOD
    sold <- prodpvps@sold
    if(complete){
        data <- data.table(prodD,
                            GefD[, Dates := NULL],
                            GOD[, Dates := NULL],
                            sold[, Dates := NULL]
                            )
    } else { data <- prodD[, c('Dates', 'Eac',
                                'Qd', 'Yf')]}

```



```

        if(day){
            ind <- indexD(object)
            data[, day := doy(ind)]
            data[, month := month(ind)]
            data[, year := year(ind)]
        }
        return(data)
    }
)

```

EXTRACTO DE CÓDIGO A.41: *as.data.tableD*

A.4.3. *as.data.tableM*

```

setGeneric('as.data.tableM', function(object, complete = FALSE, day=FALSE){
    standardGeneric('as.data.tableM')})

setMethod('as.data.tableM',
    signature=(object='G0'),
    definition=function(object, complete=FALSE, day=FALSE){
        g0 <- copy(object)
        G0dm <- g0@G0dm
        data <- G0dm
        if(day){
            ind <- indexD(object)
            data[, month := month(ind)]
            data[, year := year(ind)]
        }
        return(data)
    }
)

setMethod('as.data.tableM',
    signature=(object='Gef'),
    definition = function(object, complete=FALSE, day=FALSE){
        gef <- copy(object)
        Gefdm <- gef@Gefdm
        G0dm <- gef@G0dm
        if(complete){
            data <- data.table(Gefdm, G0dm[, Dates := NULL])
        } else {data <- Gefdm}
        if(day){
            ind <- indexD(object)
            data[, month := month(ind)]
            data[, year := year(ind)]
        }
        return(data)
    }
)

setMethod('as.data.tableM',
    signature = (object='ProdGCPV'),
    definition = function(object, complete=FALSE, day=FALSE){
        prodgcpv <- copy(object)
        prodDm <- prodgcpv@prodDm
        Gefdm <- prodgcpv@Gefdm
        G0dm <- prodgcpv@G0dm
        if(complete){

```

```

        data <- data.table(prodDm,
                           Gefdm[, Dates := NULL],
                           G0dm[, Dates := NULL])
      } else {data <- prodDm}
      if(day){
        ind <- indexD(object)
        data[, month := month(ind)]
        data[, year := year(ind)]
      }
      return(data)
    }
  )

setMethod('as.data.tableM',
  signature = (object='ProdPVPS'),
  definition = function(object, complete=FALSE, day=FALSE){
    prodpvps <- copy(object)
    prodDm <- prodpvps@prodDm
    Gefdm <- prodpvps@Gefdm
    G0dm <- prodpvps@G0dm
    if(complete){
      data <- data.table(prodDm,
                         Gefdm[, Dates := NULL],
                         G0dm[, Dates := NULL])
    } else {data <- prodDm}
    if(day){
      ind <- indexD(object)
      data[, month := month(ind)]
      data[, year := year(ind)]
    }
    return(data)
  }
)

```

EXTRACTO DE CÓDIGO A.42: *as.data.tableM*A.4.4. *as.data.tableY*

```

setGeneric('as.data.tableY', function(object, complete=FALSE, day=FALSE){
  standardGeneric('as.data.tableY')})

setMethod('as.data.tableY',
  signature=(object='G0'),
  definition=function(object, complete=FALSE, day=FALSE){
    g0 <- copy(object)
    G0y <- g0@G0y
    data <- G0y
    if(day){data[, year := Dates]}
    return(data)
  }
)

setMethod('as.data.tableY',
  signature = (object='Gef'),
  definition = function(object, complete=FALSE, day=FALSE){
    gef <- copy(object)
    Gefy <- gef@Gefy
    G0y <- gef@G0y
  }
)

```

```

        if(complete){
            data <- data.table(Gefy, G0y[, Dates := NULL])
        } else {data <- Gefy}
        if(day){data[, year := Dates]}
        return(data)
    }
)

setMethod('as.data.tableY',
  signature = (object='ProdGCPV'),
  definition = function(object, complete=FALSE, day=FALSE){
    prodgcpv <- copy(object)
    prody <- prodgcpv@prody
    Gefy <- prodgcpv@Gefy
    G0y <- prodgcpv@G0y
    if(complete){
        data <- data.table(prody,
                           Gefy[, Dates := NULL],
                           G0y[, Dates := NULL])
    } else {data <- prody}
    if(day){data[, year := Dates]}
    return(data)
  }
)

setMethod('as.data.tableY',
  signature = (object='ProdPVPS'),
  definition = function(object, complete=FALSE, day=FALSE){
    prodpvps <- copy(object)
    prody <- prodpvps@prody
    Gefy <- prodpvps@Gefy
    G0y <- prodpvps@G0y
    if(complete){
        data <- data.table(prody,
                           Gefy[, Dates := NULL],
                           G0y[, Dates := NULL])
    } else {data <- prody}
    if(day){data[, year := Dates]}
    return(data)
  }
)

```

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

A.4.5. compare

```

## compareFunction: no visible binding for global variable 'name'
## compareFunction: no visible binding for global variable 'x'
## compareFunction: no visible binding for global variable 'y'
## compareFunction: no visible binding for global variable 'group.value'

if(getRversion() >= "2.15.1") globalVariables(c('name', 'x', 'y', 'group.value'))

setGeneric('compare', signature='...', function(...){standardGeneric('compare')}
  })

compareFunction <- function(..., vars){

```

```

dots <- list(...)
nms0 <- substitute(list(...))
if (!is.null(names(nms0))) { ##in do.call
  nms <- names(nms0[-1])
} else {
  nms <- as.character(nms0[-1])
}
foo <- function(object, label){
  yY <- colMeans(as.data.tableY(object, complete = TRUE)[, ..vars])
  yY <- cbind(stack(yY), name=label)
  yY
}
cdata <- mapply(FUN=foo, dots, nms, SIMPLIFY=FALSE)
z <- do.call(rbind, cdata)
z$ind <- ordered(z$ind, levels=vars)
p <- dotplot(ind~values, groups=name, data=z, type='b',
  par.settings=solaR.theme)
print(p+glayer(panel.text(x[length(x)], y[length(x)],
  label=group.value, cex=0.7, pos=3, srt=45)))
return(z)
}

setMethod('compare',
  signature='G0',
  definition=function(...){
    vars <- c('D0d', 'B0d', 'G0d')
    res <- compareFunction(..., vars=vars)
    return(res)
  }
)

setMethod('compare',
  signature='Gef',
  definition=function(...){
    vars <- c('Defd', 'Befd', 'Gefd')
    res <- compareFunction(..., vars=vars)
    return(res)
  }
)

setMethod('compare',
  signature='ProdGCPV',
  definition=function(...){
    vars <- c('G0d', 'Gefd', 'Yf')
    res <- compareFunction(..., vars=vars)
    return(res)
  }
)

```

EXTRACTO DE CÓDIGO A.44: *compare*

A.4.6. getData

```

## extracts the data for class Meteo ##
setGeneric('getData', function(object){standardGeneric('getData')})

### getData ####

```

```
setMethod('getData',
  signature = (object = 'Meteo'),
  definition = function(object){
    result <- object@data
    return(result)
  })
```

EXTRACTO DE CÓDIGO A.45: *getData***A.4.7. getG0**

```
## extracts the global irradiance for class Meteo ##
setGeneric('getG0', function(object){standardGeneric('getG0')})

### getG0 ###
setMethod('getG0',
  signature = (object = 'Meteo'),
  definition = function(object){
    result <- getData(object)
    return(result$G0)
  })
```

EXTRACTO DE CÓDIGO A.46: *getG0***A.4.8. getLat**

```
## extracts the latitude from the objects ##
setGeneric('getLat', function(object, units = 'rad')
{standardGeneric('getLat')})

## extracts the latitude from the objects ##
setGeneric('getLat', function(object, units = 'rad')
{standardGeneric('getLat')})

setMethod('getLat',
  signature = (object = 'Meteo'),
  definition = function(object, units = 'rad'){
    stopifnot(units %in% c('deg', 'rad'))
    result = switch(units,
      rad = d2r(object@latm),
      deg = object@latm)
    return(result)
  })
```

EXTRACTO DE CÓDIGO A.47: *getLat***A.4.9. indexD**

```
## extract the index of the daily data ##
setGeneric('indexD', function(object){standardGeneric('indexD')})
### indexD ###
setMethod('indexD',
  signature = (object = 'Sol'),
  definition = function(object){as.POSIXct(object@solD$Dates)
  })

setMethod('indexD',
  signature = (object = 'Meteo'),
```

```
definition = function(object){as.POSIXct(getData(object)$Dates)}}
```

EXTRACTO DE CÓDIGO A.48: *indexD*A.4.10. *indexI*

```
## extract the index of the intradaily data ##
setGeneric('indexI', function(object){standardGeneric('indexI')})
### indexI ###
setMethod('indexI',
  signature = (object = 'Sol'),
  definition = function(object){as.POSIXct(object@solI$Dates)
})
```

EXTRACTO DE CÓDIGO A.49: *indexI*A.4.11. *levelplot*

```
setGeneric('levelplot')

setMethod('levelplot',
  signature=c(x='formula', data='Meteo'),
  definition=function(x, data,
    par.settings = solaR.theme,
    panel = panel.levelplot.raster, interpolate = TRUE
  ,
    xscale.components = xscale.solar,
    yscale.components = yscale.solar,
    ...){
    data0=getData(data)
    ind=data0$Dates
    data0$day=doy(ind)
    data0$month=month(ind)
    data0$year=year(ind)
    data0$w=h2r(hms(ind)-12)
    levelplot(x, data0,
      par.settings = par.settings,
      xscale.components = xscale.components,
      yscale.components = yscale.components,
      panel = panel, interpolate = interpolate,
      ...)
  }
)

setMethod('levelplot',
  signature=c(x='formula', data='Sol'),
  definition=function(x, data,
    par.settings = solaR.theme,
    panel = panel.levelplot.raster, interpolate = TRUE
  ,
    xscale.components = xscale.solar,
    yscale.components = yscale.solar,
    ...){
    data0=as.data.tableI(data, complete=TRUE, day=TRUE)
    ind=data0$Dates
    data0$day=doy(ind)
    data0$month=month(ind)
    data0$year=year(ind)
  }
)
```

```

        levelplot(x, data0,
                  par.settings = par.settings,
                  xscale.components = xscale.components,
                  yscale.components = yscale.components,
                  panel = panel, interpolate = interpolate,
                  ...)
    }
)

setMethod('levelplot',
signature=c(x='formula', data='G0'),
definition=function(x, data,
                  par.settings = solaR.theme,
                  panel = panel.levelplot.raster, interpolate = TRUE
,
                  xscale.components = xscale.solar,
                  yscale.components = yscale.solar,
                  ...){
    data0=as.data.tableI(data, complete=TRUE, day=TRUE)
    ind=data0$Dates
    data0$day=doy(ind)
    data0$month=month(ind)
    data0$year=year(ind)
    levelplot(x, data0,
              par.settings = par.settings,
              xscale.components = xscale.components,
              yscale.components = yscale.components,
              panel = panel, interpolate = interpolate,
              ...)
    }
)

```

EXTRACTO DE CÓDIGO A.50: *levelplot*

A.4.12. losses

```

setGeneric('losses', function(object){standardGeneric('losses')})

setMethod('losses',
signature=(object='Gef'),
definition=function(object){
    dat <- as.data.tableY(object, complete=TRUE)
    isShd=('Gef0d' %in% names(dat)) ##is there shadows?
    if (isShd) {
        shd <- with(dat, mean(1-Gefd/Gef0d))
        eff <- with(dat, mean(1-Gef0d/Gd))
    } else {
        shd <- 0
        eff <- with(dat, mean(1-Gefd/Gd))
    }
    result <- data.table(Shadows = shd, AoI = eff)
    result
}
)

setMethod('losses',
signature=(object='ProdGCPV'),
definition=function(object){

```

```

datY <- as.data.tableY(object, complete=TRUE)
module0=object@module
module0$CoefVT=0 ##No losses with temperature
Pg=object@generator$Pg
Nm=1/sample2Hours(object@sample)
datI <- as.data.tableI(object, complete=TRUE)
if (object@type=='prom'){
  datI[, DayOfMonth := DOM(datI)]
  YfDC0 <- datI[, sum(Vmpp*Imp/Pg*DayOfMonth, na.rm = TRUE),
    by = month(Dates)][[2]]
  YfDC0 <- sum(YfDC0, na.rm = TRUE)
  YfAC0 <- datI[, sum(Pdc*EffI/Pg*DayOfMonth, na.rm = TRUE),
    by = month(Dates)][[2]]
  YfAC0 <- sum(YfAC0, na.rm = TRUE)
} else {
  datI[, DayOfMonth := DOM(datI)]
  YfDC0 <- datI[, sum(Vmpp*Imp/Pg*DayOfMonth, na.rm = TRUE),
    by = year(Dates)][[2]]
  YfAC0 <- datI[, sum(Pdc*EffI/Pg*DayOfMonth, na.rm = TRUE),
    by = year(Dates)][[2]]
}
gen <- mean(1-YfDC0/datY$Gefd)
YfDC <- datY$Edc/Pg*1000
DC=mean(1-YfDC/YfDC0)
inv=mean(1-YfAC0/YfDC)
AC=mean(1-datY$Yf/YfAC0)
result0 <- losses(as(object, 'Gef'))
result1 <- data.table(Generator = gen,
  DC = DC,
  Inverter = inv,
  AC = AC)
result <- data.table(result0, result1)
result
}
)

###compareLosses

## compareLosses,ProdGCPV: no visible binding for global variable 'name'
if(getRversion() >= "2.15.1") globalVariables(c('name'))

setGeneric('compareLosses', signature='...', function(...){standardGeneric('
  compareLosses')})

setMethod('compareLosses', 'ProdGCPV',
  definition=function(...){
    dots <- list(...)
    nms0 <- substitute(list(...))
    if (!is.null(names(nms0))){ ##do.call
      nms <- names(nms0[-1])
    } else {
      nms <- as.character(nms0[-1])
    }
    foo <- function(object, label){
      yY <- losses(object)
      yY <- cbind(yY, name=label)
      yY
    }
  }

```



```

        cdata <- mapply(FUN=foo, dots, nms, SIMPLIFY=FALSE)
        z <- do.call(rbind, cdata)
        z <- melt(z, id.vars = 'name')
        p <- dotplot(variable~value*100, groups=name, data=z,
                     par.settings=solaR.theme, type='b',
                     auto.key=list(corner=c(0.95,0.2), cex=0.7), xlab='
Losses (%)')
        print(p)
        return(z)
    }
)

```

EXTRACTO DE CÓDIGO A.51: *losses*

A.4.13. mergeSolar

```

setGeneric('mergesolaR', signature='...', function(...){standardGeneric('
mergesolaR')})

fooMeteo <- function(object, var){yY <- getData(object)[, .SD,
                                                         by = Dates,
                                                         .SDcols = var]}

fooGO <- function(object, var){yY <- as.data.tableD(object)[, .SD,
                                                         by = Dates,
                                                         .SDcols = var]}

mergeFunction <- function(..., foo, var){
  dots <- list(...)
  dots <- lapply(dots, as, class(dots[[1]])) ##the first element is the one
  that dictates the class to everyone
  nms0 <- substitute(list(...))
  if (!is.null(names(nms0))){ ##do.call
    nms <- names(nms0[-1])
  } else {
    nms <- as.character(nms0[-1])
  }
  cdata <- sapply(dots, FUN=foo, var, simplify=FALSE)
  z <- cdata[[1]]
  for (i in 2:length(cdata)){
    z <- merge(z, cdata[[i]], by = 'Dates', suffixes = c("", paste0('.', i))
  )
  }
  names(z)[-1] <- nms
  z
}

setMethod('mergesolaR',
          signature='Meteo',
          definition=function(...){
            res <- mergeFunction(..., foo=fooMeteo, var='GO')
            res
          }
)

setMethod('mergesolaR',
          signature='GO',
          definition=function(...){

```

```

        res <- mergeFunction(..., foo=fooG0, var='G0d')
        res
    }
)

setMethod('mergesolaR',
  signature='Gef',
  definition=function(...){
    res <- mergeFunction(..., foo=fooG0, var='Gefd')
    res
  }
)

setMethod('mergesolaR',
  signature='ProdGCPV',
  definition=function(...){
    res <- mergeFunction(..., foo=fooG0, var='Yf')
    res
  }
)

setMethod('mergesolaR',
  signature='ProdPVPS',
  definition=function(...){
    res <- mergeFunction(..., foo=fooG0, var='Yf')
    res
  }
)

```

EXTRACTO DE CÓDIGO A.52: *mergeSolaR*A.4.14. *shadeplot*

```

setGeneric('shadeplot', function(x, ...)standardGeneric('shadeplot'))

setMethod('shadeplot', signature(x='Shade'),
  function(x,
    main='',
    xlab=expression(L[ew]),
    ylab=expression(L[ns]),
    n=9, ...){
    red=x@distances
    FS.loess=x@FS.loess
    Yf.loess=x@Yf.loess
    struct=x@struct
    mode=x@modeTrk
    if (mode=='two'){
      Lew=seq(min(red$Lew),max(red$Lew),length=100)
      Lns=seq(min(red$Lns),max(red$Lns),length=100)
      Red=expand.grid(Lew=Lew,Lns=Lns)
      FS=predict(FS.loess,Red)
      Red$FS=as.numeric(FS)
      AreaG=with(struct,L*W)
      GRR=Red$Lew*Red$Lns/AreaG
      Red$GRR=GRR
      FS.m<-matrix(1-FS,
        nrow=length(Lew),
        ncol=length(Lns))
    }
  }
)

```

```

GRR.m<-matrix(GRR,
              nrow=length(Lew),
              ncol=length(Lns))
niveles=signif(seq(min(FS.m),max(FS.m),l=n+1),3)
pruebaCB<-( "RColorBrewer" %in% .packages())
if (pruebaCB) {
  paleta=rev(brewer.pal(n, 'YlOrRd'))
} else {
  paleta=rev(heat.colors(n))}
par(mar=c(4.1,4.1,2.1,2.1))
filled.contour(x=Lew,y=Lns,z=FS.m,#...,
               col=paleta, #levels=niveles,
               nlevels=n,
               plot.title=title(xlab=xlab,
                                ylab=ylab, main=main),
               plot.axes={
                 axis(1);axis(2);
                 contour(Lew, Lns, FS.m,
                        nlevels=n, #levels=niveles,
                        col="black", labcex=.8, add=TRUE)
                 contour(Lew, Lns, GRR.m,
                        col="black", lty=3, labcex=.8, add=
TRUE)
                 grid(col="white",lty=3)},
               key.title=title("1-FS",cex.main=.8))
}
if (mode=='horiz') {
  Lew=seq(min(red$Lew),max(red$Lew),length=100)
  FS=predict(FS.loess,Lew)
  GRR=Lew/struct$L
  plot(GRR,1-FS,main=main,type='l',...)
  grid()
}
if (mode=='fixed'){
  D=seq(min(red$D),max(red$D),length=100)
  FS=predict(FS.loess,D)
  GRR=D/struct$L
  plot(GRR,1-FS,main=main,type='l',...)
  grid()
}
}
)

```

EXTRACTO DE CÓDIGO A.53: *shadeplot*

A.4.15. window

```

setMethod('[',
  signature='Meteo',
  definition=function(x, i, j,...){
    if (!missing(i)) {
      i <- truncDay(i)
    } else {
      i <- indexD(x)[1]
    }
    if (!missing(j)) {
      j <- truncDay(j)+86400-1 ##The end is the last second of the day
    } else {
      nDays <- length(indexD(x))
      j <- indexD(x)[nDays]+86400-1
    }
  }
)

```

```

    }
    stopifnot(j>i)
    if (!is.null(i)) i <- truncDay(i)
    if (!is.null(j)) j <- truncDay(j)+86400-1
    d <- indexD(x)
    x@data <- x@data[(d >= i & d <= j)]
    x
  }
)

setMethod('[',
  signature='Sol',
  definition=function(x, i, j, ...){
    if (!missing(i)) {
      i <- truncDay(i)
    } else {
      i <- indexD(x)[1]
    }
    if (!missing(j)) {
      j <- truncDay(j)+86400-1##The end is the last second of the
day
    } else {
      nDays <- length(indexD(x))
      j <- indexD(x)[nDays]+86400-1
    }
    stopifnot(j>i)
    if(!is.null(i)) i <- truncDay(i)
    if(!is.null(j)) j <- truncDay(j)
    d1 <- indexD(x)
    d2 <- indexI(x)
    x@solD <- x@solD[(d1 >= i & d1 <= j)]
    x@solI <- x@solI[(d2 >= i & d2 <= j)]
    x
  }
)

setMethod('[',
  signature='GO',
  definition=function(x, i, j, ...){
    sol <- as(x, 'Sol')[i=i, j=j, ...] ##Sol method
    meteo <- as(x, 'Meteo')[i=i, j=j, ...] ##Meteo method
    i <- indexI(sol)[1]
    j <- indexI(sol)[length(indexI(sol))]
    d1 <- indexD(x)
    d2 <- indexI(x)
    GOIw <- x@GOI[(d2 >= i & d2 <= j)]
    Taw <- x@Ta[(d2 >= i & d2 <= j)]
    G0dw <- x@G0D[(d1 >= truncDay(i) & d1 <= truncDay(j))]
    G0dmw <- G0dw[, lapply(.SD/1000, mean, na.rm= TRUE),
      .SDcols = c('G0d', 'D0d', 'B0d'),
      by = .(month(Dates), year(Dates))]
    if (x@type=='prom'){
      G0dmw[, DayOfMonth := DOM(G0dmw)]
      G0yw <- G0dmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
        .SDcols = c('G0d', 'D0d', 'B0d'),
        by = .(Dates = year)]
      G0dmw[, DayOfMonth := NULL]
    }
  }
)

```

```

    } else {
      GOyw <- G0dw[, lapply(.SD/1000, sum, na.rm = TRUE),
                          .SDcols = c('G0d', 'D0d', 'B0d'),
                          by = .(Dates = year(unique(truncDay(Dates))))]
    }
    G0dmw[, Dates := paste(month.abb[month], year, sep = '. ')]
    G0dmw[, c('month', 'year') := NULL]
    setcolororder(G0dmw, 'Dates')
    result <- new('G0',
                  meteo,
                  sol,
                  GOD=G0dw,
                  G0dm=G0dmw,
                  GOy=GOyw,
                  GOI=GOIw,
                  Ta=Taw)

    result
  }
)

setMethod('[',
signature='Gef',
definition=function(x, i, j, ...){
  g0 <- as(x, 'G0')[i=i, j=j, ...] ##G0 method
  i <- indexI(g0)[1]
  j <- indexI(g0)[length(indexI(g0))]
  d1 <- indexD(x)
  d2 <- indexI(x)
  GefIw <- x@GefI[(d2 >= i & d2 <= j)]
  Thetaw <- x@Theta[(d2 >= i & d2 <= j)]
  Gefdw <- x@GefD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
  nms <- c('Bod', 'Bnd', 'Gd', 'Dd',
           'Bd', 'Gefd', 'Defd', 'Befd')
  Gefdmw <- Gefdw[, lapply(.SD/1000, mean, na.rm = TRUE),
                  .SDcols = nms,
                  by = .(month(Dates), year(Dates))]
  if (x@type=='prom'){
    Gefdmw[, DayOfMonth:= DOM(Gefdmw)]
    Gefyw <- Gefdmw[, lapply(.SD*DayOfMonth, sum),
                    .SDcols = nms,
                    by = .(Dates = year)]
    Gefdmw[, DayOfMonth := NULL]
  } else {
    Gefyw <- Gefdw[, lapply(.SD/1000, sum, na.rm = TRUE),
                    .SDcols = nms,
                    by = .(Dates = year)]
  }
  Gefdmw[, Dates := paste(month.abb[month], year, sep = '. ')]
  Gefdmw[, c('month', 'year') := NULL]
  setcolororder(Gefdmw, 'Dates')
  result <- new('Gef',
                g0,
                GefD=Gefdw,
                Gefdm=Gefdmw,
                Gefy=Gefyw,
                GefI=GefIw,
                Theta=Thetaw,

```

```

        iS=x@iS,
        alb=x@alb,
        modeTrk=x@modeTrk,
        modeShd=x@modeShd,
        angGen=x@angGen,
        struct=x@struct,
        distances=x@distances
    )

    result
}
)

setMethod('[',
signature='ProdGCPV',
definition=function(x, i, j, ...){
    gef <- as(x, 'Gef')[i=i, j=j, ...] ##Gef method
    i <- indexI(gef)[1]
    j <- indexI(gef)[length(indexI(gef))]
    d1 <- indexD(x)
    d2 <- indexI(x)
    prodIw <- x@prodI[(d2 >= i & d2 <= j)]
    prodDw <- x@prodD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
    prodDmw <- prodDw[, lapply(.SD/1000, mean, na.rm = TRUE),
                          .SDcols = c('Eac', 'Edc'),
                          by = .(month(Dates), year(Dates))]
    prodDmw$Yf <- prodDw$Yf
    if (x@type=='prom'){
        prodDmw[, DayOfMonth := DOM(prodDmw)]
        prodyw <- prodDmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                          .SDcols = c('Eac', 'Edc', 'Yf'),
                          by = .(Dates = year)]
        prodDmw[, DayOfMonth := NULL]
    } else {
        prodyw <- prodDw[, lapply(.SD/1000, sum, na.rm = TRUE),
                          .SDcols = c('Eac', 'Edc', 'Yf'),
                          by = .(Dates = year)]
    }
    prodDmw[, Dates := paste(month.abb[month], year, sep = '. ')]
    prodDmw[, c('month', 'year') := NULL]
    setcolorder(prodDmw, c('Dates', names(prodDmw)[-length(prodDmw)]))
    result <- new('ProdGCPV',
                  gef,
                  prodD=prodDw,
                  prodDm=prodDmw,
                  prody=prodyw,
                  prodI=prodIw,
                  module=x@module,
                  generator=x@generator,
                  inverter=x@inverter,
                  effSys=x@effSys
    )

    result
}
)

setMethod('[',
signature='ProdPVPS',

```

```

definition=function(x, i, j, ...){
  gef <- as(x, 'Gef')[i=i, j=j, ...] ##Gef method
  i <- indexI(gef)[1]
  j <- indexI(gef)[length(indexI(gef))]
  d1 <- indexD(x)
  d2 <- indexI(x)
  prodIw <- x@prodI[(d2 >= i & d2 <= j)]
  prodDw <- x@prodD[(d1 >= truncDay(i) & d1 <= truncDay(j))]
  prodDmw <- prodDw[, .(Eac = Eac/1000,
                        Qd = Qd,
                        Yf = Yf),
                      by = .(month(Dates), year(Dates))]
  if (x@type=='prom'){
    prodDmw[, DayOfMonth := DOM(prodDmw)]
    prodyw <- prodDmw[, lapply(.SD*DayOfMonth, sum, na.rm = TRUE),
                        .SDcols = c('Eac', 'Qd', 'Yf'),
                        by = .(Dates = year)]
    prodDmw[, DayOfMonth := NULL]
  } else {
    prodyw <- prodDw[, .(Eac = sum(Eac, na.rm = TRUE)/1000,
                        Qd = sum(Qd, na.rm = TRUE),
                        Yf = sum(Yf, na.rm = TRUE)),
                      by = .(Dates = year)]
  }
  prodDmw[, Dates := paste(month.abb[month], year, sep = '. ')]
  prodDmw[, c('month', 'year') := NULL]
  setcolorder(prodDmw, c('Dates', names(prodDmw)[-length(prodDmw)]))
  result <- new('ProdPVPS',
                gef,
                prodD=prodDw,
                prodDm=prodDmw,
                prody=prodyw,
                prodI=prodIw,
                pump=x@pump,
                H=x@H,
                Pg=x@Pg,
                converter=x@converter,
                effSys=x@effSys
                )
  result
}
)

```

EXTRACTO DE CÓDIGO A.54: *window*

A.4.16. writeSolar

```

setGeneric('writeSolar', function(object, file,
                                   complete=FALSE, day=FALSE,
                                   timeScales=c('i', 'd', 'm', 'y'), sep=',',
                                   ...){
  standardGeneric('writeSolar')})

setMethod('writeSolar', signature=(object='Sol'),
          definition=function(object, file, complete=FALSE, day=FALSE,
                              timeScales=c('i', 'd', 'm', 'y'), sep=',', ...){
    name <- strsplit(file, '\\.')[[1]][1]
    ext <- strsplit(file, '\\.')[[1]][2]

```

```

timeScales <- match.arg(timeScales, several.ok=TRUE)
if ('i' %in% timeScales) {
  zI <- as.data.tableI(object, complete=complete, day=day)
  write.table(zI,
              file=file, sep=sep, row.names = FALSE, ...)
}
if ('d' %in% timeScales) {
  zD <- as.data.tableD(object, complete=complete, day = day)
  write.table(zD,
              file=paste(name, 'D', ext, sep='.'),
              sep=sep, row.names = FALSE, ...)
}
if ('m' %in% timeScales) {
  zM <- as.data.tableM(object, complete=complete, day = day)
  write.table(zM,
              file=paste(name, 'M', ext, sep='.'),
              sep=sep, row.names = FALSE, ...)
}
if ('y' %in% timeScales) {
  zY <- as.data.tableY(object, complete=complete, day = day)
  write.table(zY,
              file=paste(name, 'Y', ext, sep='.'),
              sep=sep, row.names = FALSE, ...)
}
})

```

EXTRACTO DE CÓDIGO A.55: *writeSolar*

A.4.17. xyplot

```

#####
## THEMES
#####
xscale.solar <- function(...){ans <- xscale.components.default(...); ans$top=
  FALSE; ans}
yscale.solar <- function(...){ans <- yscale.components.default(...); ans$right=
  FALSE; ans}

solaR.theme <- function(pch=19, cex=0.7, region=rev(brewer.pal(9, 'YlOrRd')),
  ...) {
  theme <- custom.theme.2(pch=pch, cex=cex, region=region, ...)
  theme$strip.background$col='transparent'
  theme$strip.shingle$col='transparent'
  theme$strip.border$col='transparent'
  theme
}

solaR.theme.2 <- function(pch=19, cex=0.7, region=rev(brewer.pal(9, 'YlOrRd')),
  ...) {
  theme <- custom.theme.2(pch=pch, cex=cex, region=region, ...)
  theme$strip.background$col='lightgray'
  theme$strip.shingle$col='lightgray'
  theme
}

#####
## XYPLOT
#####

```



```

setGeneric('xyplot')

setMethod('xyplot',
  signature = c(x = 'data.frame', data = 'missing'),
  definition = function(x, data,
    par.settings = solaR.theme.2,
    xscale.components=xscale.solar,
    yscale.components=yscale.solar,
    scales = list(y = 'free'),
    ...){
    N <- length(x)-1
    x0 <- x[, lapply(.SD, as.numeric), by = Dates]
    x0 <- melt(x0, id.vars = 'Dates')
    x0$variable <- factor(x0$variable,
      levels = rev(levels(factor(x0$variable))))
    xyplot(value ~ Dates | variable, x0,
      par.settings = par.settings,
      xscale.components = xscale.components,
      yscale.components = yscale.components,
      scales = scales,
      type = 'l', layout = c(1,N),
      ...)
  })

setMethod('xyplot',
  signature=c(x='formula', data='Meteo'),
  definition=function(x, data,
    par.settings=solaR.theme,
    xscale.components=xscale.solar,
    yscale.components=yscale.solar,
    ...){
    data0=getData(data)
    xyplot(x, data0,
      par.settings = par.settings,
      xscale.components = xscale.components,
      yscale.components = yscale.components,
      strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
  }
)

setMethod('xyplot',
  signature=c(x='formula', data='Sol'),
  definition=function(x, data,
    par.settings=solaR.theme,
    xscale.components=xscale.solar,
    yscale.components=yscale.solar,
    ...){
    data0=as.data.tableI(data, complete=TRUE, day=TRUE)
    data0[, w := h2r(hms(Dates)-12)]
    xyplot(x, data0,
      par.settings = par.settings,
      xscale.components = xscale.components,
      yscale.components = yscale.components,
      strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
  }
)

setMethod('xyplot',

```

```

signature=c(x='formula', data='GO'),
definition=function(x, data,
                    par.settings=solaR.theme,
                    xscale.components=xscale.solar,
                    yscale.components=yscale.solar,
                    ...){
  data0=as.data.tableI(data, complete=TRUE, day=TRUE)
  xyplot(x, data0,
        par.settings = par.settings,
        xscale.components = xscale.components,
        yscale.components = yscale.components,
        strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
}
)

setMethod('xyplot',
signature=c(x='formula', data='Shade'),
definition=function(x, data,
                    par.settings=solaR.theme,
                    xscale.components=xscale.solar,
                    yscale.components=yscale.solar,
                    ...){
  data0=as.data.table(data)
  xyplot(x, data0,
        par.settings = par.settings,
        xscale.components = xscale.components,
        yscale.components = yscale.components,
        strip = strip.custom(strip.levels=c(TRUE, TRUE)), ...)
}
)

setMethod('xyplot',
signature=c(x='Meteo', data='missing'),
definition=function(x, data,
                    ...){
  x0=getData(x)
  xyplot(x0,
        scales=list(cex=0.6, rot=0, y='free'),
        strip=FALSE, strip.left=TRUE,
        par.strip.text=list(cex=0.6),
        ylab = '',
        ...)
}
)

setMethod('xyplot',
signature=c(x='GO', data='missing'),
definition=function(x, data, ...){
  x0 <- as.data.tableD(x, complete=FALSE)
  x0 <- melt(x0, id.vars = 'Dates')
  xyplot(value~Dates, x0, groups = variable,
        par.settings=solaR.theme.2,
        xscale.components=xscale.solar,
        yscale.components=yscale.solar,
        superpose=TRUE,
        auto.key=list(space='right'),
        ylab='Wh/m\u00b2',
        type = 'l',

```

```

        ...
    }
)

setMethod('xyplot',
  signature=c(x='ProdGCPV', data='missing'),
  definition=function(x, data, ...){
    x0 <- as.data.tableD(x, complete=FALSE)
    xyplot(x0,
      strip = FALSE, strip.left = TRUE,
      ylab = '', ...)
  }
)

setMethod('xyplot',
  signature=c(x='ProdPVPS', data='missing'),
  definition=function(x, data, ...){
    x0 <- as.data.tableD(x, complete=FALSE)
    xyplot(x0,
      strip = FALSE, strip.left = TRUE,
      ylab = '', ...)
  }
)

```

EXTRACTO DE CÓDIGO A.56: *xyplot*

A.5. Conjunto de datos

A.5.1. aguiar

```
data(MTM)
Ktlim
```

EXTRACTO DE CÓDIGO A.57: *aguiar₁*

```

      [,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

```

```
Ktmtm
```

EXTRACTO DE CÓDIGO A.58: *aguiar₂*

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

```
head(MTM)
```

EXTRACTO DE CÓDIGO A.59: *aguiar₃*

```

      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

```

data(SIAR)
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	-0.8844444444	38.67639	519	1999-11-09	2000-03-19
2:	Camp de Mirra	A02	-0.7727777778	38.67917	589	1999-11-09	<NA>
3:	Vila Joiosa	A03	-0.2561111111	38.52778	73	1999-11-10	<NA>
4:	Ondara	A04	0.0063888889	38.81833	38	1999-11-10	<NA>
5:	Dénia Gata	A05	0.0825000000	38.79250	86	1999-11-15	<NA>
6:	Pinoso	A06	-1.0605555556	38.42722	629	1999-11-14	<NA>

A.5.3. helios

```

data(helios)
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

```

data(prodEx)
head(prodEx)

```

EXTRACTO DE CÓDIGO A.62: *prodEx*

	Dates	1	2	3	4	5	6	7
	<Date>	<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
2:	2007-07-03	8.710291	8.691992	8.655395	8.710291	8.737740	8.792637	8.774338
3:	2007-07-04	8.746889	8.737740	8.865832	8.737740	8.765188	8.838384	8.810935
4:	2007-07-05	8.280266	8.271117	8.408359	8.280266	8.344313	8.380911	8.353462
5:	2007-07-06	8.399209	8.417508	8.509003	8.435807	8.490704	8.490704	8.499854
6:	2007-07-07	8.197921	8.170473	8.335163	8.225370	8.243669	8.307715	8.298565

	8	9	10	11	12	13	14	15
	<num>	<num>	<num>	<num>	<num>	<num>	<num>	<num>
1:	8.948177	8.948177	8.984775	8.783487	8.865832	8.966476	8.884131	8.774338
2:	8.774338	8.746889	8.801786	8.545601	8.682843	8.774338	8.691992	8.591348
3:	8.792637	8.801786	8.829234	8.545601	8.618797	8.829234	8.719441	8.618797
4:	8.362612	8.316864	8.380911	8.179622	8.271117	8.353462	8.280266	8.207071
5:	8.527302	8.472405	8.509003	8.316864	8.426658	8.490704	8.435807	8.344313
6:	8.280266	8.243669	8.326014	8.152174	8.161323	8.316864	8.234519	8.143024
	16	17	18	19	20	21	22	
	<num>	<num>	<num>	<num>	<num>	<num>	<num>	
1:	8.829234	8.627946	8.911580	8.807886	6.505270	3.742131	3.980018	
2:	8.646245	8.426658	8.710291	8.563900	3.952569	4.080662	3.238911	
3:	8.664544	8.426658	8.728590	8.612697	6.331430	1.363270	1.043039	
4:	8.261968	8.188772	7.950886	8.222320	5.498829	3.998316	2.461206	
5:	8.408359	8.371761	8.463256	8.332113	6.551017	5.361587	4.959010	
6:	8.179622	8.170473	8.243669	8.161323	6.669960	5.215195	4.922413	

A.5.5. pumpCoef

```
data(pumpCoef)
head(pumpCoef)
```

EXTRACTO DE CÓDIGO A.63: *pumpCoef*

	Qn	stages	Qmax	Pmn	a	b	c	g	h	i	j
	<int>	<int>	<num>	<int>	<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
2:	2	9	2.6	370	0.02114604	0.027864	-5.4486	-0.32	0.74	0.22	-0.1614
3:	2	13	2.6	550	0.03054428	0.040248	-7.8702	-0.12	0.49	0.27	-0.1614
4:	2	18	2.6	750	0.04229208	0.055728	-10.8972	-0.16	0.42	0.47	-0.1614
5:	2	23	2.6	1100	0.05403988	0.071208	-13.9242	-0.20	0.51	0.42	-0.1614
6:	2	28	2.6	1500	0.06578768	0.086688	-16.9512	-0.24	0.50	0.49	-0.1614
	k	l									
	<num>	<num>									
1:	0.5247	0.0694									
2:	0.5247	0.0694									
3:	0.5247	0.0694									
4:	0.5247	0.0694									
5:	0.5247	0.0694									
6:	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.

- [Uni20] European Union. *NextGenerationEU*. 2020. URL: https://next-generation-eu.europa.eu/index_es.
- [BOE22a] BOE. *Real Decreto-ley 10/2022, de 13 de mayo, por el que se establece con carácter temporal un mecanismo de ajuste de costes de producción para la reducción del precio de la electricidad en el mercado mayorista*. 2022. URL: <https://www.boe.es/buscar/act.php?id=BOE-A-2022-7843>.
- [BOE22b] BOE. *Real Decreto-ley 6/2022, de 29 de marzo, por el que se adoptan medidas urgentes en el marco del Plan Nacional de respuesta a las consecuencias económicas y sociales de la guerra en Ucrania*. 2022. URL: <https://www.boe.es/buscar/doc.php?id=BOE-A-2022-4972>.
- [dem22] Ministerio para transición ecológica y el reto demográfico. *Plan + Seguridad Energética*. 2022. URL: <https://www.miteco.gob.es/es/ministerio/planes-estrategias/seguridad-energetica.html#planSE>.
- [Eur22] Consejo Europeo. *REPowerEU*. 2022. URL: <https://www.consilium.europa.eu/es/policies/eu-recovery-plan/repowereu/>.
- [Hac22] Ministerio de Hacienda. *Mecanismo de Recuperación y Resiliencia*. 2022. URL: <https://www.hacienda.gob.es/es-ES/CDI/Paginas/FondosEuropeos/Fondos-relacionados-COVID/MRR.aspx>.
- [Mer+23] Olaf Mersmann et al. *microbenchmark: Accurate Timing Functions*. Proporciona infraestructura para medir y comparar con precisión el tiempo de ejecución de las expresiones de R. 2023. URL: <https://github.com/joshuaulrich/microbenchmark>.
- [Min23] pesca y alimentación Ministerio de agricultura. *Sistema de Información Agroclimática para el Regadío*. 2023. URL: <https://servicio.mapa.gob.es/websiar/>.
- [Per23] O. Perpiñán. *Energía Solar Fotovoltaica*. 2023. URL: <https://oscarperpinan.github.io/esf/>.
- [R C23] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: <https://www.R-project.org/>.
- [UNE23] UNEF. “Fomentando la biodiversidad y el crecimiento sostenible”. En: *Informe anual UNEF* (2023). URL: <https://www.unef.es/es/recursos-informes?idMultimediaCategoria=18>.
- [Wan+23] Chris Wanstrath et al. *GitHub*. 2023. URL: <https://github.com/>.
- [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/>.
- [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>.