Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DATA ANALYTICS EN ENERGÍAS RENOVABLES: ORGANIZACIÓN Y ANÁLISIS DE MEDIDAS SATELITALES DE RADIACIÓN

Autor: Alejandro Catalina Feliú
Tutor: José Ramón Dorronsoro Ibero

Julio 2016

DATA ANALYTICS ON RENEWABLE ENERGIES: ORGANIZATION AND ANALYSIS OF RADIATION SATELLITE MEASURES

Author: Alejandro Catalina Feliú
Advisor: José Ramón Dorronsoro Ibero

Cátedra UAM – IIC en Modelado y Predicción
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2016

Abstract

The renewable energies are a promising clean and sustainable energy form for the near future. With the latest oil crisis, top energy companies have shift their interest to these new energies. Specifically, when speaking of wind power and solar energy, Spain has a promising future. A fundamental factor to make these energies a true option is the computer science, and more specifically, the data science. This branch of the computer science is the responsible for making high accuracy forecasting models, which are essential to maximize the energy production and minimize the installation costs that would take place, for instance, when installing a solar farm in a location with few sun hours.

This work is an introductory analysis of radiation satellite data, leading towards building machine learning models that would predict photovoltaic energy production in the Iberian Peninsula. Previous to the selection of such data, we have studied several global satellite agencies, analyzing their main properties, services and products, putting a special emphasis in time and spatial resolution. This first study suggests the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT) as a suitable data source for our purposes.

The products we downloaded from EUMETSAT consist on cloud mask plus eleven channels covering the light spectrum from wavelength (measured in micrometers) 0.6 µm to 13.4 µm, including three visible channels (capturing light at the visible band of the light spectrum) and eight infrared channels (channels where the Earth thermal radiation is dominant, located at the infrared band of the light spectrum). From these channels we can compute the *reflectance* (only for visible channels) and the *brightness temperature* (only for infrared channels), ending with twenty-three variables; eleven radiances, three reflectances and eight brightness temperatures plus the cloud mask. A first analysis of the data shows that there is a high correlation between the radiation captured in bands 0.8 µm, 1.6 µm and 3.9 µm and the photovoltaic energy production.

As a conclusion, we have performed a first linear regression model with the aggregated values of each variable over the Iberian Peninsula, having as target the photovoltaic energy production. This model's results showed a Mean Absolute Error (MAE) of 6.02%, which encourages us to build and research more advanced machine learning models.

Key words

Data Analysis, Radiation Data, Satellite Data, EUMETSAT, Photovoltaic Solar Energy

Resumen

Las energías renovables son una prometedora fuente de energía limpia y sostenible para el futuro cercano. Desde las últimas crisis del petróleo, las grandes compañías energéticas han mostrado un creciente interés por estas energías renovables. España, en concreto, tiene unas características que la hacen especialmente interesante para la energía eólica y solar. Un factor fundamental que ha hecho de estas energías una opción factible es la ciencia computacional, y, más concretamente, la ciencia de datos. Esta rama de la informática es la encargada del diseño de modelos de predicción cada vez más precisos, que son esenciales para maximizar la producción de energía y minimizar los sobrecostes, derivados por ejemplo de la instalación de huertos solares en zonas que no gozan de muchas horas de sol.

Este trabajo es un análisis introductorio de medidas satelitales de radiación, poniendo énfasis en la construcción futura de modelos de predicción de energía solar fotovoltaica en la Península Ibérica. Previamente a la selección de estos datos, hemos realizado un estudio de las principales agencias que trabajan con datos satelitales, analizando algunas de sus propiedades más importantes, servicios y productos, poniendo un énfasis especial en la resolución espacial y temporal. Este primer estudio sugiere el Centro Europeo para la Explotación de Satélites Meteorológicos (EUMETSAT por sus siglas en inglés) como fuente de datos para nuestro trabajo.

Los conjuntos de datos descargados de EUMETSAT consisten en un índice de nubosidad y once canales que cubren el espectro de la luz (medido en micrómetros) desde los 0.6 µm a los 13.4 µm, incluyendo tres canales visibles y ocho canales infrarrojos. Desde la información de estos canales podemos computar la reflectancia (solo en canales visibles) y la temperatura del brillo (solo en canales infrarrojos), además de la radiación en sí misma. Un primer análisis de estos datos muestra que existe una alta correlación entre la radiación capturada en las bandas 0.8 µm, 1.6 µm y 3.9 µm y la producción global de energía fotovoltaica.

Como conclusión, hemos realizado un primer modelo de regresión lineal con los valores de cada variable acumulados para toda la Península, teniendo como objetivo la producción de energía fotovoltaica. Los resultados de este modelo han mostrado un Error Medio Absoluto (MAE de sus siglas en inglés) del 6.02%, lo que supone un importante factor de cara a investigar modelos de aprendizaje estadístico más avanzados.

Palabras Clave

Data Analysis, Datos de radiación, Datos satelitales, EUMETSAT, Energía Solar Fotovoltaica

Acknowledgments

First of all, I would like to thank my advisor, José Ramón Dorronsoro, for giving me the opportunity to work on this exciting and innovative project, and, of course, for all the time invested directing and reviewing this work. His support and guidance have made me take the first step in the research world, which has been an inflection point in my career.

Moreover, this work has been done with support from the Cátedra UAM–IIC en Ciencia de Datos y Aprendizaje Automático within its activities on innovation in the area of renewable energy. Thanks also to the Centro de Computación Científica (CCC) at UAM for using its computing facilities and to Red Eléctrica de España for kindly supplying photovoltaic energy production data. We would also thank EUMETSAT for their support and work making available the data we used in this work and for being constantly improving the overall quality of their services and products.

Moreover, and not less, I would like to thank my partner, Moira López, who has been a continuous and essential support. She, more than anyone, knows how much work I've put to make this true. In a special place I would also like to thank my friends, specially Pablo Sánchez, Guillermo Sarasa and Sergio Sanz, with whom I've spent uncountable hours in these labs, working in so many practices along these years.

Finally, a really special thanks to my family, who has always been an irreplaceable support, making it possible for me to become who I am now.

Contents

Figures index

Tables index

Glossary

- **BUFR**: Binary Universal Form for the Represen- tation of meteorological data.
- **GRIB**: GRIdded Binary. Widely used meteorological format.
- **IR**: Infrared. Referred to as the infrared spectrum of the light.
- **MAE**: Mean Absolute Error.
- **MFG**: Meteosat First Generation.
- **MSG**: Meteosat Second Generation.
- **MTG**: Meteosat Third Generation.
- **MYP**: Modelado y Predicción. Format developed by Cátedra UAM–IIC en Modelado y Predicción.
- **NN**: Neural Network.
- **SEVIRI**: Spinning Enhanced Visible and Infrared Imager. Satellite instrumentation on board of the Meteosat Second Generation.
- **SVR**: Support Vector Regression.
- **VIS**: Visible. Referred to as the visible spectrum of the light.

1 Introduction

1.1 Motivation of the Project

The several oil crisis that occurred during the 70s (the first in 1973 and the second later in 1979), provoked a real revolution in the way the world thought about energy. Some of the main energy companies started researching new energy forms that would allow us to sustain the modern society. These new sources are the renewable energies, namely, wind power, solar energy, hydropower, geothermal and bio energy. This revolution has allowed the most advanced countries to start providing a substantial amount of the total energy with these new and clean sources, reducing the dependency on oil energies. In Spain, Red Eléctrica de España (REE) states that the 36.9% of the total energy demanded is provided with wind power (28%) and photovoltaic (solar energy) (9%) [1]. A fundamental factor to make these energies a real option is the computer science, and, more specifically, the data science.

The data science is the branch of the computer science that involves the analysis of large amount of data in order to build models from which extract patterns and predictions. These analysis are essential when it comes to place wind farms or photovoltaic power stations in the best possible locations and, therefore, maximizing the energy production, among other important applications, such as predicting future energy productions.

These analysis are based on correlations between variables and machine learning (ML) forecasts, which are calculated by mathematical models and algorithms. As we may expect, in order to produce valid results in terms of predictions, these models would need a large amount of data, otherwise the results might be biased by wrong measures. There are two main data sources that offer this kind of data:

- Ground station's data.
- Satellite's data.

The innovation and investment in satellites have made this the preferred source for solar energy related data because of several factors. On one hand, satellites are able to capture data of the upper layers of the atmosphere, which contains useful information about the incoming radiation to the Earth. On the other hand, the high technological devices that capture the

data are constantly improving and are able to offer global coverage and good time and spatial resolution. For instance, the Meteosat satellites offer spatial resolution up to 0.03 degrees every 15 minutes, which is much broader than those extracted from isolated ground stations' measures.

Summarizing, the motivation of the project is twofold. On one side, we will be working with satellite's data, an innovating data source with a good fit on the meteorology field. On the other side, we will be studying photovoltaic solar energy, a promising clean and efficient energy source, for which Spain is a suitable location in terms of sun hours all year long. To combine these two fields, we will introduce ourselves to the first steps in data science, which is probably one of the most growing areas in the computer science, with a very wide range of applications on several fields, such as finance, meteorology, biology or neuroscience.

1.2 Objectives and Approach

The main objective of this work is to get an introductory understanding of data science applied to solar energy data. The different developed methods and inner structures will be organized as a software platform, which will be the base system for building machine learning models in future work.

In order to achieve this, the following sub-objectives have been established:

- Identify a suitable satellite data source that offers solar energy related products, such as radiation, cloud cover indices, etc.
- Design an storage system to hold the data in a structured way.
- Perform a data analysis of the first selected products to understand (at an introductory level) the meaning of the variables involved. Build a first machine learning model to study its behavior predicting photovoltaic energy production.
- Develop inner data formats to shape the original raw data in a suitable way to apply machine learning models, which will result in our final software platform.

1.3 Document Structure

The objectives and approach discussed in the previous section are translated into the following document structure:

- **State of Art**. In the first place, we are going to study some of the organizations that are currently working with satellites' data. This study will help us to identify the organization that best fits our needs and requirements.
- **Analysis of EUMETSAT Products**. In this chapter, we will explore this selected organization in more detail, analyzing the solar energy related products they offer, along their main properties.
- **Data Interpretation and First Application**. This chapter will show a first data analysis of the selected data, ending up with a first machine learning model and a discussion on its results as a first application on photovoltaic solar energy.

Finally, the Appendices will show additional information and more detailed descriptions about different sections of the document. In particular, the code and description of the resulting software platform and several other code snippets will be presented in Appendixes A and B.

2

Satellite Data Sources. State of Art.

2.1 Introduction

This first chapter will introduce the current state of art of some of the most widely known satellite data sources. Along the chapter we will focus on the main features these sources provide us, such as time and release frequency, spatial resolution, data access and some other characteristics that might be interesting for the purposes of this work.

The data sources included in our work are the NOAA and EUMETSAT, including some of the EUMETSAT–dependent data processing centers, which we will describe in independent sections, to get a better understanding of them and of what they offer.

Our final objective is to apply data analysis methods on satellite extracted data, in the field of photovoltaic solar energy, which leads us to require several properties in our data sources, summarized as follows.

- First of all, we need the data to be focused on Europe, and mainly on the Iberian Peninsula.
- We also need these data to have a good enough spatial resolution to have a reasonable level of detail in the area we want to study.
- Finally, a high temporal resolution will be essential for this work. Hourly data will allow us to properly study the evolution of the measures and the phenomena in which we will focus on.

Nonetheless, some of the sources we have studied have not met all the requirements and so have not been included as final sources in this work. This will be one of the considerations we will make all throughout this chapter: why and based on which criteria we have decided whether to include a source or not.

To start with, we will describe each studied source and its most relevant properties, including data access, formats and main products.

2.2 NOAA

2.2.1 Source Description

The NOAA (standing for National Oceanic and Atmospheric Administration) is an American scientific agency dependent from the Department of Commerce of the USA. Among its most important duties is the study of the meteorological conditions not just over the USA but over the whole world. Due to its great data distribution service and great coverage, it is known as one of the main agencies.

2.2.2 Contents

The department working on the satellite monitorization provide us with data about climate change, tropical storms, ocean temperature, atmosphere composition or incoming radiation, among several other NOAA services.

The NOAA satellites can be divided into the following categories, depending on the area each one focuses on, which we describe in the next subsections.

1. Joint Polar Satellite System
2. Geostationary Operational Environmental Satellites
3. Defense Meteorological Satellite

Joint Polar Satellite System

The Joint Polar Satellite System (JPSS) is the next generation of orbital satellites the USA wants to set up. This kind of satellites comes from a collaborative effort between the NOAA and NASA, adding important technological advances.

These satellites are orbiting from one pole of the planet to the other, working with the European satellites to give full cover of the phenomena happening around the world, such as atmospheric or oceanic ones, or those that happen at the surface level.

Besides, they are one of the few national satellite program currently in production that can ensure continuity of observational data at the quality levels needed to keep the current weather forecasts beyond 2017, which makes them an interesting program to take into account. They maintain five kinds of satellites and one experimental payload, named Suomi NPP, JPSS-1, JPSS-2, JPSS-3, JPSS-4 and TCTE (which stands for TSI Calibration Transfer Experiment). The only one currently running is the Suomi NPP, while the JPSS satellites are scheduled to be launched in 2017, 2021, 2026 and 2031, respectively.

Given the success of the Suomi NPP program, the equipment these satellites will have is based on the following instrumentation systems:

- ATMS (Advanced Technology Microwave Sounder),
- Cross-track Infrared Sounder,
- Visible Infrared Imaging Radiometer Suite (VIIRS),
- Ozone Mapping and Profiler Suite (OMPS) and

- Clouds and the Earth's Radiant Energy System (CERES).

This instrumentation will allow them to be high technological devices with great coverage.

Geostationary Operational Environmental Satellites

The Geostationary Operational Environmental Satellites (GOES) is a set of geostationary satellites, which means that their orbital speed around the Earth is the same as the Earth's around itself, so it keeps always the same relative position. Thus, their duty is to monitorize the climate situation over the USA all day long.

They take measures and JPEG pictures of America (focusing on North America) every 15 minutes, capturing information about the atmospheric events happening, from clouds generation, ocean's temperature, tropical storms, fog, wildfires, and some other atmospheric characteristics and phenomena.

Defense Meteorological Satellite Program

The Defense Meteorological Satellite Program (DMSP) is a program launched in the 60s with the purpose of supporting military actions and missions.

These satellites are orbiting at a relative low height, being no higher than 830 km, with a period of 101 minutes. Their duties focus on studying the weather conditions, such as clouds' properties or snow, as well as fires and pollution. Moreover, its night vision devices allow them to see at night, being extremely useful for military actions, and detect radiation-emitting bodies such as clouds illuminated by the moon.

This service is coordinated by the NOAA, supervised by the Defense Department of the USA, and closely working with the Air Force.

2.2.3 Data Organization

The NOAA's department we are interested in is the National Centers For Environmental Information (NCEI), previously named National Climate Data Center (NCDC). This department focuses on studying the climate conditions through satellites' data.

The access to these data is organized among several categories:

- **Access by Dataset**: This section allows us to access datasets classified by topic, such as Cloud Data or Surface Temperature.
- **Access by Satellite Instrumentation**: In this section the data is classified by the instrumentation of the satellite that took the measures, such as ATMS, VIIRS, OMPS.
- **Under Development Datasets**: These datasets are not fully developed, so they might be incomplete and their access could be thus partial.
- **Satellites' Images**: This section holds a record of images in JPEG or PNG format, representing the areas visible by the satellite. Since we are interested in processable data, and no information is provided about the content held by each pixel of the picture, this section is not of our interest. Nonetheless, it could be useful to see certain events, impossible to be detected otherwise.

Dataset	Brief Description	Spatial Resolution	Time Resolution
ISCPP Cloud Data	Global cloud information	From 32 to 280 km	Since 1983, three-hourly to monthly
SSMI-SSMIS	Precipitations, snow and ice	1 to 2.5 degrees	Since 1987, monthly
GOES Aerosol	Visible spectrum aerosol	4 km	Since 06/2006, half-hourly
ASDTA	Smoke detection and prediction	4 km	Since 05/2005, weekly

Table 2.1: NOAA's Atmospheric Data Datasets Properties

Dataset	Brief Description	Spatial Resolution	Time Resolution
OISST	Ocean surface temperature	0.25 degrees	Since 1981, daily
Blended Sea Winds	Wind direction	0.25 degrees	1995–2005, every 6 hours

Table 2.2: NOAA's Oceanic Data Datasets Properties

2.2.4 Spatial and Time Resolution

When studying the spatial and time resolution this agency provides us, there are two main data types in which the NOAA is divided on, with different properties and focus. These are:

- *Atmospheric Data*, which focuses on studying the different components and phenomena taking place on the atmosphere, and
- *Oceanic Data*, which focuses on studying the phenomena taking place on the ocean.

On one hand, when it comes to the *atmospheric data*, we can see a brief summary of some of their most important properties in Table 2.1.

As we can see in the table, the resolution provided by the NOAA services is sometimes geographically very low and varying, so we haven't been able to extract useful information from their datasets. In addition, these satellites take measures over America, while our interest is located over Europe and, more precisely, the Iberian Peninsula.

On the other hand, we will present some important features of the *Oceanic Data*. Despite of the great amount of available data on this matter provided by satellites, there is also a wide range of non-satellite sources.

In this kind of data, there is a greater spatial and temporal resolution than in atmospheric data, reaching 0.25 degrees resolution and a daily frequency, and even better for some cases, as we can see in Table 2.2.

Although the name of the service may be confusing, these data doesn't focus exclusively on the ocean but in the coastline too, due to the big impact the ocean and coasts have in the US economy and climate issues.

2.2.5 Access, Data Refresh Rate and Format Used

The access is allowed through several services, such as FTP. Apart from the FTP server, they have also designed a new protocol specifically oriented to handle the order placing at their

site, which is also cost-free. Due to the large amount of products, departments and different kinds of data involved in the NOAA, the refresh rate may vary from a dataset to another, and we usually find rates varying from daily to monthly for climatology data. Another important factor to study is the format used in the distribution services, since there is a large number of them in the meteorological field. Among all these formats, the NetCDF is the most widely used. Due to its popularity and wide support, this is the one we will mainly use in this work. Therefore, we have dedicated an entire section (Section 3.3.1) to describe it in more detail.

2.3 European Organisation for the Explotation of Meteorological Satellites

2.3.1 Source Description

The European Organisation for the Explotation of Meteorological Satellites (EUMETSAT), as stated by its name, is an organization for the study and exploitation of meteorological satellites, focusing on monitoring the meteorological conditions and *climatology*, which is defined as the study of the *climate*, that is, the weather conditions averaged over a period of time.

It's composed by thirty states and one cooperative state, all of them located in Europe, although there is a high level of cooperation with another meteorological and space Agencies, such as NASA. The difference in being a cooperative state or a member state is the representation in the supreme decision-making organisms.

As a global agency dedicated to the meteorology, among its duties and main tasks there are some specially important ones, such as the development and construction of new satellites, the data processing of the current satellites' output, the monitoring of weather and meteorological conditions, the study of the atmosphere's components and the data distribution to its users.

This has been the only data source capable of providing us the required data with enough frequency and spatial resolution, satisfying every requirement that we needed to consider a final source, as we will see in the discussion. One of the most restrictive requirements is the need for data with an hourly or lesser (15 minutes) frequency. Moreover, another important property is to have enough spatial resolution and coverage to be able to study Europe and the Iberian Peninsula properly.

Its satellite network will be composed by the following six programs, since some of them are yet to be launched:

- Meteosat First Generation (MFG), 1977 – 2016, 7 geostationary satellites.
- Meteosat Second Generation (MSG), 2004 – 2025, 4 geostationary satellites.
- Meteosat Third Generation (MTG), 2019 – 2039, 6 geostationary satellites.
- Metop, 2007 – 2024, 3 polar satellites.
- EUMETSAT Polar System Second Generation (EPS-SG), 2 polar satellites.
- Jason, 2009 – 2036, 3 marine satellites.

As we can see in the release dates, and due to our need for a geostationary source, the most interesting networks are the MSG and MTG. But since the MTG is to be launched in 2019, we are going to study the MSG, which is composed by four satellites: MSG-8, MSG-9,

MSG-10 currently actives and MSG-11, which is not active, located over Europe and the Iberian Peninsula. They take measures and pictures every 15 minutes over a grid bounded between ±60 latitudes and ±60 longitudes (comprehending Europe and Africa), which gives us a great coverage and enough number of samples.

These satellites are equipped with the Spinning Enhanced Visible and Infrared Imager, SEVIRI, technology, so its measures are directly focused on studying the solar light spectrum, and all the information extracted from it, such as the incoming radiation.

2.3.2 Content

Since the EUMETSAT is an enormous agency, an important amount of the work we are interested in is developed by some other agencies dependent from EUMETSAT itself, such as the Satellite Application Facility, SAF, network, which is a set of decentralized centers for data processing from EUMETSAT sources.

Nonetheless, these agencies are often focused on *climatology*, a field not much of our interest for this current work, and therefore there is a lot of data that we need to take directly from the EUMETSAT distribution centers.

As we could expect, there is a different degree of processing between the EUMETSAT's data and the SAF network data, which are much more elaborated and organized, as well as more optimally converted to the distribution format, which is usually NetCDF. Due to this, we have developed some programs focused on reading and understanding the raw data downloaded from EUMETSAT.

The data and products we are more interested in are:

- Cloud Fractional Cover.
- Raw SEVIRI 1.5 Image.
  - Effective Radiation (total of eleven channels)
    - Seven Regular Bands (channels whose measures only represent effective radiation).
    - Four Absorption Bands (channels whose effective radiation's physical meaning is not just effective radiation, i.e., water vapour absorption).
  - Brightness Temperature.
  - % Reflection.
- Total Ozone.

Nonetheless, these are just some of the products developed by EUMETSAT, whose content goes from tropospheric humidity to clear-sky radiation and more. On the other hand, as it is raw data, there hasn't been applied any preprocessing algorithms, except those to convert the data to the final distribution format.

2.3.3 Access Details

Since this is the main source we have used in this work, we will explain its organization and more specific properties in Chapter 3. In Section A.4.3 we describe in detail the programs and previous work developed in order to process the data and adjust them to our further processing.

We have encountered extremely fine resolutions, reaching up to 0.03 degrees, and even better, which can allow us to perform our experiments with a good level of detail.

Moreover, the Data Centre, which is the web portal developed by EUMETSAT to handle and manage data orders, allows us to select the exact grid we want to download for a certain product. This is very convenient because it allows us to download just the data we want, avoiding a lot of extra megabytes.

There are different methods to deliver the order, such as through HTTP, DVD media, or FTP server, although the default way is through HTTP. In case of a large amount of data, the only way to deliver it is through DVD, shipped by EUMETSAT itself to any provided postal address without any additional cost.

Nonetheless, when available, we have always preferred the HTTP delivery. Other methods, such as DVD shipping, quite often take up to 15 days to deliver, while by HTTP the order is usually completed within few hours or days. For this first exploratory work, as we will later explain in more detail, our downloading and ordering process has been manual. There is also another distribution service based on reception antennas, which gives real time data and delivers it as soon as the satellite takes the measures, on the contrary of what happens with the other services, which provides data older than 24 hours. Due to its license cost, and the need to put a parabolic antenna for the reception, we are not using this service.

The refresh rate is highly determined by the frequency with which the satellite takes measures of the target. Since there is so few preprocessing in EUMETSAT data, the overhead is not noticeable, on the contrary of what happens in other data sources, such as the SAF centers we are going to describe later on.

The expected rate for the products we are interested in is really high, with measures every 15 minutes, converted to NetCDF format. Due to this high frequency, we will have a lot of data to perform analysis and experiments in future work, allowing us to get a better global picture of the data we are studying.

2.3.4 Format

The range of possibilities EUMETSAT offers is so large that we can, virtually, download every product in any format. Nonetheless, among all these formats, the most widely used are NetCDF, GRIB, and HDF5, and because of that, they are also the formats with the best programming language support in terms of libraries' availability.

As a general rule, the formats we are rather comfortable with are NetCDF and GRIB, because of their good libraries' support in Python, which is the language with we have developed our platform, and because they are considered standards in meteorology.

Finally, since this source meets our requirements, as stated at the beginning of this section, we are working with EUMETSAT as our only data source from now on. Nonetheless, we are presenting below a set of data sources that depend on EUMETSAT and may be interesting as future references in further work.

2.4 Eumetsat Satellite Facility on Support to Operational Hydrology and Water Management

The Eumetsat Satellite Facility on Support to Operational Hydrology and Water Management, H-SAF, was created by the EUMETSAT in July 3rd 2005, and integrated into the EUMETSAT's SAF (Satellite Application Facility) network.

This SAF, as we can deduce from its name, focuses its work in Hydrology and Water Management. Its main products are, therefore, precipitation rates, snow, soil moisture and some other related phenomena. As a SAF, its product quality and elaboration is greater than EUMETSAT's.

A walk through of this data source is presented below, studying their main products and properties, as well as a brief discussion about its suitability or interest in future work.

2.4.1 Source Description

The SAF network, as mentioned before, is a set of processing data centers, each one focused on some kind of product extracted and published by the EUMETSAT from its satellites. Their focus on a specific kind of products allows them to release more detailed and organized data, improving the overall EUMETSAT's quality.

The H-SAF has, among its duties and goals, the following ones:

- To provide existing and future satellite products with enough spatial and time resolution to be able to satisfy the required needs of an operational service based on hydrological products. The operational products are the following:
  - Precipitations.
  - Detection, cover, snow water equivalent and other snow parameters.
- To perform independent validation of the usefulness of the products against a range of natural disasters, including floods, landslides or avalanches.

2.4.2 Content

The H-SAF generates a set of high quality datasets processed from the raw satellite data coming from a variety of sources, mainly EUMETSAT. To properly understand the measures and their meaning, it's important to keep in mind the instrumentation they capture data from:

- LEO/MW refers to a compact infrared detector and radiometry used to retrieve some metrics involving precipitations and, more broadly, water related properties,
- MW refers to the same radiometry indicated above, and it is a dedicated instrumentation to measure water related properties,
- GEO/IR refers to Geostationary and Infrared instrumentation on board of the satellites,
- VIS/IR refers to Visible and Infrared instrumentation on board of the satellites

Keeping in mind the instrumentation just mentioned, the different products are categorized as follows:

- Precipitations
  - PR OBS 1 - H01: Precipitation rate measured at ground by MW conical scanners.
  - PR OBS 2 - H02: Precipitation rate measured at ground by MW cross-track scanners.
  - PR OBS 3 - H03: Precipitation rate measured at ground by GEO/IR supported by LEO/MW.

  - PR OBS 4 - H04: Precipitation rate measured at ground by LEO/MW supported by GEO/IR.
  - PR OBS 5 - H05: Accumulated precipitation measured at ground by blended MW and IR.
  - PR OBS 6A - H15A: Blended SEVIRI Convection area / LEO MW Convective Precipitation.
  - PR ASS 1 - H06: Instantaneous and accumulated precipitation measured at ground computed by a NWP model.

- Soil Moisture
  - SM OBS 1 - H07: Large scale surface soil moisture measured by radar scatterometer.
  - SM OBS 2 - H08: Small scale surface soil moisture measured by radar scatterometer.
  - SM DAS 2 - H14: Profile Index in the roots region measured by scatterometer data assimilation. Makes and stores a profile information index about the structure in the regions about 2 m below surface in different layers, allowing us to understand the soil moisture assimilation by the given surface.
  - SM OBS 4 - H25: Metop ASCAT Soil Moisture Time Series.

- Snow
  - SN OBS 1 - H10: Snow detection (snow mask) measured by VIS/IR radiometry.
  - SN OBS 2 - H11: Snow status (dry, wet) measured by MW radiometry.
  - SN OBS 3 - H12: Effective snow cover measured by VIS/IR radiometry.
  - SN OBS 4 - H13: Snow water equivalent measured by MW radiometry.

2.4.3 Data Organization

The data structure is highly determined by the availability and cover of the data itself, in more detail, whether it is pre-operational or operational.

In case it is pre-operational, the data is permanently improving and under development, before it is declared as an operational dataset. On the contrary, the operational products are those stable and currently running.

The inner data structure is not different from what we explained previously for other data sources, since all of them use quite similar formats, the standards for each field. Unfortunately, due to its hydrological focus, this source didn't turn to be what we were looking for, since we are currently focusing on radiation products, at least at this early stage, so we won't be using it in this work. Nonetheless, it allowed us to get a better approach to other sources and other sort of products that might interest us in future work.

In Table 2.3 we show the spatial and time resolution for these datasets.

2.4.4 Access, Refresh Rate and Format

The access is generally made through its order web portal, although it can also be ordered through the EUMETSAT data center. As mentioned in EUMETSAT's section, the access is cost-free and the orders doesn't take too long to complete.

The refresh rate is not the same for every dataset, as we can see in Table 2.3. Nonetheless, this variation is even more drastic for the under-development datasets. Moreover, the refresh

Dataset	**Spatial Resolution**	**Time Resolution**
PR OBS 1 - H01	30km	Up to 6 passes per day
PR OBS 2 - H02	Varies between $16 \times 16km^2$ y $26 \times 52km^2$	Up to 6 passes per day
PR OBS 3 - H03	8km over Europe	Every 15 min
PR OBS 4 - H04	8 km over Europe	Every 3 hours
PR OBS 5 - H05	8 km sampling, 30km effective	Every 3 hours
PR OBS 6A - H15A	8 km over Europe	Every 15 minutes, along every SEVIRI image
PR ASS 1 - H06	7 km	12 hours (phase 1) and 6 hours (phase 2) [1]
SM OBS 1 - H07	25 km	Every 2 hours
SM OBS 2 - H08	25 km	Every 2 hours
SM DAS 2 - H14	25 km	Daily at 00:00 UTC
SM OBS 4 - H25	25 km	Every 1-2 days, depending on latitude
SN OBS 1 - H10	1-5 km, depending on equipment	Daily
SN OBS 2 - H11	10-30 km (0.25 degrees)	Daily
SN OBS 3 - H12	0.25 degrees	Daily
SN OBS 4 - H13	0.25 degrees	Daily or weekly

Table 2.3: H-SAF Datasets' Properties

rate is highly related to the time and the phase in which the satellite is active, since it varies according to the capacities of the satellite and the frequency it takes measures.

The measures are taken over grid points, and the format is usually GRIB or BUFR (see Chapter 3 for more details about these formats), standards in this field, as we saw in previous data sources. Nonetheless, along the formatted data there is always a JPEG image that helps to quickly overview the inner information.

2.5 Climate Monitoring - Satellite Application Facility

The CM-SAF, standing for *Climate Monitoring - Satellite Application Facility*, is a EUMETSAT dependent processing center created at EUMETSAT 32–nd council meeting.

As its name states, the CM-SAF focuses on climate monitoring tasks, that is, among the topics it studies there are several cloud parameters, surface albedo, radiation fluxes at the top of the atmosphere and at the surface as well as atmospheric temperature and humidity products, forming a sound basis for climate monitoring of the atmosphere.

These products are categorized according to their topic or focus, and are obtained in nearly real time. Their datasets are based on inter-calibrated radiances, which will be a really important parameter, as we will see in more detail in Chapter 3. The CM-SAF gets its data from EUMETSAT satellites, mainly Meteosat satellites, specifically the Meteosat Second Generation.

Its main goal is to build satellite-derived geodesic datasets whose target is climate monitorization. As we earlier said, SAF's products will usually have better quality and organization, which makes them an interesting source.

Given the radiation-related nature of its products, this is one of the most interesting data

sources for us. Nonetheless, its goal is the climate study, not the meteorology; that means that they look for historical data to observe and study phenomena such as climate change. Due to this, the interest awaken by the CM-SAF couldn't lead us to a real use of the source. Nonetheless, some of their algorithms may have interesting applications in further work.

2.5.1 Content

As we mentioned above, CM-SAF's goal is to build climate datasets, so these are:

- Cloud Products.
- Water Vapour and Temperature.
- Surface Radiation Products.
- Top of Atmosphere Radiation.
- Hamburg Ocean Atmosphere Parameters and Fluxes from Satellite Data, known as HOAPS, which is an external climatology-focused department.

In addition to the datasets themselves, the CM-SAF also allows us to access a variety of documentation about the product we are interested in, so we can learn more about the methods and algorithms performed to build the datasets from raw data downloaded directly from other sources, such as EUMETSAT. In this way, the users are able to verify the process and quality of the products.

At this point, some of the most interesting datasets, related to the topic of this work that we have found are, doubtless, those of the radiation kind. CM-SAF offers several products on this category, emphasizing the Direct Normal Radiation and Surface Incoming Radiation, both of them related by physical laws.

These datasets intend to measure *irradiance*, that is, the radiant flux *intensity* received by a surface in Wm/m^2 units, and which we will refer to later in this work. Nonetheless, its hourly data is mostly historical, going until 2013. Our need for current and future data didn't let us to use this source.

Despite of not being the final datasets, this was the first real contact with the data we were looking for, and studying these products gave us a better understanding and knowledge about the field we are getting into.

In its data processing to produce these radiation data, CM-SAF uses other datasets such as Cloud Cover Factor, Ozone Accumulation or Aerosol Concentration in the Atmosphere, which are provided as raw data by EUMETSAT and other agencies such as ECMWF (European Centre for Medium-Range Weather Forecasts). Thus, the final radiation data are the result of the application of several algorithms to these raw data. These algorithms perform a combination of apparently unrelated products such as aerosol, ozone properties of the atmosphere, incoming radiation and clear sky characteristics to produce *irradiance* as output.

A particularly interesting algorithm (among others) is known as gnu-MAGIC [2], which internally uses the *Heliosat* [3, 4] method to compute the *Effective Cloud Albedo* (CAL). This algorithms computes, first, the clear-sky radiation, to later apply some corrections and so adjusting the result to the real value of the irradiance at a given point.

Even though the algorithm was interesting, for it revealed some of the physics behind this transformation, it couldn't be added to our final results because we couldn't get access to the same data they had in order to produce the final irradiance. Nonetheless, we keep this as an interesting reference to study in more detail in future work.

2.5.2 Data Organization

The inner dataset organization is related to the format it is assembled in. They are, generally, binary and hierarchical formats, accessed by a *groups* and *subgroups* structure. Nonetheless, we'll explain more about these formats in Chapter 3, though we have already talked about some of them, such as NetCDF and GRIB.

In order to organize this variety of products, the CM-SAF divided them into dataset categories, and within them, by whether they are operational or historical. We are mainly interested in operational data, in order to produce updated measures.

2.5.3 Spatial and Time Resolution

All the products developed and built by CM-SAF have their source in the satellite networks of EUMETSAT, particularly the Meteosat network. The most relevant Meteosat generation for us is the second generation, known as MSG (see Subsection 2.3.1), due to its location over Europe. As we explained in previous sections, given their geostationary orbit they are able to take measures all day long every 15 minutes, allowing us to study with great detail and coverage the meteorological phenomena happening all around Europe and the Iberian Peninsula. As we mentioned earlier, the grid measured by these satellites covers Europe and Africa (± 60 latitudes and ± 60 longitudes).

The spatial and time resolution is the same we get in EUMETSAT, up to 0.03 degrees with frequency varying between 15 minutes or one hour for older satellites' instrumentation, and even one day or more for the second generation of the Meteosat satellites in current data.

As stated at the beginning of this chapter, this low time resolution on current and operational data has been the main reason that stopped us from using this data source as the final one, despite of having hourly data for historical products. Nonetheless, the products and data processing that CM-SAF performs could be an interesting reference for future work.

2.5.4 Access, Refresh Rate and Format

The access to the data built by the CM-SAF is through their online order site, which is a web platform where we can select the product to download, the subgrid we actually want and the desired resolution for the data, which is bounded by the minimum values established by the instrumentation.

These orders are completed within the next hours for small orders (around 50 GB). Nonetheless, the time to complete an order is greatly increased as the data amount grows, reaching up to a week or ten days for an order of 300 GB, including a possible format conversion.

The refresh rate is related to the product category, depending on whether it is operational or historical. For operational data we find mostly daily or weekly data, while for historical data we have up to hourly frequency. Nonetheless, for historical data there is not refresh nor continuity because the satellites responsible of these datasets belong to the Meteosat First Generation, which is deprecated and no longer operative (Section 2.3.1).

When it comes to the file format, we have a whole set of possibilities, going from HDF5 to the more currently used NetCDF, converted directly from the latter. We need to take into account that if we want a format conversion from the native EUMETSAT format, the order may take more time to complete. The different formats are explained in Section 3.3.

2.6 Conclusions

All throughout this chapter we have reviewed a series of possible data sources for this work. As we have seen along the several descriptions presented about different data sources, the only one that is actually suitable for our work is EUMETSAT; summarizing, the main reasons for this choice are the following:

- It offers us enough product categories related to our research topic, focused ultimately on photovoltaic energy production.
- Its high spatial and time resolutions will allow us to perform a detailed research on the meteorological conditions taking place in the Iberian Peninsula.
- Contrary to other sources, and due to its raw data and low processing, the availability is almost instantaneous, which allows us to perform any transformation we may want and whatever algorithms we think necessary without any substantial delay.
- The formats used by EUMETSAT are widely supported by Python's libraries, what makes really easy to read, analyze and understand the data contained, without having to spend much time programming those libraries ourselves.

In short, the other sources, although interesting and important for us to get a better understanding of the field and the problems we are studying, don't meet the minimum requirements we need to actually consider a definitive data source. On one hand, the American data sources, the NOAA and its derivates, are too wide and more climate-focused than meteorology-focused. This means that their efforts are more directed to better understand the climate change and other climate phenomena. On the other hand, the centers from the SAF network, closer to our needs, are also focused on historical data, and don't provide enough resolution or frequency for real time data, which would keep us away from maintaining our possible results updated. Moreover, its large data processing is another added delay to the release of the data, which results in daily or monthly data. Nonetheless, we need at least hourly or, if possible, even more frequent data access to get enough samples to produce our forecasts in near real time.

As a result, the only data source we are going to use is the EUMETSAT data center, where we have extracted a set of products to support this TFG and our future research.

After all we have presented in this chapter, in the next one we will focus our discussion exclusively on the EUMETSAT source, going through its main features and describing in more detail the data and variables we are using, along with a detailed description of the formats we will use.

3

Analysis of EUMETSAT Data

3.1 Access Details

To access the available data, EUMETSAT has developed a web portal where users can handle their subscriptions to any product or services the agency offers, such as disseminated data services. The registration is free and allows the user to access a wide variety of services, not all of them directly related to the data access and download, but to other services like newsletters and so on.

There are several ways to access EUMETSAT, either through the portal data center we explained in Chapter 2, which is a javaweb application where we can make our data petitions, or by adding a FTP server where we want EUMETSAT to upload the data we order whenever it is available from the Satellite. This last method is different but highly related to EUMETCAST, a private dissemination service that delivers the data to its users without the need of an order, whenever the subscribed data is available. The EUMETCAST service has a cost of €100 on licenses plus the added cost of the parabolic antenna required to fetch the products.

For this work we have accessed the data by the online Data Center Application, for it requires just a basic subscription, which is cost-free.

3.2 File and Data Structure

Although the most interesting data source for us is the MSG (Meteosat Second Generation) satellites, EUMETSAT works and manages an immense data network, divided and organized into several categories, which makes it a lot easier to search in and to order new data.

These categories are split according to the satellite's orbit type, that is, we can choose between two basic categories: whether the orbit is geostationary or not. Since we just need data over a specific region and not worldwide, we are mainly interested in geostationary data sources. Nonetheless, there is also a lot of products and datasets derived from non-geostationary satellites, which can be of some interest in the future.

Looking into the selected category, we can see a great variety of products, mainly in raw format, that is, with almost no preprocessing. For instance, some of these products are Ozone

Concentration on the Atmosphere, Tropospheric Humidity, Cloud Cover Factor, Clear-Sky Radiation, and more. These data are available in a variety of different formats, explained in next subsection. Some of these products even allow us to select the subgrid we want to download.

3.3 Studied Formats

All along the way to build and design our storage system for data analysis we have had to deal with a variety of formats and data. To properly work with these data and choose the best suitable formats in terms of storage and accessibility, we had to understand them. Additionally, we are looking for formats with good Python libraries' support and ease of access.

In the following subsections we will walk through all of these formats, describing their main properties, as well as their advantages over others. Finally, we will be working with just two of them, NetCDF and GRIB, which we consider the best suited for our work, apart from being the most widely used in the field.

3.3.1 NetCDF4

The term NetCDF appeared at the Unitdata labs as a set of libraries and a self-describing, machine-independent data format. This format focuses on the access, creation and release of scientific data organized into a matrix structure.

This format has grown all the way to a standard format in the meteorological field, due to, mainly, its ease of reading and its good libraries' support on several of the most widely used programming languages, like Python, the language we have used in our inner software.

Its structure is essentially hierarchical, built around the concept of *group* and *subgroup* as the main container units, and the *variable* and *dimension* as the main data units. Every variable contained in a NetCDF file is represented as a matrix whose axes are determined by the dimensions it depends on. These dimensions are usually, at least in the meteorological field, the latitude and longitude of the measured data at a certain point. These are, on the other hand, a plain data type that only represents the unit and the size of the data contained in the matrix. Their only goal is to define a user-friendly wrapper for the data, so its context and units are not unnamed and therefore we can easily understand what kind of data we are looking at.

The only negative side this format has is that it is not a native format and its structure is not standardized, which brings different possible representations for the same information, depending on the format it is converted from and the process performed to convert it. Thus, the structure of each file depends highly on the conversion process.

3.3.2 HDF5

HDF5 is a data model, library and file format for data storage and management. Its design focuses on getting the best flexibility, on working with big data and on getting an input/output efficient management of great amounts of data and complex data structures. It is developed and maintained by the HDF Group (we can see more information in their website).

The HDF5 is a widely used format for meteorological data storage, as well as for general information handling. Its structure is built around the *Group* and the *Dataset* concepts. A group is a natural hierarchical subdivision of the dataset with common logic properties among the data it contains. On the other hand, a dataset is the final unit that actually stores the data itself, and, just as in the NetCDF format, the ultimate data structure is a matrix.

Semantically, the information is divided upon DATA and METADATA. In the METADATA section we can find some metaparameters, such as inter-calibration parameters, offset settings, indications of what kind of processing is the data prepared to perform, and so on. It also includes information about the instrumentation the data was captured with, as well as of the satellite itself.

Finally, under the DATA classification we find information related to our specific domain. For the case of radiation products we find the channels data. These channels represent the data captured from each one of the light's spectrum bands (wavelength frequencies) by the satellite instrumentation. For the data we have downloaded, these bands go from the 0.6 µm through 13.4 µm, making a total of eleven channels and an added one for the *High Resolution Visible* channel.

3.3.3 GRIB

The GRIB format, whose acronym stands for GRIdded Binary, and is defined as a Regularly-distributed Information in Binary Format, was developed by the ECMWF, the European Center for Medium Term Weather Forecast. This format is considered a standard by the World Meteorological Organization's Commission for Basic Systems. The second edition of this format, namely GRIB2, is being widely used in an increasing amount of EUMETSAT products, given its growing popularity and support.

From its name itself we can see the nature of its inner structure, which, essentially, represents a grid containing a set of points. Each point is a pair `latitude, longitude`, being thus a very intuitive way to access the data.

Nonetheless, we can sometimes find that the grid is a rotated grid, which means that the points and its relations are stored as a curved grid, so the access may get a lot harder, having to un-rotate the grid, which may not be obvious. To achieve this goal, there are some tools and commands such as *cdo*, and its *remapbil* option. We have also developed a program to handle this kind of references, which is described in Appendix A.

Due to the `pygrib` Python's library, which is a wrapper for GRIB API access and methods, decoding and reading this kind of files is quite simple and allows us to work directly on the data, without delays because of format issues. Nonetheless, to being able to use this library, some other components are needed, such as the GRIB_API C's library itself, developed by the ECMWF.

3.3.4 BUFR

Another well known format is BUFR, standing for Binary Universal Form for the Representation of meteorological data. Its last edition, the BUFR Edition 3, is one of the currently used formats for operational products at well known systems such as the ECMWF or EUMETSAT itself.

Its inner structure is organized in sections, having a sort of general template for every file, which means that every BUFR file will have essentially the same sections and structure. This template provides five sections, each one of them with a specific purpose, as we shall see next.

In detail, Section 1 is oriented to give metainformation about the file itself, Section 2 contains a summary of the variables contained in each *sector* (which is a subsection of the data segment in the file) and its flags (several options related to the specific data), and finally Sections 3, 4 and 5 contain the dataset itself, named `bufr messages`.

For Python, there is not a good library to wrap the low level methods and thus make easier to read and decode BUFR files. There is an incomplete approach to a library though it is not fully developed. This library is named `pybufr-ecmwf`, developed by a Github user self-named jdkloe that includes a low-level version as well as an intermediate and high-level versions of the library. The lower the level, the greater the control it allows us to have over the file and the process. Fortunately, along with the library itself, *jdkloe* has developed some generic decoders that could be useful in other scenarios. Nonetheless, we haven't finally used this format in the data we will work with due to the hard access, API, and general file structure, which actually made us unable to adapt or improve the *default* decoders to our needs.

3.3.5 Native

This format is the native one used in EUMETSAT products. Unfortunately, we haven't found enough documentation about the format to be able to write a simple script to read and decode it directly. Since apparently no Python libraries have been written to read this format and most products have alternatives for it, we have avoided using it and always preferred using NetCDF or GRIB whenever possible.

3.4 Variable Summary

For our photovoltaic energy purposes, and establishing our understanding of the field, we have analyzed some radiation-related data from EUMETSAT data center. We have categorized the data in several variables, which are contained in the following products:

- *High Resolution SEVIRI Level 1.5 Images* and
- *Cloud Mask from Meteosat Satellites.*

Each product is organized in a different way; the *High Resolution SEVIRI Level 1.5 Images* is split into channels, counting eleven regular channels, each one covering a different spectral band (whose units are wavelengths in micrometers), going from 0.6 µm to 13.4 µm. There is another version of this product including an extra High Resolution channel covering the visible band with greater detail, which we have not downloaded because the regular resolution of $3km \times 3km$ is enough for this exploratory work. On the other hand, the *Cloud Mask* just holds one variable, representing the cloud thickness at a given point.

From the different channels and data included in the products, we have made a categorization in domain variables, which we will describe next.

- Cloud Mask form Meteosat Satellites.
  - Cloud Mask.
- High Resolution SEVIRI Level 1.5 Images.
  - Effective Radiance, on eleven channels.
    - Water Vapour Absorption, on Channel 5 and Channel 6, covering the spectrum at bands 6.2 µm and 7.3 µm.
    - Ozone Absorption, on Channel 8, located at band 9.7 µm.
    - CO2 Absorption, on Channel 11, located at band 13.4 µm.

    - On the rest of channels the Effective Radiance doesn't have any special meaning.
  - Reflectance for visible light channels, on Channel 1, Channel 2 and Channel 3, capturing bands 0.6 µm, 0.8 µm and 1.6 µm respectively.
  - Brightness Temperature, on the other eight infrared channels.

While the CloudMask variable is directly the measure contained in the downloaded data, we have to perform some transformations to extract the other variables from the channels included in the High Rate SEVIRI product. To get a better understanding of the data we are going to work with, we are presenting in the next sections a brief description of each variable.

At first sight, every variable extracted from the SEVIRI data is given by a formula in terms of the direct information contained in the measurements, so, as long as all the variables are related, they also incorporate different factors, which could make them more interesting for our modeling purposes in future work.

We now go over the variables extracted from the SEVIRI data starting with the CloudMask variable, for it is the simplest one.

3.4.1 Cloud Mask

As we just explained, the Cloud Mask variable is the easiest to get, as it is directly contained in the measures from the image downloaded (for other variables we will have pixel–like values, referred to as *pixel counts*, which are essentially the raw measures captured by the Satellite's instrumentation, and are proportional to the radiant energy received on each pixel).

This variable represents the percentage of cloud mask at certain time at a given point, so it is a floating point number within the range $[0, 5]$, where 0 means clear-sky and 5 totally covered sky.

3.4.2 Effective Radiance

The effective radiation measures the reflected incoming radiation flux (energy) by surface unit and wavelength. This is so because the satellite can only measure the radiance reflected by the Earth's surface. The brief pseudocode that explains its transformation from the *pixel counts* in the measures is:

```
radiances = (pixel_count + cal_offset) * cal_factor
```

where *cal_ offset* and *cal_ factor* are channel-related metadata, which are defined as parameters in the NetCDF file. These parameters are calculated by EUMETSAT and described in their inter-calibration documentation [5].

3.4.3 Reflectance

In case we are studying a visible channel, we could compute the reflectance percentage for this channel. This parameter is the fraction of reflected radiance with respect to the solar irradiance, which is the maximum radiant flux that can reach the Earth's top of atmosphere. This factor also includes some corrections due to the sun-earth distance and the cosine of the solar zenith angle. The mathematical formula that gives the reflectance is shown in [6]:

$$r_{\lambda_i} = \frac{\pi R_{\lambda_i} d^2(t)}{I_{\lambda_i} cos(\theta(t, x))}$$

where R_{λ_i} refers to the *radiances* for the channel λ_i, $d^2(t)$ is the sun-earth distance at time t, I_{λ_i} is the band solar irradiance for the channel λ_i and $\theta(t, x)$ is the solar zenith at time t and location x.

3.4.4 Brightness Temperature

Another interesting variable may be the brightness temperature, which, according to the definition in Wikipedia, measures the temperature a black body must be in thermal equilibrium to duplicate the gray body radiation intensity, observed at a certain wavelength. In essence, it is a measure of the radiance of infrared radiation traveling upward from the top of the Earth's atmosphere. These measures are considered a fundamental climate factor from which we can derive ocean measurements of wind speed, water vapour, cloud liquid water, rain rate and sea surface. Nonetheless, those derivations are not considered in this work. This variable may only be measured on the infrared channels, i.e., those whose wavelength is over 3.9 µm.

The equation that gives its value is:

$$\frac{\dfrac{100C_2w}{\log\left(\dfrac{w^3C_1\text{1E6}}{r\text{1E-5}+1}\right)}-\beta}{\alpha}$$

where α and β are channel-dependent parameters, w is the wavelength of the channel, and C_1 and C_2 are constants. As the other variables, all the parameters are defined by EUMETSAT's inter-calibration documentation at [7]. Again, r, standing for *radiances*, is the variable we calculated earlier in this section.

3.4.5 Water Vapour Absorption and other Absorption Bands

Finally, the last set of variables we will be studying is that of the absorption bands. One of these bands is the Water Vapour absorption, which is just the *pixel count* (we recall that the *pixel count* is the raw measure the satellite's instrumentation directly captures) measured in those channels whose band is located at 6.2 µm and 7.3 µm.

Along with this channel there are other meaningful absorption bands on the eleven channels captured in the High Rate SEVIRI data. These channels are the Ozone and CO2 absorption bands, located at 9.7 µm and 13.4 µm respectively, which we will also study.

3.4.6 Brief Conclusions

Once we understand the nature of the variables we are studying and analyzing, we have ranked them in terms of ease of access and processing, being the ranking as follows:

1. **CloudMask**. It doesn't require any kind of computation apart from the extraction itself.
2. **Effective Radiance and Absorption Bands**. These variables are computed in the same way, the only difference is the meaning of the channels they both come from. While the effective radiance is the *pixel count* in every channel, the absorption bands variables are exclusively extracted from the channels mentioned in Subsection 3.4.5.
3. **Reflectance**. This one is also easy to handle because all the parameters are easily retrieved from the EUMETSAT documentation.

4. **Brightness Temperature**. As the most complex variable we have the brightness temperature. To compute this variable we need more factors than any of the other variables, although they are also given by the NetCDF file, and defined in [7].

Nonetheless, this ranking is useful when it comes to evaluate the data processing each variable needs, but not when it comes to evaluate the meaning and importance of each variable in our work. From this ranking, we understand that the SEVIRI images provide us with more meaningful data than the Cloud Mask on its own as a first hypothesis. In Chapter 4 we present a more detailed analysis and conclusions about the relations of these variables with the photovoltaic solar energy.

3.5 Storage System

Once the data to download have been selected, there appears the problem of how to download and to store such data. As we discussed throughout this chapter and Chapter 2, the format is an important matter to discuss here, for it is an essential factor when it comes to space and data volume.

The online EUMETSAT data orders are somehow restrictive, and we can only make up to 10 simultaneous orders under 300GB each. This is a constraint because our goal is to download three years of historical data of these two products: *High Rate SEVIRI 1.5 Images* and *Cloud Mask*. The format and frequency of the downloaded data is highly determinant on the final size. At this respect, we have made some calculations.

- Cloud Mask: one hourly file with size depending on format
  - NetCDF, 160MB per file, having 8,760 hourly files per year, would lead to 1.401TB.
  - GRIB, 1MB per file, having 8,760 hourly files per year, would lead to 8.76GB.
- High Rate SEVIRI Level 1.5 Image: one file with size depending on format
  - NetCDF, 3.90MB per file, having 8,760 hourly files per year, would lead to 34.25GB.
  - HDF5, 3.81MB per file, having 8,760 hourly files per year, would lead to 33.37GB.
  - Native, 2MB per file, having 8,760 hourly files per year, would lead to 17.52GB.

This would have several consequences in terms of storage and order placing. Since we want to download at least three years of data to have enough samples, we need to look for an equilibrium between size and accessibility. In the first case, the CloudMask data, we directly chose the GRIB files because of the great difference of weight. Nonetheless, in the case of the SEVIRI data, the most space saving option would be the Native format, which we don't know how to read. This leaves us the HDF5 or NetCDF formats as options. Since NetCDF is becoming the main format for EUMETSAT products, we finally chose it over HDF5 for the SEVIRI dataset.

These calculations are based on hourly data downloads, although EUMETSAT actually provides these products every 15 minutes. We chose the hourly data considering the following:

- If we decided to take the 15 minutes products, it would have caused us to place four times more orders, delaying the completion time.
- It would have also quadrupled the disk storage needs.

- The hourly data should be a suitable step to build our models, while keeping a good weight equilibrium.

Summarizing, our total amount of data storage is about 130GB for three years, which is the result of the following count:

- SEVIRI NetCDF, downloading 8,760 hourly files per year, which makes 34.25GB each, and a total of 102.75GB.
- CloudMask GRIB, downloading 8,740 hourly files per year, which makes 8.76GB each, and a total of 26.28GB.

We downloaded these data by manually placing 12 orders of 6 months of data each in the online EUMETSAT data center. The total time spent downloading and ordering data has been nearly a month, including all the process: from the order placing to the final download. The time that takes EUMETSAT to complete an order varies from product to product and it is proportional to the amount of files and size of the order. Another factor that affects this completion time is the selected format, because it may involve a conversion process from EUMETSAT's native format to the chosen one. Once the order is completed, EUMETSAT mails us the URL where we can find the requested data. To automatize the process, we tried to write a simple Python program that would place and download the orders periodically for us. Nonetheless, since the page is fully rendered in `JavaScript`, and the different forms are generated dynamically, we weren't able to complete the process. It wasn't a requirement for this work, but it may be important for the future, either this way or directly by satellite, as we mentioned in Section 3.1.

On the other hand, we also need to design a file structure that helps us to organize the products once downloaded. A first attempt to this structure would be:

- eumetsat
  - raw
    - seviri
    - cloudmask
  - processed
    - seviri
    - cloudmask

where the *raw* directory stores the data directly downloaded from EUMETSAT, and the *processed* directory stores the output of our data processing, which will be a plain format in a matrix-like structure. A complete description of this format is in Appendix A.

4

Data Interpretation and First Application

4.1 Introduction

After downloading the selected products and setting the base storage system, we are now focusing on interpreting the data and extracting some conclusions about what the products and their variables are actually representing. Since the number of channels and possible calibrations are too many to explore here, we are first focusing on the SEVIRI's *effective radiation*. Nonetheless, after this first interpretation we will perform a more detailed analysis of all the variables. In further work we may study some possible extra products, such as *Aerosols* and others that may be related to photovoltaic solar energy.

As we explained in Chapter 3, every channel in the SEVIRI data represents a different spectrum of the light, covering all the visible light spectrum and the first channels of the lowest infrared band. Nonetheless, each channel has its own properties and specific characteristics. For instance, the first two channels represent the visible light, while the rest of the channels are located in the infrared segment. These properties and each channel's specific meaning are the main focus of this section. To study it properly, we are first presenting some definitions and concepts, and, after that, we will discuss several figures of different channels' data. To finish this chapter, we will present a first data analysis focused on building a simple linear regression model as a first application on photovoltaic energy, along a brief discussion on the model's errors.

4.2 Channels Coverage

The SEVIRI data cover eleven channels, as we saw in Chapter 3, and each one of them captures data from a different spectrum band. The first two channels cover the visible light spectrum on the 0.6 µm and 0.8 µm bands and the third one is the first of the infrared segment, at 1.6 µm band. This third channel, since it is so close to the visible light spectrum, it is considered to be a *near* infrared channel. For the purposes of this work, we will consider this third channel as a *visible* one. This means that the Sun radiation is much stronger in these *visible* channels than in longer wavelengths, where the Earth radiation will be dominant.

The other eight channels will cover the wavelengths from 3.9 µm to 13.4 µm. These channels are characterized by the high influence of the Earth thermal radiation, which makes these

channels interesting for different factors. Examples of these channels are the *Carbon Dioxid*, the *Ozone* or the *Water Vapour* absorption bands, as we can see in Figure 4.1.

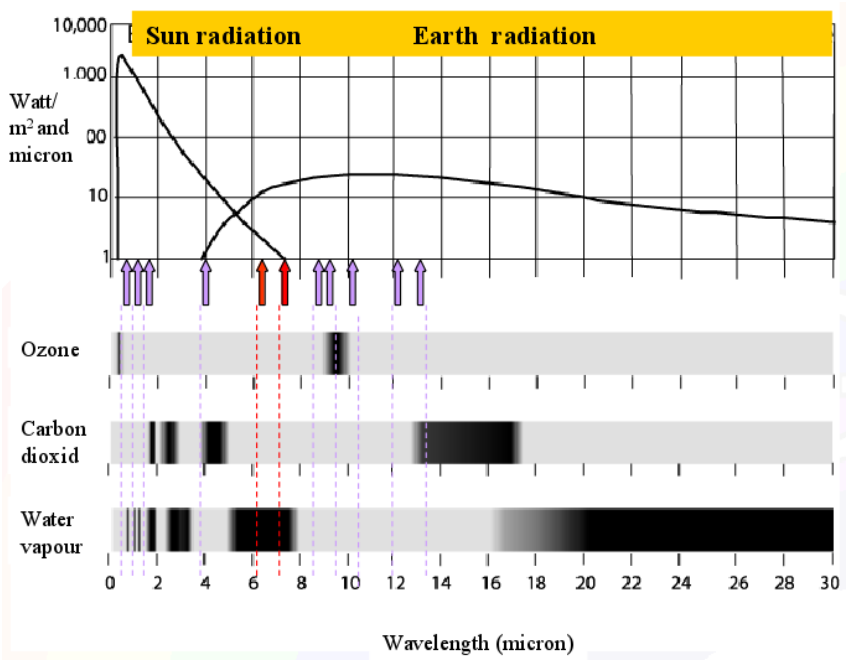

Figure 4.1: Sun Radiation and Earth Radiation in EUTMETSAT channels. Different channels are indicated with vertical arrows. Image extracted from EUMETSAT's training presentations

Due to the SEVIRI instrumentation and the properties of each spectrum band and the different surfaces the radiation reaches, we will consider the following two effects on the measures:

- On one hand, as the day goes (until the evening), we will notice a general increase of the *irradiance* (the measure at a given point), in the channels dominated by the Sun radiation.
- On the other hand, we also have to take into account the *reflection* and *absorption* of the light by the different elements that are in suspension in the atmosphere and by the different surfaces. As we shall see, the reflectance of the land is radically different from the reflectance of the sea, which is caused by the *albedo*, defined in Subsection 4.2.1.

This will also affect the images and the possible interpretations, easily confusing *brightness* in the image or higher values in the measures with *radiance intensity*, when it might represent a cloudy area, i.e., a surface with a high reflectance factor. We can see a graphic comparing the soil and leaf reflectance factors in the three main visible channels in the solar radiation spectrum in Figure 4.2.

The three darkest vertical bands we see in this figure correspond to the three visible channels, Channel 1, Channel 2 and Channel 3 respectively. This figure shows how much reflection we can expect in each channel, i.e., Channel 2 will have a significantly higher leaf reflectance, when compared to the other two. This reflectance property can be easily adapted to some applications related to recognizing clouds or snow, because of the high reflectance of these elements, which will appear as *bright* areas in our *visible channels* images.

4.2.1 Land and Sea Surface

The behavior of the different surfaces when incoming radiation lands is a really important factor, as we mentioned before. This behavior is strongly affected by the *albedo*, which is one of

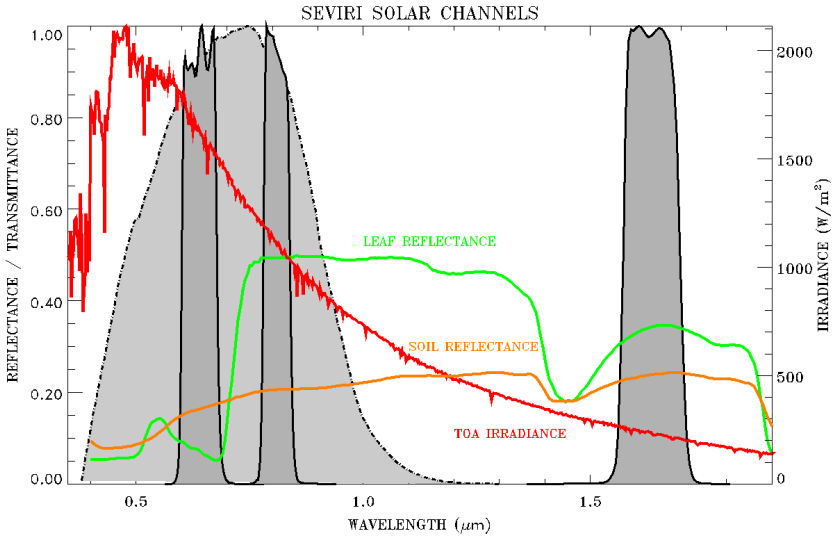

Figure 4.2: Top of Atmosphere radiation, Soil and Leaf reflectance in the three visible (VIS) channels. Image extracted from EUMETSAT's training presentations

the most important properties we are studying here, as we briefly mentioned earlier.

The **albedo** [8, 9] is the fraction of incoming light reflected by a given surface. The radiation that is not reflected is therefore absorbed energy, which causes melting of the ice, increase of the surface temperature, evaporates water, etc.

The main surfaces that will affect the measures are *land*, *sea* and *clouds*, and it will be important to remember this, as we will see the ocean as *dark* areas, which means that the sea doesn't reflect so much radiation as a desert or a cloud. The *albedo* is also dependent on the wavelength of the incoming radiation, thus different channels might show a completely different absorption or reflection.

4.3 Images

Keeping these concepts in mind, for they are essential to understand the images we are presenting here, we introduce a brief interpretation and discussion of the SEVIRI's effective radiance on the most interesting channels.

Along the images we are going to show the satellite's *greyshades* image of the Iberian Peninsula. This will allow us to better identify the interesting areas and the different meteorological phenomena, such as cloudy areas. At this point, understanding the different surface *albedos* will help us to identify effects such as clouds, snow and icing phases. In future work we will try to explore further these relations. A common misunderstanding might happen during dark hours, where the satellite doesn't capture any valid data and only sends instrumentation noise.

As we earlier mentioned, since we have so many channels and calibration results, we are going to focus on representing the most interesting ones, namely the visible (VIS) channels and some infrared (IR) absorption channels.

4.3.1 Visible Channels

In Figure 4.3 we can see the increasing radiance measures at 06:00, 08:00 and 10:00 am (UTC time), due to the sun raising effect. The subfigures in the left show our representations of the

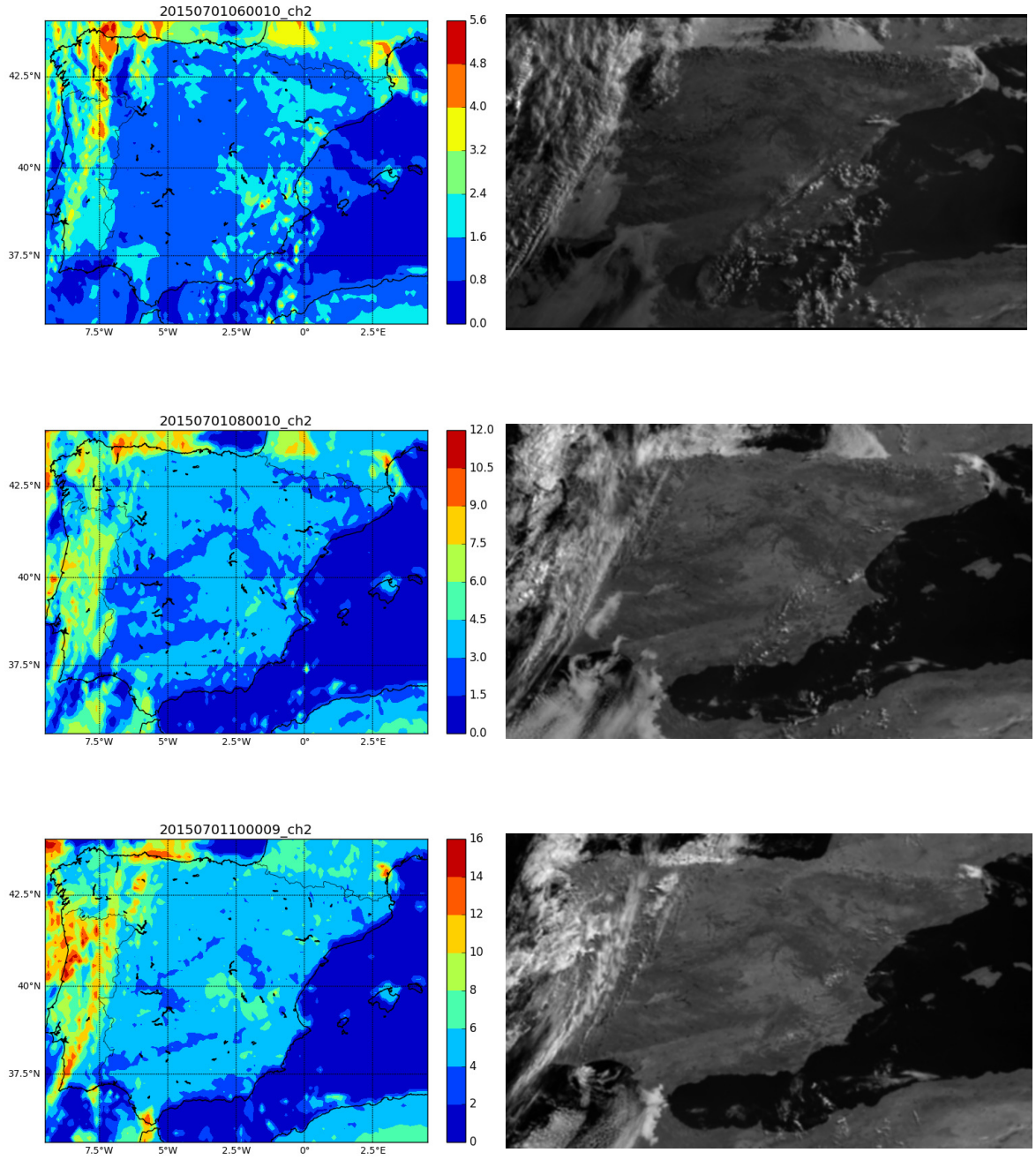

Figure 4.3: Evolution with respect to the hour for the Visible Channel 2, July 1st of 2015, at 06:00, 08:00 and 10:00 (UTC time).

measures and the ones in the right are the images downloaded from the satellite. It is very clear how the radiation is gradually increasing, which confirms the sun raising phenomenon.

Looking at the images in Figure 4.3 we can confirm what we expect, that is, the increasing in the measures is caused both by the natural sun raising and by the presence of cloudy areas, which appears to be *brighter* due to its *albedo*. This has an important impact in the interpretations we make, since a first naive approach was to expect a *brighter* area wherever the incoming radiation was greater. We will see later how this representation might vary depending on the wavelength of the channel we study.

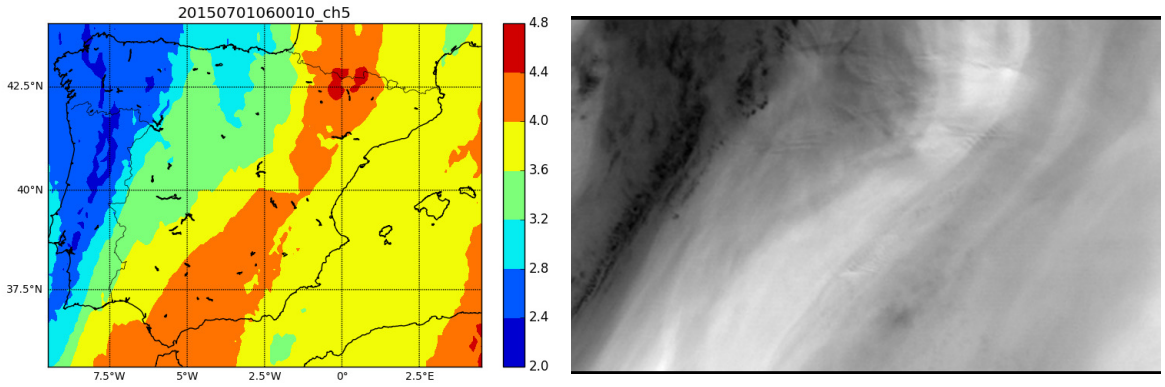

Figure 4.4: IR Channel 5, Water Vapour absorption band at 6.2 µm, July 1st of 2015 at 06:00 UTC time

As these channels measure the visible light spectrum bands, the main applications where they may be useful are:

- Recognition of cloud because of reflected sun radiation (brightest areas).
- Recognition of snow/ice because of reflected sun radiation.
- Discrimination of water and ice cloud.
- Recognition of Earth surface characteristics (soil and vegetation).

Some references, i.e. these EUMETSAT training presentations, lead us to believe that the most important information is extracted from the differences between several channels in relative bands.

4.3.2 Infrared Channels

Other interesting channels are not located in the visible band, having a possible impact in other completely different areas. The WV (Water Vapour) absorption is measured in both Channels 5 and 6, located at 6.2 µm and 7.3 µm, where the Earth radiation is dominant. The greyshades in Figure 4.4 show the WV content in the upper layers of the troposphere. As we can see in the subfigure at the right (in Figure 4.4), the darkest area correspond to the cloud mass we see in the VIS channels (in Figure 4.3). Since the Channel 5 captures the center of the Water Vapour absorption band, which is the thickest segment of this band, only the radiation from the upper stratospheric levels comes to the satellite. On the other hand, the Channel 6, which doesn't focus on the strongest absorption band, also captures radiation from lower levels.

These channels can help us to identify the different moisture presence in the upper and lower levels of the troposphere, even semi-transparent or transparent clouds.

Other interesting band might be the Ozone band, represented in the IR Channel 8, located at 9.7 µm, which measures the ozone concentration in the upper layers of the atmosphere, as shown in Figure 4.5.

The use and applications of this channel are not really different from the applications we can find in the usual IR channels. Nonetheless, the white areas that we can see in Figure 4.5

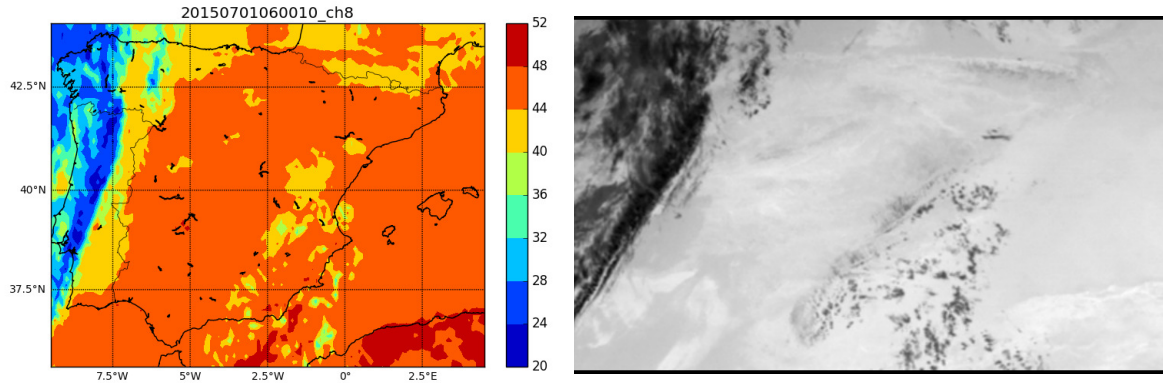

Figure 4.5: IR Channel 8, Ozone absorption band at 9.7 µm, July 1st of 2015 at 06:00 UTC time

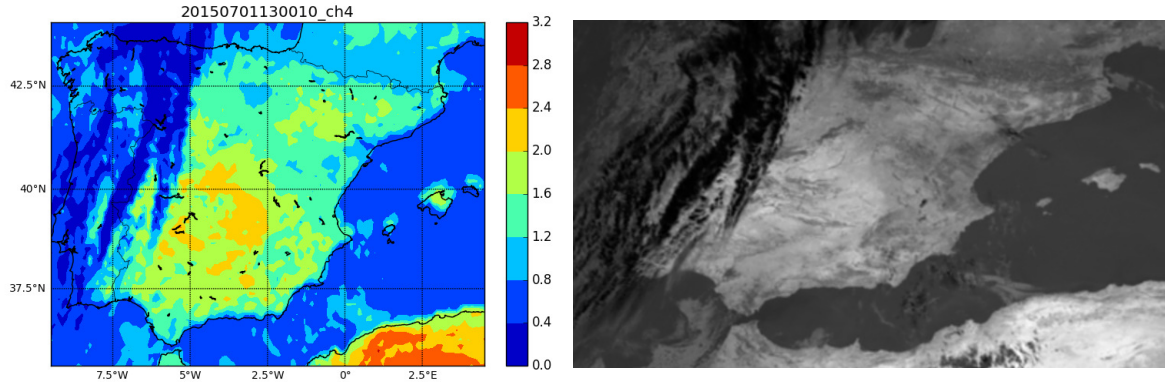

Figure 4.6: IR Channel 4, mixed Visible and Infrared band at 3.9 µm, July 1st of 2015 at 13:00 UTC time

indicate high stratospheric ozone in upper tropospheric heights. The physical meaning of these white stripes is related to high content of ozone, which absorbs radiation from below (Earth radiation) and emits radiation according to Kirchoff's Law of Thermal Radiation.

This kind of bands are better studied with global images, i.e., an image of Europe, where we could identify longer stripes and different areas with further relations to the meteorological phenomena. Nonetheless, ozone absorption band's EUMETSAT interpretation training presentation indicates that the important focus here is in the differences between the measures of different channels. Further relations on these combinations between channels might be interesting in future work.

Finally, another interesting band is located in Channel 4 at 3.9 µm, which is an special channel because both Sun radiation and Earth radiation are present, as we can see in Figure 4.1. This channel is presented in Figure 4.6.

The images shown here are interesting due to different facts:

- The IR band at 3.9 µm is not fully dominated by either Sun or Earth radiation, so it reflects both, which is an interesting characteristic.
- This means that the image can be shown and studied both as a visible image and as an

infrared one, which affects the representation of the greyshades in the image, showing the clouds as dark areas and the clear surfaces as brighter areas. At first sight, this channel could actually relate *brightest* areas with *irradiance* (radiance intensity), which could be interesting in future work.

Nonetheless, during night time this channel only reflects the emitted thermal radiation from the Earth, as we can expect, since no Sun radiation is reaching the Earth's area under the satellite at this time. Keeping in mind these facts will be definitely important, in order to not misinterpret the data.

Some applications where this channel may be useful are the following:

- Detection of low clouds and fog (during day and night)
- Detection of thin Cirrus (day and night) and multi-layer clouds (day)
- Cloud phase and particle size (day and night)
- Sea and land surface temperature (night)
- Detection of forest fires (day and night)
- Urban heat island (night)
- Super-cooled clouds (day and night)
- Cloud top structures (overshooting tops) (day)
- Sunglint (day), which is an effect that occurs when the sunlight is reflected off the ocean at the same angle that the satellite is viewing that the surface.

4.4 First Application on Photovoltaic Solar Energy

First of all, we would like to thank REE (Red Eléctrica de España) for providing us with information about the solar energy production in the Iberian Peninsula. After the first visualization of the data we are going to perform a more detailed analysis. This analysis will be focused on applying a first and basic machine learning model that will give us an introductory view of how the data behaves when trying to predict photovoltaic solar energy.

In this first approach we will aggregate the different variables over the entire Iberian Peninsula, ending up with a matrix of 23 columns (the 23 variables we presented in Section 3.4, coming from the different calibration modes for the 11 SEVIRI channels plus the cloud mask) and as many rows as hours in our date range. To perform a meaningful analysis we will need a fair amount of data as training set. Since we have three years of data, we will take two years as our training set, from January 1st of 2013 to January 1st of 2015. The final matrix has shape 16664×23, due to some missing hours (a complete year is formed by 8760 hours).

To build such a model, we will first analyze the correlation matrix of the different variables against the target; the photovoltaic energy production aggregated over the entire Iberian Peninsula, provided by REE. Moreover, since the photovolatic energy in dark hours is just zero, our model will only process daylight hours. To clip our data and drop the dark hours we have taken the sunrise time of Girona (Cataluña) and the sunset time of Santiago de Compostela (Galicia) at the fifteenth of each month. After dropping the dark hours, the matrix's final shape is 9830×23. The Figure 4.7 shows this correlation matrix, where we can see some interesting facts.

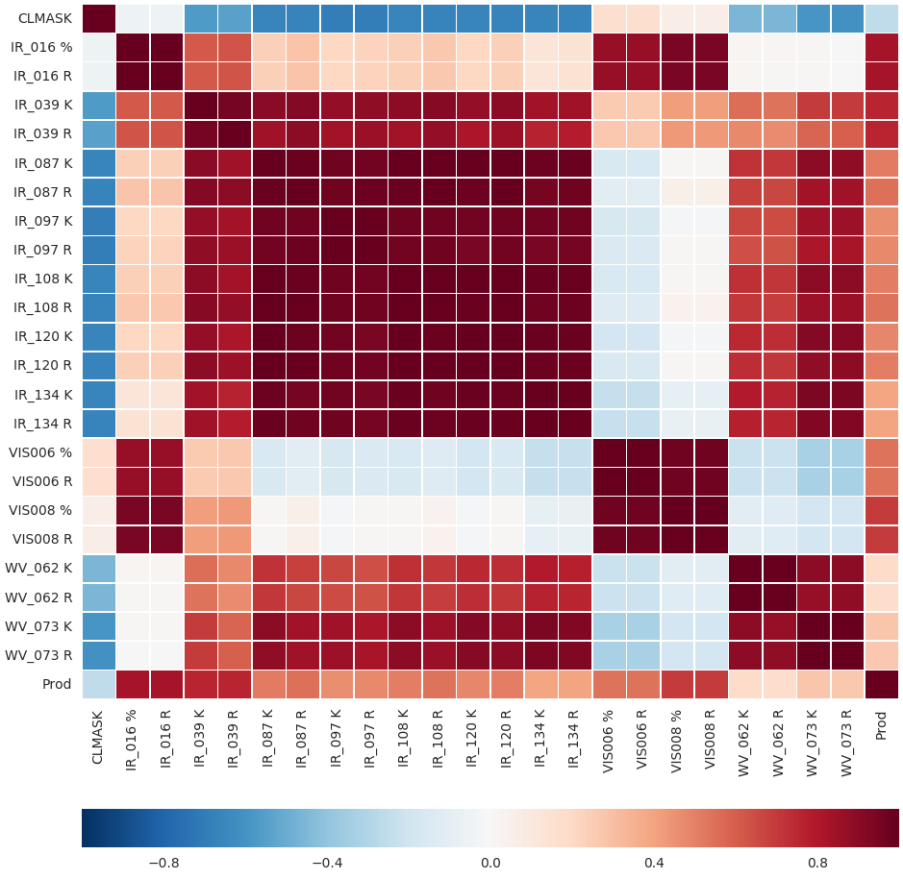

Figure 4.7: Correlation matrix of the aggregated variables over 2013

- All the infrared variables (those channels whose wavelength is over 3.6 µm) are highly correlated between them, with a very low correlation with the energy production. Since the Earth thermal radiation is dominant in these channels, their correlation with the photovoltaic solar energy (which is produced by sun radiation) must be lower.
- Moreover, the visible channels (those whose wavelength is between 0.6 µm and 1.6 µm) are negatively related to the infrared channels. To our understanding, this can be explained because a high value in the visible channels means that a high proportion of the incoming radiation is being reflected, therefore not increasing the surface temperature.
- As we can expect, the CLMASK (cloud mask) is also negatively related to the photovoltaic production, as well as to the infrared channels. This can be explained because a high cloud mask means a covered sky, avoiding the radiation to reach the ground level.
- The most correlated variables with the energy production are the variables IR_016 %, IR_016 R, IR_039 R, IR_039 K, VIS008 R and VIS008 %. As we explained in Section 4.2, this can be explained because the sun radiation is only dominant in these channels.

Next, we build a linear regression model [10] using the scikit-learn's implementation. This is the most basic regression model, for which we don't need to specify any meta parameter. We

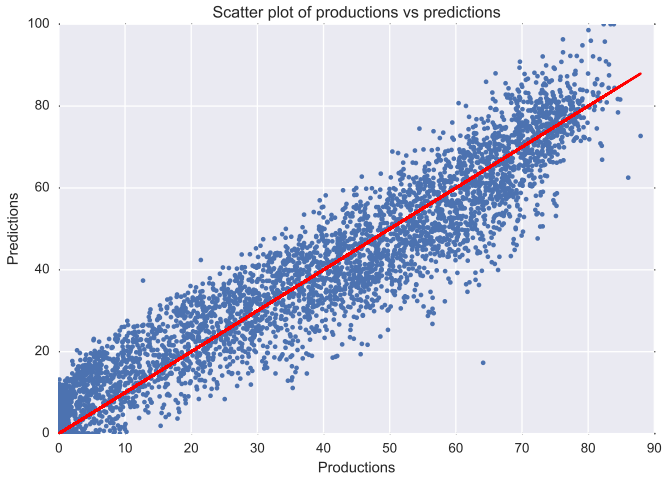

Figure 4.8: Scatter plot of productions against predictions

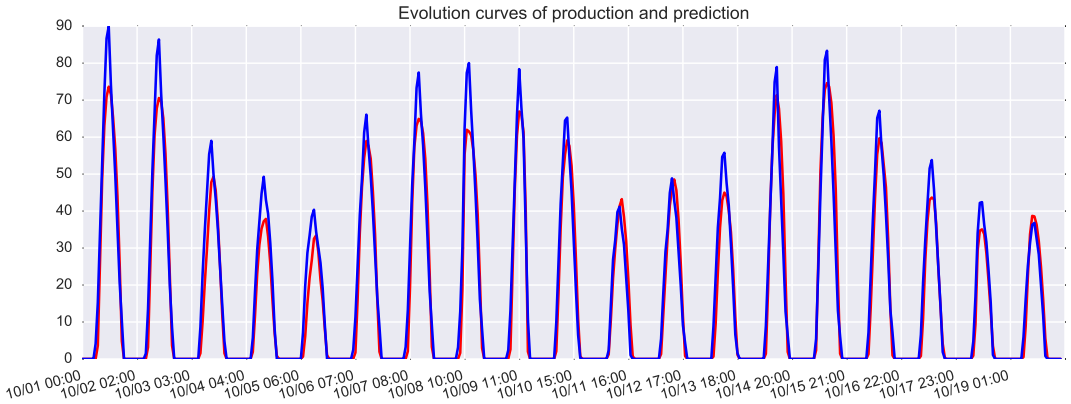

Figure 4.9: Evolution curves of predictions (blue) against productions (red)

call $\mathbf{X}$ to our input 9830×23 matrix, where 9830 is the number of hours in the model's training set, since January 1st of 2013 to January 1st of 2015, and 23 is the number of variables. In the same way, we call $\mathbf{Y}$ to our output vector of productions, normalized in the range $[0, 100]$, as a percentage of the total power installed. To fit our model, we first normalize the input $\mathbf{X}$ to zero mean and unit variance. Since our $\mathbf{Y}$ productions are actually percentage with respect to the total power installed, our predictions will also be percentage energy production clipped in the range $[0, 100]$.

In Figures 4.8 and 4.9 we can see how the test model behaves in terms of prediction accuracy. For this test model we have taken as test set the year 2015, since January 1st of 2015 to January 1st of 2016. The red line in Figure 4.8 represents the productions' line. The more compact the points are with respect to the productions' line the better the model will be. In this case, although we can see small deviations, the model's results are encouraging. In Figure 4.9 we show the evolution curves of predictions against productions with respect to the hour. From this figure, we can deduce that the hardest hours to predict are the mid day hours, specially for a linear model, and therefore the highest error values will be located on this slot. Since these are also the most important hours in terms of energy production, our goal is to find a model that minimizes the error in this time slot.

Error	Measure
MAE train	6.159
MAE test	6.018

Table 4.1: MAEs for train and test sets expressed in percentage with respect to the total power installed

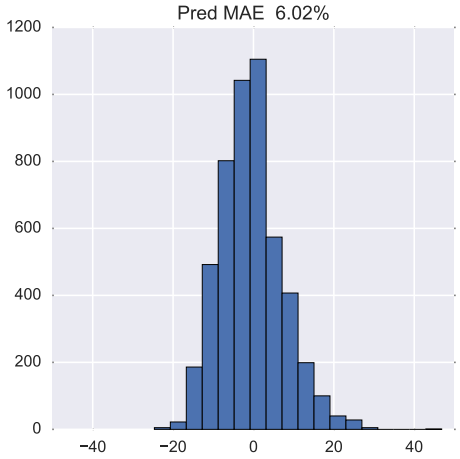

Figure 4.10: MAE histogram expressed as a percentage of the total power installed

To measure the error of the model we have used the *Mean Absolute Error* (MAE), which is given by the formula:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

where y is the real productions vector and $\hat{y}$ is the predictions vector. The MAE is a widely used metric in the energy prediction field, since it is also common to have the output as percentage with respect to the total power installed, thus making the MAE error easily translatable to energy production.

In Table 4.1 we present the MAEs for train and test. The train error is easily calculated from the predictions of the $\mathbf{X}$ input matrix itself. On the other hand, to calculate the test error, we recall that we have taken as test set the year 2015, since January 1st of 2015 to January 1st of 2016. Given the large test period, we can conclude that our results are consistent. The sharp shape in Figure 4.10 indicates that the majority of errors is concentrated around 0, encouraging the construction of more advanced models that may improve the overall prediction accuracy.

4.5 Conclusions

This chapter has showed the following important points:

- The information held by the different channels is radically different and we have to closely study each channel's specific details and meanings in order to find proper applications.
- A really important factor to take into account when visualizing the satellite images is the albedo.

- The sun radiation is only dominant in the visible channels. At first sight, these seem to be the most interesting channels for further applications in photovoltaic solar energy forecasting.
- A first data analysis, focused on a simple linear regression model, shows that, in fact, the most correlated channels to the photovoltaic energy production are the channels where the sun radiation is dominant.
- The results of this first machine learning model are encouraging, and lead us to explore more advanced models in further work.

The model we have developed has been a first exploratory experiment that, even though it is interesting from the machine learning point of view, it is far from being useful for real applications. In further work we will explore the idea of a *nowcasting* model, predicting energy productions in the range of $[1, 3]$ hours, which would be more useful.

In order to improve the accuracy, we could build more detailed models for specific time slots, i.e., a mid-day model for hours 9-15 and an evening model for hours 16-20, for instance. Other time slots combination may also be explored. A natural enhancement to our current model would be to not aggregate the different variables over the Iberian Peninsula, but instead consider each point in the grid as an independent variable. We believe that this approach would be more accurate, since aggregation usually tends to smooth the measure, often missing small details.

Other ideas that could improve the overall results may include studying more advanced models, such as the Lasso, Support Vector Regression (SVR) or Neural Networks (NN), whose popularity is currently growing.

5

Conclusions and Further Work

The renewable energies are a clear promise of clean and sustainable energies, helping the modern society to maintain an eco-friendly environment. A fundamental factor in this growing trend has been the computer science, and, more specifically, the data science. The increasing accuracy of the forecast models developed by some well known agencies such as ECMWF (European Centre for Medium-Range Weather Forecasts) has increased the interest of the energy companies in these new energy forms, leading to a continuous integration of these energies. Along these models there is the growing availability of related data to the field, with high technological capture devices such as satellites, helping the development of innovating research projects.

Along this work, we have introduced ourselves to the satellite radiation data field, downloading some interesting datasets that could be important to the construction of photovoltaic energy forecasting models. In order to understand the products we downloaded, we have made a first data interpretation, showing several figures and colormaps, explaining the characteristics of the radiation channels involved. After visualizing the data, and in a first approach to build a machine learning model, we have developed a simple linear regression model, using the Python's scikit-learn implementation. With this model we have been able to predict photovoltaic energy production obtaining promising results, a 6.02% MAE test error.

Despite this encouraging results, we have many more possibilities to study in further work. First thing to notice is that we have built this model with equally aggregated values over the entire Iberian Peninsula. Nonetheless, not every region in Spain contributes equally to the overall photovoltaic energy production. Therefore, a first potentially interesting improvement would be to aggregate the different variables according to the weight each region has on the overall energy production, thus ending up with a finer approach. A combination of both models, the arithmetic average and the weighted one, could be another step towards a stronger model. Moreover, another approach would be to not aggregate the different regions' variables, having instead each region as an independent component. This approach would lead us to evaluate which regions are better correlated with the overall energy production.

Our belief is that a model taking into account every point in the grid as an independent component would make a better model, since the aggregated values tend to smooth the final average, missing some important local characteristics that could have an impact in the overall production. Thus, in future work we will explore a model with as many variables as points in the grid (we have 3391 points within the administrative Spain's peninsular boundaries). Nonetheless,

the large amount of variables that this model would have (23×3391, being 23 the number of variables after the calibration of the 11 SEVIRI channels plus the cloud mask, and 3391 the number of points in the Spain's peninsular grid) leads us to think of models with shrinkage methods [10]. Shrinkage methods are still linear regression models that shrink the regression coefficients by imposing a penalty on their size. Among them, we would consider studying the Lasso method, which does a kind of continuous subset selection putting directly to zero those coefficients irrelevant to the prediction output, regularized by the shrinkage factor λ. Strategies to choose the optimal λ will also be studied in future work.

Putting aside these linear regression models, we could think of more advanced ones, such as Support Vector Regression (SVR). Other interesting non-linear models may be Neural Networks (NN), which are now in a rising trend.

As well as investigating new and more advanced machine learning models, we could also think of downloading and selecting new datasets. As we earlier presented in Chapter 3, EUMETSAT and some of its Satellite Application Facilities (SAFs) have a large amount of products on radiation. At first sight, a clear-sky radiation model may be interesting, applying later a cloud correction. As we read in [2], with a clear-sky model it is possible to estimate the irradiance at a certain point. Aside of our current products we could also explore the possibility of downloading some of the following datasets:

- *Aerosols.*
- *Clear-Sky Radiation.*
- *Ozone.*

As a final conclusion, with this work we have introduced ourselves in the satellite radiation data field, getting an introductory view of their properties. This first work has allowed us to know the main agencies working with satellite data and their most important characteristics when it comes to spatial and time resolution. With the first analysis of these data we have understood how the radiation at different wavelengths is related to the photovoltaic energy production and a first linear regression model has been presented with encouraging results, opening a wide range of possible improvements to be studied in further work. Furthermore, we have developed a software platform to handle and shape the data towards building and testing machine learning models. This platform is described in Appendix A.

Bibliography

[1] REE. Red eléctrica de españa: Demanda de energía.

[2] R. W. Mueller, C. Matsoukas, A. Gratzki, H. D. Behr, and R. Hollmann. The cm-saf operational scheme for the satellite based retrieval of solar surface irradiance — a {LUT} based eigenvector hybrid approach. *Remote Sensing of Environment*, 113(5):1012 – 1024, 2009.

[3] A. Hammer, D. Heinemann, C. Hoyer, R. Kuhlemann, E. Lorenz, R. Müller, and H. G. Beyer. Solar energy assessment using remote sensing technologies. *Remote Sensing of Environment*, 86(3):423–432, August 2003.

[4] C. Rigollier, M. Lefèvre, and L. Wald. The method heliosat-2 for deriving shortwave solar radiation from satellite images. *Solar Energy*, 77(2):159 – 169, 2004.

[5] EUMETSAT. Msg-1/seviri solar channels calibration. *Commissioning Activity Report*, pages 1–39, 2004.

[6] EUMETSAT. Conversion from radiances to reflectances for seviri warm channels. *EUM/MET/TEN/12/0332*, pages 1–6, 2012.

[7] A. P. Trishchenko. Solar irradiance and effective brightness temperature for swir channels of avhrr/noaa and goes imagers. *Journal of Atmospheric and Oceanic Technology*, 23:198–210, 2006.

[8] CM-SAF. Meteosat solar surface radiation and effective cloud albedo climate data records - heliosat. *SAF/CM/DWD/ATBD/Meteosat_HEL*, pages 1–28, 2014.

[9] J. A. Coakley. Reflectance and albedo, surface. *Elsevier*, pages 1914–1923, 2003.

[10] T. Hastie, J. Friedman, and R. Tibshirani. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, 2009.

[11] World Meteorological Organisation. *Guide to the WMO Table Driven Code Form Used for the Representation and Exchange of Regularly Spaced Data In Binary Form*. WMO GRIB Edition 2. WMO, 2003.

[12] EUMETSAT. On differences in effective and spectral radiance msg level 1.5 image products. *EUM/OPS-MSG/TEN/08/0161*, pages 1–11, 2008.

[13] EUMETSAT. A simple conversion from effective radiance back to spectral radiance for msg images. *EUM/OPS-MSG/TEN/07/1053*, pages 1–7, 2008.

[14] J. L. Bentley and T. A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, C-28(9):643–647, September 1979.

[15] Lindi Liao, editor. *Proceedings of the 2nd International Conference and Exhibition on Computing for Geospatial Research & Application, COM.Geo 2011, Washington, DC, USA, May 23-25, 2011*, ACM International Conference Proceeding Series. ACM, 2011.

[16] Russ Rew. Accessing netcdf data by coordinates, 2013.

[17] C. Carter. Great circle distances, computing the distance between two points on the surface of the earth. *SiRF*, pages 1–6, 2002.

[18] R. A. Schumacker, B. Brand, M. G. Gilliland, and W. H. Sharp. Study for applying computer-generated images to visual simulation. Technical report, DTIC Document, 1969.

A

Software Platform

A.1 Introduction

To properly structure and organize the large amount of data we have downloaded, we have developed a software platform addressing these needs. This platform is the base system for our future work, focused on modeling the different variables oriented to the photovoltaic solar energy field.

This chapter is organized in four main sections:

- Inter-calibration process. This first section deals with the previous processing we need to perform to the data in order to extract all the variables described in Section 3.4.
- Coordinate's referencing methods. This section will cover the methods we have implemented to access the different points of the grid in which the raw data is structured. Without this method we wouldn't be able to extract the measures from each point of the grid, which makes it an essential step before moving on to further processing.
- Conversion process to `myp` format. To store the data in a friendlier way, we have used a format designed by Cátedra UAM-IIC, namely `myp`, standing for *modeling and prediction*. It holds a matrix–like structure that only contains the information we need, without the overhead that NetCDF or GRIB necessarily contains.
- `DataMatrix`. After converting the different products to the `myp` format, we have developed this software platform specifically designed to keep the data organized and perform different models. This is the base system for our future work.

A.2 Inter-Calibration Process

The calibration process will allow us to extract meaningful information from the raw *pixel count* information, which, we recall, is the raw measure directly taken by the Satellite's instrumentation.

A.2.1 `Calibrator` Class

The goal we pursue with this class is to provide a user-friendly interface to make easier the decoding of the several variables contained in the raw SEVIRI measures. As we mentioned before, the variables we will be computing here are:

1. Effective Radiance.
    - Regular Channels.
    - Absorption Bands.
2. Reflectance for visible light channels.
3. Brightness Temperature for infrared channels.

To achieve that, this calibrator will receive a parameter that indicates the kind of decoding to perform, depending on the channel we want to inspect. The different calibration modes can be applied to the channels following the next restrictions:

- Effective Radiance can be extracted from all the eleven channels.
- Reflectance can be extracted *only* from the three visible channels.
- Brightness Temperature can be extracted *only* from the eight infrared channels.

As we explained in Section 3.4, each variable must follow a specific calibration process, which is essentially the application of the formulas detailed in previous sections for the corresponding channels.

Implementation

Although this code will be available in the Appendix B, we are presenting here a brief overview. It includes a main class named `Calibrator`. The most important method it contains is `calibrate`, which performs the actual calibration on the desired channel. This method takes as an argument a NetCDF structure and it returns a `Numpy` `maskedArray`.

Depending on the selected calibration *mode*, we will receive a *Brightness Temperature*, *Effective Radiance* or a *Reflectance* matrix for the given channel, based on the criteria previously presented for each channel.

EUMETSAT has approached the design and codification of a software with the same purpose, namely `mipp`. The reason that kept us from using the software itself is that it doesn't return the calibrated data, but instead its objective is generating RGB (images) products for meteorological remote sensing, which differs from our data processing goals. Due to this, we have adapted the class to match our output requirements and data processing standards.

The most important issue we have faced during the data processing step is the addressing of the different coordinates' systems that may be used in a curved surface. More specifically, this happens due to the `native` format EUMETSAT implements on its data, where the file is not indexed by latitude and longitude pairs, but by lines in the grid file (as stated in GRIB Edition 2 documentation [11]), so it is hard to properly address a desired point. This problem is present in both products we will use, the CloudMask and the SEVIRI data.

In order to better understand the methods and logic behind this calibration, we refer to [5, 6, 7, 12, 13]. The code of the calibrator is available in the Appendix B, in Listing B.9.

A.3 Coordinate's Referencing Methods

This referencing method appears as a need to reference points in curved surfaces, a spatial organization that is common in some EUMETSAT products, given its lack of high-level data processing algorithms to convert the file to a friendlier structure. This problem does not exist in products provided by the SAF network centers, like CM-SAF, because of their higher data processing specialization.

As a result of this low-level conversion in EUMETSAT products, the final NetCDF file contains a matrix to reference the latitudes and another one for the longitudes. These matrices are a representation of the curved structure of the Earth, as they are stored in the rotated GRIB file. To solve the issue of referencing coordinates in a curved surface we have written this code (see Listing B.4), based on computing K–D Trees [14] to store the representation matrices for the latitudes and longitudes, and have this way a much easier method to reference a certain pair of coordinates. For more information about the math behind the method, look at [15, 16].

To correctly access the coordinates in the grid, we have studied several algorithms and methods, taken from [16], and available in the programmer's Appendix B, which we discuss next.

A.3.1 Algorithms and Data Structures

These algorithms, namely `naive_slow`, `naive_fast`, `tunnel_fast` and `kd_tree`, compute the nearest coordinates to the point we want to access. This section will follow a bottom-up structure, starting with the least accurate algorithm and ending up with our final choice, discussing its benefits and the reasons behind this choice.

Naive Slow

This first method studies the Euclidean approach by calculating the Euclidean square distance, assuming a flat Earth, which gives wrong values to the distances as we move towards the poles. Nonetheless, it is fair enough as a first naive approach.

The implementation is in the Listing B.1. As we can see by inspecting the referenced code, this computes the distances one point at a time, what makes the code run really slow.

Naive Fast

A second version of the same algorithms uses `Numpy` matrices, which is much faster due to the great optimization `Numpy` has achieved in its structures, but it still assumes a flat Earth, so the result will be still wrong. Due to the use of `Numpy` methods, we see a more readable code (Listing B.2), taking advantage of the many features the library gives us.

Tunnel Fast

Despite of the fair enough results that the naive algorithms might give us in certain circumstances, specially those in the equator or near the equator, they are not good enough for other zones, such as the poles, so we could think of another approach that actually takes the Earth as an sphere. This algorithm is known as *tunnel distance for spherical spaces* [17], whose approach is to compute the distance between two points tracing a line through the *3*-dimensional space that joins the two points.

```
def kdtree(self, latvar, lonvar):
    rad_factor = pi / 180.0 # for trignometry, need angles in radians
    latvals = latvar[:] * rad_factor
    lonvals = lonvar[:] * rad_factor
    shape = latvals.shape
    clat, clon = cos(latvals), cos(lonvals)
    slat, slon = sin(latvals), sin(lonvals)
    clat_clon = clat * clon
    clat_slon = clat * slon
    triples = list(zip(np.ravel(clat * clon), np.ravel(clat * slon), np.ravel(slat)))
    return cKDTree(triples)
```

Listing A.1: K–D Tree computation from latitudes and longitudes matrices

As we can see in its implementation, in Listing B.3, some angular factors are considered, which makes this a correct solution, assuming an spherical Earth, a much better approach than the naive one.

KDTree Fast

Finally, this last algorithm is based on computing K–D Trees, which gives us a correct and fast approach to operate on large amounts of data, but not so good results on a small amount of queries, as we shall see at the end of this section.

As an overview, the K–D Trees [14] are space-partitioning data structures for organizing points in a k-dimensional space based in euclidean partitions. They are a special case of BSP (Binary Space Partitioning) trees, which is a data structure developed in the context of image processing [18].

We recall that we are mainly using NetCDF meteorological files to access our data, at least at this early stage. These files provide us with a multidimensional access through index tags. Nonetheless, a more user-friendly way to address the data is through the coordinates themselves, trying to calculate the correct indices. However, this kind of access is not always the most efficient or optimal one, since calculating the corresponding indices for the coordinates we want to access is neither easy nor trivial.

In the case we are working on, the files containing radiation-related data are converted from a NetCDF file where the access to every `latitude, longitude` pair is organized through $3,712 \times 3,712$ matrices, that is, more than 13 million cells inside the matrix. The computation of such matrices takes time and is not easy, so finding an efficient method is essential.

For the use of K-D trees in Python, we are using the `Scipy.spatial` module, which contains a variety of API access methods to operate with K–D Trees and many other data structures. This module provides us with an efficient search method on K–D Trees to locate the nearest point in a grid to a desired origin. The parameters we are passing to cKDTree (which is the K–D Trees implementation in Python's `scipy` package) are the latitude and longitude matrices. Nonetheless, the coordinates are not stored *as is*, but a transformation to their radian–equivalents. The process is described in Listing A.1.

Since the information stored in the K–D Tree is the transformation from a pair `latitude, longitude` to their radian–equivalents, the first step to get the corresponding indices of a given point is to perform on it the same processing. Thus, we are transforming a pair of coordinates, consider `' (35, 7)'` as an example, by applying the same method we referred before, described in Listing A.1. Now that we have transformed the coordinates we are looking for to the same units the K–D Tree contains, we can search for it, getting in return the *closest point* actually present in the grid.

Method	Setup (ms)	Query (ms)	Setup + 10000 queries (sec)
naive_slow	3.76	7790	77900
naive_fast	3.8	2.46	24.6
tunnel_fast	27.4	5.14	51.4
kdtree_fast	2520	0.0738	3.3

Table A.1: Algorithms comparison of general performance

KD-Tree Implementation

The Python code is, as the other algorithm's, in the Appendix B, in Listing B.4.

As one can see by inspecting the source code, the final code has been embedded into a Python class, adding some useful methods to operate on the structure that allow a better access to the methods outside its file.

A.3.2 Comparisons Between Algorithms

The Table A.1 shows a comparison of some experimental times to perform a series of queries by all methods explained in the previous section. From this information we can extract some conclusions, that we present next.

We can clearly see how the K–D Tree algorithm is much faster and efficient as the amount of queries increase, while, in the contrary, the setup time is too high to be worth it for a small number of queries. Since we use it for a very large amount of queries, this method fits our needs. Note that we are not comparing here the correctness of the algorithms, which has been discussed in the previous sections. If we are not searching for a large amount of points, it is worth it to use the `naive_fast` or the `tunnel_fast` methods, depending on the zone we are studying and the correctness we need.

A.4 Processing Raw Data

A.4.1 Motivation and Goals

After downloading and organizing the data, the first issue that arises while trying to work with it is the different formats the products have. This leads to a harder-to-write program to handle all the data into a single structure. To solve this, we used a new format designed by Cátedra UAM-IIC whose main purpose is to make easier subsequent processes. Therefore, instead of writing the harder program, we wrote a set of conversion programs to this new format.

We can summarize the main reasons that lead us to use this new format and the conversion programs as follows:

- The products downloaded are not in the same format, which makes difficult to write a single program to process all the data.
- The original formats used to access and operate with the data are too complex to easily handle them. The overhead of information makes these formats good for research and storage, but it is unnecessary for our data processing.

Thus, these conversion programs aim to put the original data in a single format that we can use to write a unified program to read and process all the data. We will describe each program and its structure in the next section, as well as the format.

A.4.2 Format Description

The data will be ideally organized in a matrix-like structure. To have a visual understanding of what is going on, this structure will have as rows the different time-stamps we have data for, and as columns we will set the different variables and coordinates in a certain grid. Nonetheless, to gradually reach this final step we are making a two-phase conversion. This first step will cover the conversion from GRIB or NetCDF to a plain text format, named *myp*, specifically designed to manage these data.

This format will not have any header information, as every file will follow a fixed structure defined next. The data will be organized following a horizontal sweep; that means that for each pair of coordinates (row), we will have as many columns as variables are defined for that point. Furthermore, this will also introduce temporal structure in the columns. This will lead to a very long file in terms of lines, due to the fine-grained resolution we have.

The format's structure is described by the following pattern:

Lat	Lon	v1	v2	...	vn

Despite this not being the final format with which we will build machine learning models, we will store the data in this way because it is quite intuitive and lightweight. The order we are putting the variables in columns is:

ch1	ch2	...	ch11	ch1_r	...	ch3_r	ch4_bt	...	ch11_bt	cloud_mask

where the variables `ch1`, ..., `ch11` are the different channels from the SEVIRI satellite instrumentation, measuring the *effective radiance*, going from wavelengths 0.6 µm to 13.4 µm, the variables named `chx_r` measure *reflectance* on the given channel (from `ch1` to `ch3`) and the ones named `chx_bt` represent the *brightness temperature* on a given infrared channel (from `ch4` to `ch11`). Finally, the `cloud_mask` measures the *cloud mask* for a certain point. This makes a total of 25 columns per row, counting also the coordinate columns.

These files will be named following the pattern `YmdH`, which is a Python pattern meaning `Year month day Hour`, having one file per hour, with the purpose of making our further processing easier.

A.4.3 Conversion Programs to the MYP Format

This section will cover the two different processes to convert our data to the `myp` format. On one hand, we will discuss the conversion from `NetCDF` to `myp`, and, on the other, the conversion from the `GRIB` format. Recall that the `NetCDF` files hold our radiation data, while the cloud-related data is stored in `GRIB` files.

From NetCDF to MYP

The NetCDF format is the one EUMETSAT uses to assemble the SEVIRI data, which is, as we saw in Chapter 3, our main data source. Nonetheless, despite of us currently having just one

```
myp = []
for (lat,lon),index in grid:
      row = [lon, lat]
      for chn_name in sorted(channels.keys()):
            for calibration in calibration_modes:
                  row.append(data[chn_name][calibration][0][index])
      myp.append(row)
myp = np.matrix(myp)
```

Listing A.2: Building up the myp structure

product using NetCDF, future variables and products are very likely to have this format too, so writing a generic program is a good approach that may be useful in future work in case we add more products to our platform.

Regarding the discussion in Sections 3.3.1, A.2.1 and A.3 about the NetCDF format, the calibration and the problem of coordinates' accessing, we need to go through several steps to make the conversion. These steps involve *decoding*, *calibration*, *coordinate accessing* and finally building the matrix. Basically, the algorithm will work as follows:

- Read up the original data.
- Build a K–D Tree to retrieve the desired grid.
- Calibrate the data to get the final variables.
- Frame the final variables into a matrix.

The reading process is based on the NetCDF file format, which turns to be really easy by using the Python library netcdf-python. The code is presented in Listing B.6.

The K–D Tree's structure and purpose was explained in Section A.3 and it will be the core method to successfully read the original data.

For the data calibration, we recall the `Calibration` class, described in Section A.2.1. To gather all the possible calibrations (Section 3.4) we have used a Python dictionary, using as key the sort of calibration (an integer in the range $[0, 2]$), namely, no calibration, *radiances* or *reflectance/bt*, respectively.

Finally, we build the matrix by iterating through the different calibrations' output stored in the dictionary. A brief snippet describing this process is presented in Listing A.2.

From GRIB to MYP

The GRIB library written for Python is not as high level as the NetCDF is, so our methods to access the data are based on a typical file input/output transaction. Nonetheless, the EUMETSAT processing of this product and the format's structure itself makes the coordinates projection easier to read, without needing a `K-D Tree` for the accessing and decoding.

The algorithm scheme is essentially the same we explained for NetCDF conversion:

- Open the file using the `pygrib`'s file descriptor.
- Retrieve the data inside the desired grid.

- Filter the grid to extract the resolution we want.
- Build the final matrix.

`Pygrib`, the Python library for accessing GRIB files, provides us with a method that solves the problem of referencing coordinates in a curved surface, namely `grb.data(lat1, lat2, lon1, lon2)`, which allows us to query for data within the region defined by the coordinates we pass as arguments. This method returns three structures, the data matrix, the latitudes vector and the longitudes vector. As we will do with the NetCDF conversion code, this code will be available in Appendix B, in the Listing B.6.

With this conversion process we solve two issues at once:

- Getting the data in a matrix and,
- Retrieving the desired subgrid.

A.4.4 Discussion on Storage

On one hand, the creation of the *myp* format translates directly into an increase of the storage needs. Since we are using Python to develop the code, and our new format is essentially a matrix, we will use the `Numpy save` method to store it. Each one of these files will be about 1.7MB, which actually saves some space compared to other formats to save the matrix, such as `Numpy savetxt`. Apart from the already downloaded data (130GB for 3 years), these new `Numpy` files make an extra 45GB, which comes from generating three years (8,760 hourly files each) of 1.7MB (after combining the `myp` SEVIRI file with the `myp` CloudMask file) of these files, counting a total of 175GB (130 + 45GB) including all the data.

On the other hand, the overall performance might be affected by this added conversion process. Nonetheless, since converting one file takes less than 10 seconds, and our overall processing will take longer than that, we can iteratively convert the data without adding a large overhead.

A.5 DataMatrix

A.5.1 Introduction and Motivation

With the *myp* format we have developed a first data processing to handle the original products downloaded from EUMETSAT and put them in a more suitable shape. Nonetheless, this format is not prepared to directly build models and develop further processing. To achieve that goal, we need another step that gathers this *myp* files into one structure specifically designed to work on machine learning models and deeper data science.

To build this new structure, we have developed the `DataMatrix class`, which is a Python class that prepares the *myp* files for further processing, focused in modeling and predicting in future work. The core idea that leads this development is based on the following criteria that we need to build models:

- The data must be organized in a temporal sequence.
- The data must be easily accessed and indexed, for which we will need some fast indexing methods.

- We should also be able to extract desired subsets of data to work on a more specific region or variables.

Thus, `DataMatrix` will be based on a `Pandas' DataFrame`, which is an organized structure designed to maintain an indexed matrix. The main advantages, matching our requirements, that this structure will give us are the following:

- Time-stamps indices to keep our data sorted and intuitively organized.
- Tags for each column that help us to quickly identify what kind of data each column stores.
- Since the `Pandas` library is based on `Numpy`, we will gain features such as support for indexing, slicing and handling of matrices and generic `nd-array` structures.

Moreover, since our `DataMatrix` is built on top of a `Pandas' DataFrame` and `Numpy`, it will provide us with a higher abstraction level when it comes to deal with the inner structure's methods. This will allow us to focus on our specific problem, without explicitly having to handle the raw `Pandas'` interface. An example of that are the `slicing` methods, in which we won't have to indicate *indices* to slice but *variables* or *dates*, which are a friendlier wrapper.

The `DataMatrix` class has the following methods and attributes.

- Attributes
  1. `dataMatrix`, which is the core structure in the class. Holds the entire `DataFrame`.
  2. `variables`, which holds a list of the variables contained in the `dataMatrix`.
  3. `grid`, which is the grid for which the `dataMatrix` was generated.
  4. `channels`, a constant dictionary associating channels with abbreviations, i.e., `VIS006: ch1`.
  5. `calibration_modes`, a list of the different calibration processing we can perform on our data.
- Methods
  1. `generateMatrix`, which will create the inner matrix in a `Panda's DataFrame` structure.
  2. `queryIndex`, which will return a subset of the data within the given dates.
  3. `queryCols`, which will return a subset of the data by indexing columns, that is, variables in a grid.
  4. `saveMatrix`, which will save `dataMatrix` to a `Numpy` file.
  5. `querySubgrid`, which is a domain's specific method that will extract the data for a given subgrid.
  6. `querySubgridDates`, which will query the data for a given subgrid within a given date range.

Next, we are going to describe briefly the main methods of `DataMatrix`.

A.5.2 DataMatrix's Methods

This section will introduce some of the core methods. Some of these methods are just wrappers to other `Panda's` or `Numpy's` methods, but we usually incorporate new useful parameters related to our specific data processing and treatment, as we previously explained. We will focus on the three main methods: matrix generation, sub selection of a date range and sub selection of variables in a subgrid.

Matrix Generation Method

This first method of the class `DataMatrix`, named `generateMatrix`, will receive as input a set of `myp` files. This set will contain the files for the desired period to build the matrix with. As we have hourly `myp` files, we will have about 8,760 files for one year. Along these files, we also receive a list of *variables* to tag the columns of the inner `DataFrame`. The process of generating the `DataMatrix` takes about one hour to load one year of data.

To build the inner `DataFrame`, which will be stored as a class attribute named `dataMatrix`, we follow an iterative model based on the following steps:

1. Load each hourly `myp` file as a `Numpy` matrix.
2. Generate hourly indices for the `DataFrame`.
3. Generate column tags for the `DataFrame`.
4. Iteratively concatenate the partial `DataFrames` generated in the previous step.

The resulting `DataFrame` is then stored as an instance variable of the class, namely `dataMatrix`, along with some other useful parameters, such as the *date* and the columns' tags.

Querying Date Ranges Method

One of the most useful functions we have written is `queryIndex`. This method allows us to query the `DataFrame` for the data contained in a certain date range. The process is represented in the Listing A.3. The indices will have the same shape as the date format we used in previous methods, `YmdH`.

Querying Variables in a Subgrid Method

Since we won't be usually studying the whole grid at once, but instead smaller subgrids, a quite useful operation would be to extract the corresponding variables for a given subgrid. This operation is realized by the `queryCols` method. This method will receive as parameters the *upper left corner*, the *bottom right corner*, the *resolution* of the grid to study and a list of variables to extract.

The process is illustrated by Listing A.4.

```
def queryIndex(self, start_date, end_date, nsteps=8):
    '''
    Generates a list of indexes to address the pandas DataFrame.

    This indexes are generated by checking the meteorology files
    we have inside the matrix. for each of this files, it calculates
    the timestamps by using the nsteps parameter, which indicates
    how many three-hourly intervals the file contains.
    '''
    index = []
    for date in pd.date_range(start_date, end_date, freq='H'):
        files = self.path + date.strftime(self.date_format) + '.seviri.myp.npy'
        filec = self.path + date.strftime(self.date_format) + '.clmask.myp.npy'
        if os.path.isfile(files) and os.path.isfile(filec):
            ts = date.strftime('%Y%m%d%H')
            index.append(int(ts))
    return index
```

Listing A.3: Method queryIndex that extracts submatrices by date ranges

```
def queryCols(self, grid=None, latlons=None, tags=[]):
    '''
    Generates a list of the tags used to address each of the columns
    of the pandas DataFrame.

    Each tag is the string composition of the coordinates at the
    point and the name of the variable it refers to.
    '''
    if grid:
        lon_l, lon_r = grid.get_lons()
        lat_l, lat_r = grid.get_lats()
        res = grid.get_res()
        latlons = [(i, j) for i in np.arange(lat_l, lat_r+res, res)
                   for j in np.arange(lon_l, lon_r+res, res)]

    return np.array([['({0}, {1}) {2}'.format(j, i, tag)
                      for tag in tags]
                      for i, j in latlons]).flatten()
```

Listing A.4: Method queryCols that extracts a submatrix by indexing columns

A.5.3 Conclusions

Summarizing, `DataMatrix` will help us to organize the data in a structured way and, by providing us with useful methods, we will gain a new abstraction easier to deal with, rather than raw `Pandas'` or `Numpy's` methods.

In this last sections we have gone through the process of first processing the downloaded data from a raw and hard to use format to a more domain oriented and specific one. This process has been divided in two phases:

1. Converting the data to *myp* format, which, as we saw, is a matrix-like structure that helps us to handle and to store the data for further use.
2. Building the `DataMatrix`, which will finally gather all the hourly *myp* files into one structure. This structure is organized by timestamps as rows, which allows us to keep the data sorted, and variables by coordinates as columns, which helps us to easily identify what each column represents.

This platform will serve as the support system for our future work, focused on building machine learning models in the field of the photovoltaic solar energy.

B

Code Snippets

The following listings contain the different Python programs that we have developed in order to accomplish the different steps of our data processing. For further details and complete descriptions about this processing, refer to Appendix A.

```
import numpy as np
import netCDF4

  def naive_slow (latvar,lonvar,lat0,lon0):
    '''
    Find ``closest'' point in a set of (lat,lon) points to specified point
    latvar - 2D latitude variable from an open netCDF dataset
    lonvar - 2D longitude variable from an open netCDF dataset
    lat0,lon0 - query point Returns iy,ix such that (lonval[iy,ix] - lon0)**2 +
    (latval[iy,ix] - lat0)**2 is minimum. This ``closeness'' measure works
    badly near poles and longitude boundaries.
    '''

    # Read from file into numpy arrays
    latvals = latvar[:]
    lonvals = lonvar[:]
    ny,nx = latvals.shape
    dist_sq_min = 1.0e30

    for iy in range (ny):
      for ix in range (nx):
        latval = latvals[iy, ix]
        lonval = lonvals[iy, ix]
        dist_sq = (latval - lat0)**2 + (lonval - lon0)**2

        if dist_sq < dist_sq_min:
          iy_min, ix_min, dist_sq_min = iy, ix, dist_sq

    return iy_min,ix_min
```

Listing B.1: Naive slow implementation to compute nearest coordinates in a grid

```
import numpy as np
import netCDF4

  def naive_fast (latvar,lonvar,lat0,lon0):
    # Read latitude and longitude from file into numpy arrays
    latvals = latvar[:]
    lonvals = lonvar[:]
    ny,nx = latvals.shape
```

```
    dist_sq = (latvals-lat0)**2 + (lonvals-lon0)**2
    minindex_flattened = dist_sq.argmin ()

    # 1D index of min element
    iy_min,ix_min = np.unravel_index (minindex_flattened, latvals.shape)

    return iy_min,ix_min
```

Listing B.2: Naive fast implementation to compute nearest coordinates in a grid, using Numpy abstractions

```
import numpy as np
import netCDF4
from math import pi
from numpy import cos, sin

  def tunnel_fast (latvar,lonvar,lat0,lon0):
    '''
    Find closest point in a set of (lat,lon) points to specified  point
    latvar - 2D latitude variable from an open netCDF dataset
    lonvar - 2D longitude variable from an open netCDF dataset
    lat0,lon0 - query point
    Returns iy,ix such that the square of the tunnel distance between
    (latval[it,ix],lonval[iy,ix]) and (lat0,lon0) is minimum.
    '''

    rad_factor = pi/180.0 # for trignometry, need angles in radians

    # Read latitude and longitude from file into numpy arrays
    latvals = latvar[:] * rad_factor
    lonvals = lonvar[:] * rad_factor
    ny,nx = latvals.shape
    lat0_rad = lat0 * rad_factor
    lon0_rad = lon0 * rad_factor

    # Compute numpy arrays for all values, no loops
    clat,clon = cos (latvals),cos (lonvals)
    slat,slon = sin (latvals),sin (lonvals)
    delX = cos (lat0_rad)*cos (lon0_rad) - clat*clon
    delY = cos (lat0_rad)*sin (lon0_rad) - clat*slon
    delZ = sin (lat0_rad) - slat

    dist_sq = delX**2 + delY**2 + delZ**2
    minindex_1d = dist_sq.argmin () # 1D index of minimum element
    iy_min,ix_min = np.unravel_index (minindex_1d, latvals.shape)

    return iy_min,ix_min
```

Listing B.3: Tunnel fast algorithm to compute nearest coordinates in a 3D surface

```
import numpy as np
import netCDF4
from math import pi
from numpy import cos, sin
from scipy.spatial import cKDTree

      def kdtree_fast (latvar,lonvar,lat0,lon0):
        rad_factor = pi/180.0 # for trignometry, need angles in radians

        # Read latitude and longitude from file into numpy arrays
        latvals = latvar[:] * rad_factor
        lonvals = lonvar[:] * rad_factor
        ny,nx = latvals.shape

        clat,clon = cos (latvals),cos (lonvals)
        slat,slon = sin (latvals),sin (lonvals)

        # Build kd-tree from big arrays of 3D coordinates
        triples = list (zip (np.ravel (clat*clon), np.ravel (clat*slon),
                             np.ravel (slat)))
        kdt = cKDTree (triples)
        lat0_rad = lat0 * rad_factor
```

```
        lon0_rad = lon0 * rad_factor
        clat0,clon0 = cos (lat0_rad),cos (lon0_rad)
        slat0,slon0 = sin (lat0_rad),sin (lon0_rad)
        dist_sq_min, minindex_1d = kdt.query ([clat0*clon0, clat0*slon0, slat0])
        iy_min, ix_min = np.unravel_index (minindex_1d, latvals.shape)

        return iy_min,ix_min
```

Listing B.4: KDTree fast implementation to compute nearest coordinates in a 3D surface

```
import numpy as np
import datetime
from math import pi
from numpy import cos, sin
from scipy.spatial import cKDTree

class KDTree(object):
    def __init__(self, ncfile, latvarname, lonvarname):
        self.ncfile = ncfile
        self.latvar = self.ncfile.variables[latvarname]
        self.lonvar = self.ncfile.variables[lonvarname]
        rad_factor = pi / 180.0 # for trignometry, need angles in radians
        self.latvals = self.latvar[:] * rad_factor
        self.lonvals = self.lonvar[:] * rad_factor
        self.shape = self.latvals.shape
        clat, clon = cos(self.latvals), cos(self.lonvals)
        slat, slon = sin(self.latvals), sin(self.lonvals)
        clat_clon = clat * clon
        clat_slon = clat * slon
        triples = list(zip(np.ravel(clat * clon), np.ravel(clat * slon), np.ravel(slat))
            )
        self.kdt = cKDTree(triples)

    def query(self,lat0,lon0):
        rad_factor = pi / 180.0
        lat0_rad = lat0 * rad_factor
        lon0_rad = lon0 * rad_factor
        clat0, clon0 = cos(lat0_rad), cos(lon0_rad)
        slat0, slon0 = sin(lat0_rad), sin(lon0_rad)
        dist_sq_min, minindex = self.kdt.query([clat0 * clon0, clat0 * slon0, slat0])
        iy_min, ix_min = np.unravel_index(minindex, self.shape)
        return iy_min,ix_min

    def querySubset(self, ullat, ullon, lrlat, lrlon, spatialRes):
        ilat, ilon, coords = list(), list(), list()
        for i in np.arange(lrlat, ullat, spatialRes):
            for j in np.arange(ullon, lrlon, spatialRes):
                iy, ix = self.query(i,j)
                ilat.append(iy), ilon.append(ix), coords.append((i, j))
        return zip(coords, zip(ilat, ilon))
```

Listing B.5: Final KDTree implementation as a Python class

```
import sys

from netCDF4 import Dataset
from KDTree import KDTree
from _Calibrator import _Calibrator
import pandas as pd
import os
import fnmatch
import argparse
import numpy as np
import pygrib
import cdo
import subprocess


def convertNetCDF(path, file):
    ncfile = Dataset(file, 'r')

    ns = KDTree.KDTree(ncfile, 'lat', 'lon')
```

```
    cal = _Calibrator.Calibrator(ncfile, "VIS006")
    calibration_modes = ['radiances', 'reflectances/bt']

    channels = {"VIS006": 'ch1',
                "VIS008": 'ch2',
                "IR_016": 'ch3',
                "IR_039": 'ch4',
                "WV_062": 'ch5',
                "WV_073": 'ch6',
                "IR_087": 'ch7',
                "IR_097": 'ch8',
                "IR_108": 'ch9',
                "IR_120": 'ch10',
                "IR_134": 'ch11'}
    data = {}
    for chn_nb in channels.keys():
        data[chn_nb] = {}
        for calibration in calibration_modes:
            data[chn_nb][calibration] = cal(calibrate=calibration, channel_name=chn_nb)

    grid = list(ns.querySubset(44.25, -9.5, 35.5, 4.75, 0.125))

    myp = []
    for (lat,lon),index in grid:
        row = [lon, lat]
        for chn_name in sorted(channels.keys()):
            for calibration in calibration_modes:
                row.append(data[chn_name][calibration][0][index])
        myp.append(row)
    myp = np.matrix(myp)

    filedate = ncfile.wmo_filename.split('_')[-1].split('.')[0][:-4]
    np.save(path + filedate + '.seviri.myp', myp) # hourly compressed np files


def round_to_value(number, roundto):
    return (round(number / roundto) * roundto)


def convertGRIB(path, file):
    pgfile = pygrib.open(file)

    for grb in pgfile:
        data, lats, lons = grb.data(lat1=35.5, lat2=44, lon1=-9.5, lon2=4.5)

    orig = zip([(round_to_value(x, 0.125), round_to_value(y, 0.125))
                for (x, y) in zip(lats, lons)], data)
    newlist = []
    seen = set()
    for (a, b) in orig:
        if not a in seen:
            newlist.append((a, b))
            seen.add(a)

    myp = np.matrix([(y, x, d) for ((x, y), d) in sorted(newlist)])

    filedate = grb.analDate.strftime('%Y%m%d%H')
    np.save(path + filedate + '.clmask.myp', myp) # hourly np file


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Convertion from NetCDF and GRIB to MYP
        ')
    parser.add_argument("path",help='Path from where recursively look for netcdf files')
    parser.add_argument("out",help='Path where to save the myp files')

    args = parser.parse_args()
    for root, dirs, files in os.walk(args.path):
        for file in sorted(files):
            if fnmatch.fnmatch(file, '*.nc'):
                fdate = file.split('_')[-1].split('.')[0][:-4]
                out    = args.out + 'seviri/'
                target = out + fdate + '.seviri.myp.npy'
```

```
                if not os.path.exists(target):
                    convertNetCDF(out, os.path.join(root, file))
            elif fnmatch.fnmatch(file, '*.grb'):
                fdate  = file.split('.')[0][-14:-4]
                out    = args.out + 'clmask/'
                target = out + fdate + '.clmask.myp.npy'
                if not os.path.exists(target):
                    convertGRIB(out, os.path.join(root, file))
            print(target)
```

Listing B.6: Conversion programs to convert from NetCDF and GRIB to the myp format

```
class AreaGrid(object):
    '''
    AreaGrid allows the creation and use of unconstrained grids for general
    purpose, similar to Grid. This kind of Grid is not attached to a farm, so
    its use is further flexible and useful in a lot more situations.
    '''

    def __init__(self, ullat, ullon, lrlat, lrlon, res):
        self.ullat = ullat
        self.ullon = ullon
        self.lrlat = lrlat
        self.lrlon = lrlon
        self.res   = res

    def get_lats(self):
        return self.lrlat, self.ullat

    def get_lons(self):
        return self.ullon, self.lrlon

    def get_lats_lons(self):
        latlons = []
        for i in np.arange(self.lrlat, self.ullat + self.res, self.res):
            for j in np.arange(self.ullon, self.lrlon + self.res, self.res):
                latlons.append([i, j])
        return latlons

    def get_res(self):
        return self.res
```

Listing B.7: AreaGrid class to handle grid abstractions in the final platform

```
import sys
from mpl_toolkits.basemap import Basemap
from netCDF4 import Dataset as NetCDFFile
import numpy as np
import matplotlib.pyplot as plt

from KDTree import KDTree


def draw_data(filename):
    nc = NetCDFFile(filename)

    ns = KDTree.KDTree(nc, 'lat', 'lon')
    index = list(ns.querySubset(44, -9.5, 35.5, 4.5, 0.125))
    nlats, nlons = (44-35.5)/0.125, (4.5-(-9.5))/0.125

    for chn in ['ch' + str(i) for i in range(1, 12)]:
        rad = nc.variables[chn]

        proj_data, x, y, size = np.zeros((int(nlats), int(nlons))), -1, 0, len(index)
        for coord, i in index:
            ypos = y % (size/nlats)
            if not ypos:
                y, x = 0, x + 1
            proj_data[x][ypos] = rad[i]
            y += 1

        m = Basemap(projection='merc', resolution='h',
```

```
                    lat_0=40, lon_0=0,
                    llcrnrlon=-9.5, llcrnrlat=35.5,
                    urcrnrlon=4.5, urcrnrlat=44.0)

        m.drawcountries()
        m.drawstates()
        m.drawcoastlines()

        m.drawmeridians(np.arange(-10, 10, 2.5), labels=[0, 0, 0, 1], fontsize=10)
        m.drawparallels(np.arange(35, 50, 2.5), labels=[1, 0, 0, 0], fontsize=10)

        ny = proj_data.shape[0]
        nx = proj_data.shape[1]
        lons, lats = m.makegrid(nx, ny)  # get lat/lons of ny by nx evenly space grid.
        x, y = m(lons, lats)  # compute map proj coordinates.

        m.contourf(x, y, proj_data, cmap=plt.get_cmap('jet'))
        m.colorbar(location='right', pad='5%')

        ts = nc.wmo_filename.split('_')[-1][:-3]
        output = ts + chn + '.png'
        plt.title(ts + "_" + chn)
        plt.savefig(output)

        plt.close()

    return output
```

Listing B.8: Program to plot on a map the different channels' data

```
import numpy as np
from netCDF4 import Dataset

#  Constants

no_data_value = 0

# Reflectance factor for visible bands
HRV_F    = 25.15
VIS006_F = 20.76
VIS008_F = 23.30
IR_016_F = 19.73

# Calibration coefficients from
# 'A Planned Change to the MSG Level 1.5 Image Product Radiance Definition'
# ,
# "Conversion from radiances to reflectances for SEVIRI warm channels"
# EUM/MET/TEN/12/0332
# , and
# "The Conversion from Effective Radiances to Equivalent Brightness
# Temperatures"
# EUM/MET/TEN/11/0569

CALIB = {}

# Meteosat 10

CALIB[323] = {'HRV': {'F': 78.9416 / np.pi},
              'VIS006': {'F': 65.5148 / np.pi},
              'VIS008': {'F': 73.1807 / np.pi},
              'IR_016': {'F': 62.0208 / np.pi},
              'IR_039': {'VC': 2547.771,
                         'ALPHA': 0.9915,
                         'BETA': 2.9002},
              'WV_062': {'VC': 1595.621,
                         'ALPHA': 0.9960,
                         'BETA': 2.0337},
              'WV_073': {'VC': 1360.337,
                         'ALPHA': 0.9991,
                         'BETA': 0.4340},
              'IR_087': {'VC': 1148.130,
                         'ALPHA': 0.9996,
                         'BETA': 0.1714},
```

```
                'IR_097': {'VC': 1034.715,
                           'ALPHA': 0.9999,
                           'BETA': 0.0527},
                'IR_108': {'VC': 929.842,
                           'ALPHA': 0.9983,
                           'BETA': 0.6084},
                'IR_120': {'VC': 838.659,
                           'ALPHA': 0.9988,
                           'BETA': 0.3882},
                'IR_134': {'VC': 750.653,
                           'ALPHA': 0.9982,
                           'BETA': 0.5390}}

# Polynomial coefficients for spectral-effective BT fits
BTFIT_A_IR_039 =  0.0
BTFIT_A_WV_062 =  0.00001805700
BTFIT_A_WV_073 =  0.00000231818
BTFIT_A_IR_087 = -0.00002332000
BTFIT_A_IR_097 = -0.00002055330
BTFIT_A_IR_108 = -0.00007392770
BTFIT_A_IR_120 = -0.00007009840
BTFIT_A_IR_134 = -0.00007293450

BTFIT_B_IR_039 =  1.011751900
BTFIT_B_WV_062 =  1.000255533
BTFIT_B_WV_073 =  1.000668281
BTFIT_B_IR_087 =  1.011803400
BTFIT_B_IR_097 =  1.009370670
BTFIT_B_IR_108 =  1.032889800
BTFIT_B_IR_120 =  1.031314600
BTFIT_B_IR_134 =  1.030424800

BTFIT_C_IR_039 =  -3.550400
BTFIT_C_WV_062 =  -1.790930
BTFIT_C_WV_073 =  -0.456166
BTFIT_C_IR_087 =  -1.507390
BTFIT_C_IR_097 =  -1.030600
BTFIT_C_IR_108 =  -3.296740
BTFIT_C_IR_120 =  -3.181090
BTFIT_C_IR_134 =  -2.645950


C1 = 1.19104273e-16
C2 = 0.0143877523

#  Class _Calibrator

eval_np = eval
log = np.log

class Calibrator(object):
    """
    Calibrator for Level 1.5 SEVIRI Images, the expected format for ‘hdr‘ is
    either a NetCDF4 Dataset or a HDF5 group, before METADATA or DATA
    bifurcation. On the other hand, ‘channel_name‘ is expected to be a string
    containing the name of the cannel, for instance 'VIS_006'.
    """

    def __init__(self, hdr, channel_name):
        self.hdr = hdr
        self.instance = type(hdr)
        self.channel_name = channel_name

    def __call__(self, calibrate='reflectances/bt', channel_name=None):
        """
        Computes the radiances and reflectances/bt of a given channel. The
        *calibrate* argument should be set to 'counts' for no calibration,
        'reflectances/bt' for default reflectances/bt calibration, and 'radiances'
        for returning radiances. The default value is 1.
        """
        hdr = self.hdr
        channel_name = self.channel_name if not channel_name else channel_name
        calibrate = {'counts': 0, 'reflectances/bt': 1, 'radiances': 2}[calibrate]
```

```
        if self.instance == Dataset:
            channels = {"VIS006": 'ch1',
                        "VIS008": 'ch2',
                        "IR_016": 'ch3',
                        "IR_039": 'ch4',
                        "WV_062": 'ch5',
                        "WV_073": 'ch6',
                        "IR_087": 'ch7',
                        "IR_097": 'ch8',
                        "IR_108": 'ch9',
                        "IR_120": 'ch10',
                        "IR_134": 'ch11'}
        else:
            channels = {"VIS006": 'Channel 01',
                        "VIS008": 'Channel 02',
                        "IR_016": 'Channel 03',
                        "IR_039": 'Channel 04',
                        "WV_062": 'Channel 05',
                        "WV_073": 'Channel 06',
                        "IR_087": 'Channel 07',
                        "IR_097": 'Channel 08',
                        "IR_108": 'Channel 09',
                        "IR_120": 'Channel 10',
                        "IR_134": 'Channel 11',
                        "HRV": 'Channel 12'}

        chn_nb = channels[channel_name]
        chn_index = list(channels.values()).index(chn_nb)
        planned_chan_processing = hdr['METADATA']['HEADER']['ImageDescription']['
            ImageDescription_DESCR'][26][1].split(',') \
                                  if self.instance is not Dataset \
                                  else hdr.variables['planned_chan_processing'][:]
        cal_type = [int(d) for d in planned_chan_processing]

        image = hdr['DATA'][chn_nb]['IMAGE_DATA'][:] if self.instance is not Dataset
            else hdr.variables[chn_nb][:]

        if calibrate == 0:
            return (image,
                    "counts")

        mask = (image == no_data_value)

        cal = hdr['METADATA']['HEADER']['RadiometricProcessing']['
            Level15ImageCalibration_ARRAY'] \
              if self.instance is not Dataset \
              else [[1, 0] for chn in channels.values()]
        cslope = cal[chn_index][0]
        coffset = cal[chn_index][1]

        radiances = eval_np('image * cslope + coffset')
        radiances[radiances < 0] = 0

        if calibrate == 2:
            return (np.ma.MaskedArray(radiances, mask=mask),
                    "mW m-2 sr-1 (cm-1)-1")

        sat = 323 # Meteosat 10 hdr["SatelliteDefinition"]["SatelliteId"]
        if sat not in CALIB:
            raise Exception("No calibration coefficients available for "
                                + "this satellite (" + str(sat) + ")")

        # Reflectance percentage
        if channel_name in ["HRV", "VIS006", "VIS008", "IR_016"]:
            solar_irradiance = CALIB[sat][channel_name]["F"]
            reflectance = eval_np('(radiances / solar_irradiance) * 100.')
            return (np.ma.MaskedArray(reflectance, mask=mask),
                    "%")

        # Brightness temperature
        chn_nb = chn_index
        wavenumber = CALIB[sat][channel_name]["VC"]
```

```
        if cal_type[chn_nb] == 2:
            #computation based on effective radiance
            alpha = CALIB[sat][channel_name]["ALPHA"]
            beta = CALIB[sat][channel_name]["BETA"]

            cal_data = eval_np('((C2 * 100. * wavenumber / '
                               'log(C1 * 1.0e6 * wavenumber ** 3 / '
                               '(1.0e-5 * radiances) + 1)) - beta) / alpha')

        elif cal_type[chn_nb] == 1:
            #computation based on spectral radiance
            cal_data = eval_np('C2 * 100. * wavenumber / '
                              'log(C1 * 1.0e6 * wavenumber ** 3 / '
                              '(1.0e-5 * radiances) + 1)')

            coef_a = eval("BTFIT_A_" + channel_name)
            coef_b = eval("BTFIT_B_" + channel_name)
            coef_c = eval("BTFIT_C_" + channel_name)

            cal_data = eval_np(('cal_data ** 2 * coef_a + '
                               'cal_data * coef_b + coef_c'))

        mask = mask | np.isnan(cal_data) | np.isinf(cal_data)
        cal_data = np.ma.MaskedArray(np.array(cal_data), mask=mask)
        return (cal_data,
                "K")
```

Listing B.9: Calibrator class designed to extract all the variables from the raw pixel count measured by the satellite

```
import pandas as pd
import numpy as np
from datetime import timedelta, datetime
from itertools import groupby
from functools import reduce
import os.path
import re
import sys

from Platform.platform_models.models import AreaGrid


def shape_data(data, lon_l, lon_r, lat_l, lat_r, variables, nsteps):
    idxlon = np.logical_and(data[:, 0] >= lon_l,
                            data[:, 0] <= lon_r)
    idxlat = np.logical_and(data[:, 1] >= lat_l,
                            data[:, 1] <= lat_r)
    idx = np.logical_and(idxlon, idxlat)
    data = data[idx]

    data = np.delete(data, np.s_[0:2], axis=1)
    data = data.reshape((data.shape[0], nsteps, variables))
    data = data.transpose((1, 0, 2))
    data = data.reshape((nsteps, -1))
    return data


def get_seviri_tags(calibration_modes, channels):
    tags = []
    for chn_name in sorted(channels.keys()):
        for cal_mode in calibration_modes:
            if cal_mode == 'reflectances/bt':
                if chn_name in ['VIS006', 'VIS008', 'IR_016']:
                    tags.append(chn_name + ' %')
                else:
                    tags.append(chn_name + ' K')
            else:
                tags.append(chn_name + ' R')
    return tags


def get_clmask_tags():
```

```
    return ['CLMASK']

class DataMatrix(object):
    '''
    Template for the dataMatrix model, the most important attribute is
    dataMatrix, which stores a pandas DataFrame containing the matrix
    itself. another attributes are:

      * config; it\'s only used to identify the farms and its predefined
          coords, until only square grids are used. will be deprecated.
      * date_format; format to translate the dates to.
      * path; the path to the directory containing all the myp files.
      * date; the date when the dataMatrix is created for.

    '''
    date_format = '%Y%m%d%H'

    channels = {"VIS006": 'ch1',
                "VIS008": 'ch2',
                "IR_016": 'ch3',
                "IR_039": 'ch4',
                "WV_062": 'ch5',
                "WV_073": 'ch6',
                "IR_087": 'ch7',
                "IR_097": 'ch8',
                "IR_108": 'ch9',
                "IR_120": 'ch10',
                "IR_134": 'ch11'}

    calibration_modes = ['radiances', 'reflectances/bt']

    def __init__(self, date, pathin, pathout, ncoord=2, delta=122,
                 grid=False, latlons=False, ifexists=False,
                 create=True, dm=False, suffix=""):
        '''
        Builds a dataMatrix object by reading up to DELTA MYP files stored
        in PATHIN. The PATHOUT argument indicates the path where the final
        dataMatrix may be stored.

        We can indicate a specific GRID for the data or a DataFrame for
        which to create the wrapper.
        '''
        self.path = pathin
        self.out = pathout
        self.file = self.out + date.strftime(self.date_format) + '.mdata' + suffix + '.
            npy'
        self.date = date

        self.tags_seviri = get_seviri_tags(self.calibration_modes,
                                           self.channels)
        self.tags_clmask = get_clmask_tags()
        self.tags = self.tags_seviri + self.tags_clmask

        if not grid:
            # Hard-code Iberian Peninsula corner coordinates
            self.grid = AreaGrid(44, -9.5, 35.5, 4.5, 0.125)
        else:
            self.grid = grid

        if not latlons:
            latlons = self.grid.get_lats_lons()

        self.cols_seviri = self.queryCols(latlons=latlons,
                                          tags=self.tags_seviri)
        self.cols_clmask = self.queryCols(latlons=latlons,
                                          tags=self.tags_clmask)

        if os.path.isfile(self.file) and ifexists:
            fechaini = (self.date - timedelta(days=delta))
            data = np.load(self.file)
            index = self.queryIndex(fechaini, date)
            cols = list(self.cols_seviri) + list(self.cols_clmask)
```

```
            self.dataMatrix = pd.DataFrame(data, index=index, columns=cols)
        elif create:
            self.dataMatrix = False
            self.generateMatrix(ncoord, delta)
        elif not create and dm is not False:
            self.dataMatrix = dm

    def generateMatrix(self, ncoord, delta):
        '''
        Generates the matrix itself, receives the same parameters as
        __init__, plus delta, which indicates how many days backwards
        we shall look to build the matrix. after reading the base variables,
        which are stored in the base files, it computes the modules
        of the velocity.

        This method doesn\'t return anything, but the resulting matrix
        is stored in the dataMatrix attribute of the object.
        '''
        lon_l, lon_r = self.grid.get_lons()
        lat_l, lat_r = self.grid.get_lats()
        nsteps = 1

        for date in pd.date_range(self.date - timedelta(days=delta),
                                  self.date, freq='H'):
            fecha = datetime.strftime(date, self.date_format)
            files = self.path + fecha + ".seviri.myp.npy"
            filec = self.path + fecha + ".clmask.myp.npy"

            if not os.path.isfile(files) or not os.path.isfile(filec):
                continue

            datas = np.load(files)
            datac = np.load(filec)

            datas = shape_data(datas, lon_l, lon_r, lat_l, lat_r, 22, nsteps)
            datac = shape_data(datac, lon_l, lon_r, lat_l, lat_r, 1, nsteps)

            index = self.queryIndex(date, date)

            df_seviri = pd.DataFrame(datas, index=index,
                                     columns=self.cols_seviri)
            df_clmask = pd.DataFrame(datac, index=index,
                                     columns=self.cols_clmask)
            df = pd.concat([df_seviri, df_clmask], axis=1)

            if isinstance(self.dataMatrix, bool):
                self.dataMatrix = df
            else:
                self.dataMatrix = pd.concat([self.dataMatrix,
                                             df], join='outer', axis=0)

    def dataMatrixFromDataFrame(self, df):
        '''
        Build a new DataMatrix copying the properties of self but with the
        inner DataFrame being replaced by df.
        '''
        return DataMatrix(self.date, self.path, self.out,
                          grid=self.grid, create=False, dm=df)

    def queryIndex(self, start_date, end_date):
        '''
        Generates a list of indexes to address the pandas DataFrame.

        This indexes are generated by checking the meteorology files
        we have inside the matrix. for each of this files, it calculates
        the timestamps by using the nsteps parameter, which indicates
        how many three-hourly intervals the file contains.
        '''
        index = []
        for date in pd.date_range(start_date, end_date, freq='H'):
            files = self.path + date.strftime(self.date_format) + '.seviri.myp.npy'
            filec = self.path + date.strftime(self.date_format) + '.clmask.myp.npy'
            if os.path.isfile(files) and os.path.isfile(filec):
```

```
189                 ts = date.strftime('%Y%m%d%H')
                    index.append(int(ts))
191         return index

193     def queryCols(self, grid=None, latlons=None, tags=[]):
            '''
195         Generates a list of the tags used to address each of the columns
            of the pandas DataFrame.
197
            Each tag is the string composition of the coordinates at the
199         point and the name of the variable it refers to.
            '''
201         if grid:
                lon_l, lon_r = grid.get_lons()
203             lat_l, lat_r = grid.get_lats()
                res = grid.get_res()
205             latlons = [(i, j) for i in np.arange(lat_l, lat_r+res, res)
                           for j in np.arange(lon_l, lon_r+res, res)]
207
            return np.array([['({0}, {1}) {2}'.format(j, i, tag)
209                           for tag in tags]
                          for i, j in latlons]).flatten()
211
        def querySubgrid(self, grid=None, latlons=None, inplace=False):
213         '''
            By giving a grid object, extracts the submatrix referring to it. If
215         grid is none, the complete matrix is returned.

217         If inplace is True, the inner dataMatrix is replaced by the
            resulting submatrix.
219         '''
            if grid or latlons:
221             result = self.dataMatrix.loc[:, self.queryCols(grid=grid,
                                                           latlons=latlons,
223                                                        tags=self.tags)]
            else:
225             result = self.dataMatrix

227         if inplace:
                self.dataMatrix = result
229         return result

231     def querySubgridDates(self, start_date, end_date,
                              grid=None, latlons=None, inplace=False):
233         '''
            This method returns a subset of the general matrix following the
235         above method, for a farm and dates between start_date and
            end_date.
237
            If inplace is True, the inner dataMatrix is replaced by the
239         resulting submatrix.
            '''
241         if grid or latlons:
                submatrix = self.querySubgrid(grid=grid, latlons=latlons)
243         else:
                submatrix = self.dataMatrix
245
            result = submatrix.loc[self.queryIndex(start_date, end_date), :]
247         if inplace:
                self.dataMatrix = result
249         return result

251     def querySubsetChannels(self, channels=channels, inplace=False):
            '''
253         Return a subset of the channels for the whole grid.

255         If inplace is True, the inner dataMatrix is replaced by the
            resulting submatrix.
257         '''
            result = self.dataMatrix.loc[:, self.queryCols(grid=self.grid,
259                                                        tags=channels)]
            if inplace:
261             self.dataMatrix = result
```

```
        return result

    def querySubsetChannelsGrid(self, grid=None, latlons=None,
                                channels=channels, inplace=False):
        '''
        Return a subset of the channels for a given grid. If the grid is
        None, the complete grid is taken.

        If inplace is True, the inner dataMatrix is replaced by the
        resulting submatrix.
        '''
        if grid is None and latlons is None:
            grid = self.grid

        result = self.dataMatrix.loc[:, self.queryCols(grid=grid,
                                                       latlons=latlons,
                                                       tags=channels)]
        if inplace:
            self.dataMatrix = result
        return result

    def querySubsetChannelsGridDateRange(self, start_date, end_date,
                                         grid=None, channels=channels,
                                         inplace=False):
        '''
        Return a subset of the total channels within a given date range
        and grid. If grid is None, the complete grid is taken.

        If inplace is True, the inner dataMatrix is replaced by the
        resulting submatrix.
        '''
        if grid is None:
            grid = self.grid

        result = self.dataMatrix.loc[self.queryIndex(start_date, end_date),
                                     self.queryCols(grid=grid, tags=self.tags)]
        if inplace:
            self.dataMatrix = result
        return result

    def groupByChannel(self, variable):
        '''
        Averages the value of channel (with calibration calib) on the
        whole grid.
        '''
        regexp = '(\w{6}[ ]?[A-Z%]?)'
        def keyf(col):
            return re.search(regexp, col).group(0)

        grid = {}
        for t in self.tags_seviri + self.tags_clmask:
            grid[t] = []

        cols = sorted(self.dataMatrix.columns)
        groups = map(lambda x: [x[0], list(x[1])[0]], groupby(cols, key=keyf))

        for k, v in groups:
            grid[k].append(v)

        return reduce(lambda acc, c: acc + self.dataMatrix[c].fillna(0),
                      grid[variable], 0) / len(grid[variable])

    def groupChannels(self):
        '''
        Return a DataFrame with all the aggregated values for each
        column in the dataMatrix.
        '''
        data = {var: self.groupByChannel(var) for var in self.tags}
        df_agg = pd.concat([data[k] for k in sorted(data.keys())], axis=1)
        df_agg.columns = sorted(data.keys())
        return df_agg

    def combineChannels(self, f, variables=None, init=0):
```

```
335         '''
            Combines the given variables by f with an initial value init.
337
            F should by a function taking an accumulated value and a
339         grouped DataFrame. Init is the initial value for this accumulation.

341         '''
            if variables is None:
343             return reduce(f, [self.groupByChannel(var) for var in self.tags],
                              init)
345         return reduce(f, [self.groupByChannel(var) for var in variables],
                          init)
347
    def saveMatrix(self, suffix=""):
349         '''
            Stores the matrix as a np binary format file, so we reduce
351         slightly the read times, but also has a counterpart, as long
            as the resulting file is quite big.
353
            This is intended to use within local environments with big
355         storage capacities, and mainly to run on benchmarking mode for
            a whole year or so, when a lot of dataMatrix are to be
357         created.
            '''
359         file = self.out + self.date.strftime(self.date_format) + '.mdata' + suffix
            np.save(file, self.dataMatrix.values)
```

Listing B.10: Software platform's core class, DataMatrix, which builds and operates on data matrices