A MATLAB program for PV module performance analysis based on real outdoor data stored in a PostgreSQL database

Hanne Kauko

Master's thesis

Master's Degree Programme in Renewable Energy

University of Jyväskylä, Department of Physics

Supervisors: Robert Kenny, Joint Research Centre

Jussi Maunuksela, University of Jyväskylä

2.12.2008

Preface

I spent a five-month period, from September 2007 to February 2008, as a trainee at the Solar Electricity Action (SOLAREC) of the Joint Research Centre (JRC) in Ispra, Italy. This thesis is a report of the project I conducted there: a MATLAB program for analyzing outdoor measurement data of photovoltaic (PV) modules stored in a PostgreSQL database. I found this traineeship extremely useful and highly educational. My experience in any kind of programming was quite limited upon arrival in Ispra; however, during my time there I developed my skills and managed to build up a fairly functional MATLAB program for diverse data analysis.

I would like to thank Robert Kenny, my supervisor at the JRC, who often spent several hours helping me solve problems I had – and never ran out of proposals for improvements to the program. If it is to be believed my trainee colleagues' stories about their supervisors, I was very fortunate. I would also like to acknowledge Thomas Huld (a researcher at the JRC) who was in charge of the interface that was required between MATLAB and the database and provided lots of computational support. Furthermore, I want to thank Zaira Girbau-Garcia (another researcher at the JRC), who followed on with my work and continued using the program after my period at the JRC had finished. This made me feel like I actually had created something useful.

Finally, I would like to acknowledge Jussi Maunuksela, my supervisor at the University of Jyväskylä. I am very grateful for his continual support in the writing process of this thesis and his encouragement in all stages of this process.

Abstract

The outdoor measurement field of the European Solar Test Installation (ESTI) at the Joint Research Centre (JRC) in Ispra, Italy, measures the performance of PV modules under real outdoor conditions over several months and years. This thesis discusses a project where a PostgreSQL database was introduced for appropriate storage for the vast amount of measurement data generated. To interface with the database and analyze the data, a MATLAB program was written. The analysis performed by the program involved primarily calculation of the daily and monthly energy yields and efficiencies of different modules, and comparison of these results with predicted results calculated from the basis of indoor generated models. This was related to development of an energy rating method at ESTI. In addition the program compared the performance of different modules. The results were presented both graphically and numerically in output text files.

The program has been constructed to be a flexible tool for analyzing and comparing the performance of different modules, and to provide valuable information about the accuracy of the energy rating method. Further developments to the program may include for instance improvements in its computational efficiency and making it more user-friendly.

Contents

1 Introduction

Energy from the sun is by far the largest energy source available on the earth: the amount of solar energy reaching the surface of the earth annually ($3.9 \cdot 10^{24}$ J) is about ten thousand times more than the global primary energy demand and more than all available energy sources on the earth [1, chapter 1]. Furthermore, it is inexhaustible and clean source of energy, which is essential in the globally prevailing circumstances: the fossil fuel resources are being rapidly depleted, population is increasing and environmental problems are growing ever more serious. One of the most potential and versatile methods for utilizing this vast energy source is its direct conversion into electrical energy with photovoltaic (PV) devices.

The first PV cells were produced in 1954 [1, chapter 4], and since then the technology has been improving, resulting in growth in the efficiency and reduction of costs. The efficiency of a typical flat-plate PV module, *i.e.*, the ability of a module to convert solar energy into electrical energy, is still relatively low: in laboratories efficiencies of up to 25% have been reached, but in commercial use the efficiencies reach only 14-17% [1, chapter 4]. Nevertheless, considering the enormous amount of energy coming from the Sun, the potential of PV technology in electricity production is still remarkable.

Cost has been the biggest barrier for rapid growth of PV markets so far as the most common material for PV modules is silicon, which is expensive to produce. On the other hand, market growth tends to bring the costs of a novel technology down; indeed the PV technology has recently been caught in a situation where it is not widely produced due to the high costs, and reduction of the costs is hindered by the low level of production [2]. However, improvements in both the module technology and the manufacturing process are continuously decreasing costs [3]. There are also new PV technologies, for instance thin films, with lower manufacuring costs awaiting wider adoption.

While new PV technologies are being developed, a reliable system for testing the technologies is fundamental. If a module is characterised by a reliable institution, it increases the credibility of the technology and thus promotes its acceptance to the markets. Furthermore, testing helps to ensure that the technologies that are accepted by the markets are superior and mature [2].

There are several laboratories dedicated to testing PV modules, and one example is the European Solar Test Installation (ESTI) at the Joint Research Centre (JRC) in Ispra, Italy. At ESTI, a range of measurements are performed on the modules, both indoors and outdoors. Under development is also an energy rating method, whose objective is to be able to predict the output of a module in a certain location by using only ambient temperature and irradiance data for the location. Energy rating would provide a valuable tool for ranking different module types and finding the most suitable module type for certain environmental conditions.

This thesis is related to a project where a data analysis system was created for outdoor measurement data of PV modules at ESTI. The system is comprised of a database that was introduced to provide an adequate storage for the vast amount of data received from the outdoor measurements, and a MATLAB program, written by the author, whose purpose was to calculate module outdoor energy yield and performance using data extracted from the database. The program was also to be used in testing the energy rating method of ESTI by comparing the outdoor measured energy yields with the predicted yields. Furthermore, energy yields of different modules were to be compared with each other.

The purpose of this thesis is to present the outdoor measurement system at ESTI, the database introduced for the data storage and most significantly the MATLAB program created for analyzing this data. The MATLAB program and the database system as well as the interface required between these two are discussed in chapter 5, and results obtained with the program are covered in chapter 6. The ESTI laboratory, the relevant measurements performed there and also the energy rating method developed at ESTI are introduced in chapter 4. Finally, chapters 2 and 3 provide the necessary background information about solar irradiance and the physics behind PV modules.

2 Solar radiation

It is difficult to understand the operation of an electricity production system based on photovoltaics without basic knowledge of its energy source, the radiation received from the sun. This chapter provides a brief introduction on this subject. Firstly, the sun itself is discussed: what it actually is, where does its energy originate and how can we approximate the amount of its radiation energy. Thereafter, the effect of the atmosphere on the radiation is considered in order to be able to estimate the amount of terrestrial solar radiation. Finally, the seasonal and daily motion of the sun, as we see it from the surface of the earth, is discussed in order to explain how the output of a solar collector can be maximized.

2.1 Energy from the sun

The sun is a vast and hot sphere of gas: the temperature in the interior of the sun is approximated to be around 15 million kelvins, its diameter is 1.4 million kilometers and its mass is $2.0 \cdot 10^{30}$ kilograms. Comparatively, for the earth these figures are 12800 kilometers and $6.0 \cdot 10^{24}$ kilograms, respectively. The sun consists mainly of helium and hydrogen; by mass approximately 80% is hydrogen and 20% is helium. [4, Chapter 7], [1, Chapter 2]

The energy of the sun emerges from fusion reactions where four hydrogen nuclei fuse into one helium nucleus in the hot interior of the sun. In this reaction, the mass of the reactants is more than the mass of the products, and thus energy is released. The energy moves by radiation and convection to the the surface of the sun, from which it radiates to the surrounding space. [1, Chapter 2]

To approximate the sun's radiation energy it is considered as a *blackbody.*

A blackbody is defined as a perfect emitter and a perfect absorber: it emits more energy per unit area than any real object, and absorbs all radiation incident on its surface, without favoring any particular frequencies. The temperature of a blackbody can be determined from its radiant power. The total intensity emitted by a blackbody, I (W/m^2), at absolute temperature T (K) is given by an emprirically determined law, Stefan-Boltzmann's law [5]:

$$I = \sigma T^4, \tag{1}$$

where σ is the Stefan-Boltzmann constant ($5.67 \cdot 10^{-8}$ W/m^2 K^4). The surface temperature of the Sun is estimated to be 5800 K. [5, Chapter 38], [4, Chapter 7]

The spectral distribution of a blackbody at a certain temperature is given by Planck's law, which was derived after the Stefan-Boltzmann's law from the basis of empirical results. The Planck's law is [5]

$$I(\lambda) = \frac{2\pi hc^2}{\lambda^5 \left(\exp(hc/\lambda kT) - 1\right)} \tag{2}$$

where $I(\lambda)$ is the spectral emittance, *i.e.*, intensity per wavelength interval, of a blackbody (W/m^3), h is the Planck's constant ($6.626 \cdot 10^{-34}$ Js), c is the speed of light ($2.998 \cdot 10^8$ m/s), k is the Boltzmann constant ($1.381 \cdot 10^{-23}$ J/K) and λ is the wavelength (m). The spectral distribution of a blackbody at 5800 K approximates very well the actual measured spectrum of the sun, as figure 1 illustrates. [5, Chapter 38], [4, Chapter 7]

2.2 Solar radiation on the earth's surface

As the solar radiation reaches the atmosphere of the earth, it is scattered, reflected and absorbed by atmospheric particles. As a result, only a portion of the solar radiation outside the earth's atmosphere, *i.e.*, the extraterrestrial

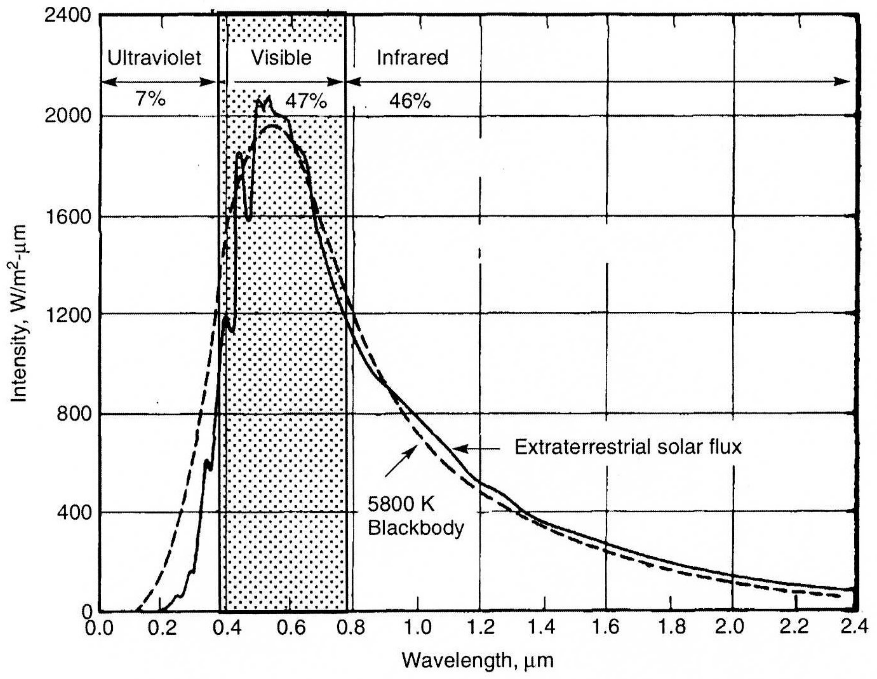


Figure 1: The extraterrestrial spectrum and the spectrum of a blackbody at 5800 K. Image reproduced from [4] (fig. 7.2).

radiation, actually reaches the surface of the earth. This portion, the terrestrial radiation, varies from less than 50% to 70%, depending on the position of the sun and the clearness of the sky. In this section the extraterrestrial radiation is first defined, and then the terrestrial radiation, consisting of three components, is discussed. [4, Chapter 7]

2.2.1 Extraterrestrial radiation

The extraterrestrial radiation is defined as the radiation that passes perpendicularly through an imaginary surface just outside the earth's atmosphere. It varies from day to day, depending on the distance between the sun and the earth. The extraterrestrial radiation, I_0 (W/m^2), on each day of the year is given by [4]

$$I_0 = SC \cdot \left[1 + 0.034\cos\left(\frac{2\pi}{365}n\right)\right], \tag{3}$$

where SC is the solar constant and n is the day number (starting from the 1st of January). The solar constant is an estimate of the average annual extraterrestrial radiation, having a generally accepted numerical value of 1377 W/m^2. [4, Chapter 7]

2.2.2 Components of the terrestrial radiation

The total terrestrial solar flux hitting for instance the surface of a solar collector consists of three components: direct-beam, diffuse and reflected radiation. Direct beam radiation is the radiation that passes in a straight line through the atmosphere; diffuse radiation is radiation scattered by atmospheric particles or moisture or reflected from clouds; and reflected radiation is the radiation reflected from surfaces in front of the collector. [4, Chapter 7]

The amount of direct-beam radiation depends on how much attenuation occurs on its way through the atmosphere. The direct-beam intensity is defined in a plane perpendicular to the beam, but in the general case the effective intensity will reduce with increasing angle of incidence (see figure 4). The amount of attenuation depends primarily on the length of the path the sun's rays have travelled through the atmosphere. It is important to note that the attenuation caused by the atmosphere does not occur for all wavelengths equally, so the spectral distribution at the earth's surface is not the same as that of the extraterrstrial radiation, as it can be seen in figure 3. For this reason, the spectral distribution will also change during the day and depending on atmospheric conditions. The total radiation is the integral of the intensities at all the individual wavelengths. A quantity describing the path length of the sun's rays through the atmosphere, and thus the solar spectrum when the sun is at a certain altitude, is called the *air mass ratio.* It is the ratio of the length of the actual path taken by the rays h_2 and the minimum path length h_1, *i.e.*, the path length when the sun is directly overhead. The air mass ratio can be expressed as [4]

$$\texttt{Air mass ratio } m = \frac{h_2}{h_1} = \frac{1}{\sin\beta}, \tag{4}$$

where β is the altitude angle of the sun, ilustrated in figure 2. The air mass ratio is equal to unity when the sun is directly overhead; that case is designated as AM1 spectrum. The extraterrestrial spectrum is referred to as AM0 spectrum. In the field of photovoltaics, AM1.5, *i.e.*, the spectrum when the altitude angle of the sun is 41.8°, is often used as an estimate of the average spectrum on the surface of the earth. The AM0 and AM1.5 spectra are compared in figure 3. [4, Chapter 7]

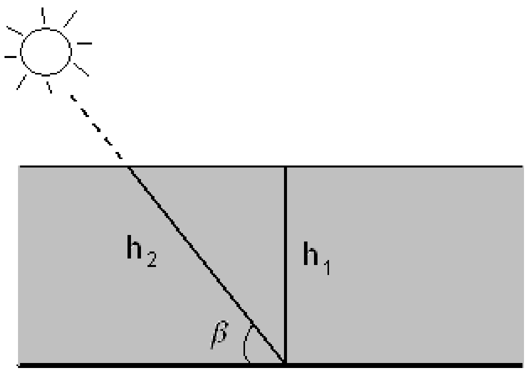


Figure 2: The air mass ratio. h_1 is the minimum path length for the sun's rays through the atmosphere and h_2 is the length of the actual path, and β is the altitude angle of the sun.

Finally, the amount of direct beam radiation can be calculated. It is an exponential function of the attenuation parameters [4]:

$$I_B = Ae^{-km}, \tag{5}$$

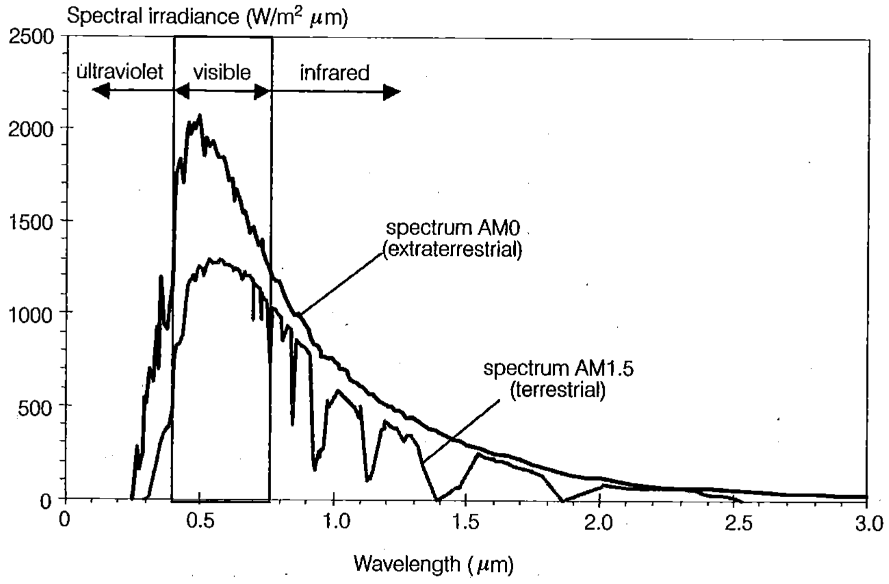


Figure 3: The extraterrestrial spectrum and the terrestrial AM1.5 spectrum. Image reproduced from [1] (fig. 2.3).

where I_B is the direct beam radiation on a surface normal to the rays (W/m^2), A is an "apparent" extraterrestrial flux (W/m^2), k is a dimensioless factor known as the optical depth, and m is the air mass ratio. The values of A and k depend on the time of the year, and usually tabulated values are used for these parameters. When calculating the amount of direct-beam radiation on an actual surface, the angle of incidence has to be taken into account as well. As figure 4 shows, the angle of incidence θ is the angle between the incoming rays and a line normal to the collector. It is a function of the tilt angle of the collector, and the altitude and azimuth angles of the sun (see section 2.3). The direct-beam radiation on a tilted collector, I_{BC} (W/m^2), is calculated as follows [4]:

$$I_{BC} = I_B \cos(\theta). \tag{6}$$

For the special case of horizontal surface $\theta = 90^\circ - \beta$, where β is the altitude angle of the sun. The expression (6) then becomes [4]

$$I_{BH} = I_B \cos(90^\circ - \beta) = I_B \sin(\beta), \tag{7}$$

where I_{BH} is the direct-beam radiation on a horizontal surface (W/m^2). [4, Chapter 7]

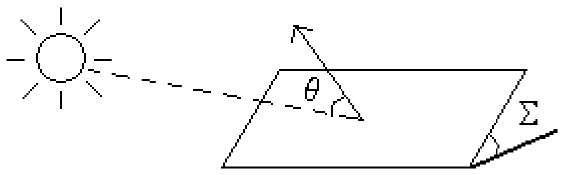


Figure 4: The angle of incidence θ on a collector tilted with an angle Σ.

The amount of diffuse radiation can be measured with a radiation sensor simply by blocking out the direct beam radiation with a shading disc of the apparent size of the sun. It is more difficult to estimate a theorethical value for the diffuse radiation. Usually the estimates are based on the assumption that the sky is isotropic, *i.e.*, the diffuse radiation arrives with equal intensity from all directions no matter where in the sky the sun happens to be, as figure 5 illustrates. Thus the diffuse radiation on a horizontal surface, I_{DH} (W/m^2), is a simple function of direct-beam radiation [4]:

$$I_{DH} = CI_B \tag{8}$$

where C is the sky diffuse factor, dependent on the time of the year. Its values can be either calculated or taken from a table. To calculate the amount of diffuse radiation on a tilted collector – from the basis of the same assumption of isotropic sky – one has to consider, how big portion of the sky the collector "sees": a horizontal collector sees the whole sky, and a vertical one sees half of it. The diffuse radiation on a tilted collector, I_{DC} (W/m^2), is then calculated as follows [4]:

$$I_{DC} = I_{DH}\frac{1+\cos(\Sigma)}{2} = CI_B\frac{1+\cos(\Sigma)}{2}, \tag{9}$$

where Σ is the tilt angle of the collector. [4, Chapter 7]

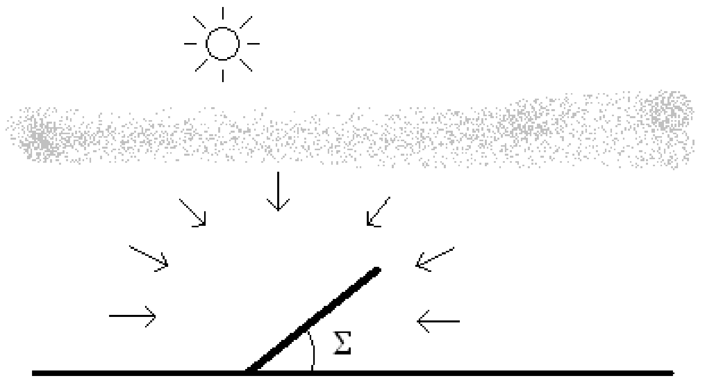


Figure 5: Diffuse radiation on a collector tilted with an angle Σ.

To estimate the amount of the final component, reflected radiation, broad assumptions are needed again. The simplest models assume a large horizontal surface in front of the solar collector, having a uniform reflectance ρ and reflecting the radiation with equal intensity to all directions. The value for ρ varies from 0.8 for fresh snow to 0.1 for a black, gravel roof. The reflecting surface obviously reflects all the incident radiation – both direct-beam and diffuse; thus the total reflected radiation is the sum of these two components multiplied with ρ. To estimate the amount of reflected radiation hitting a tilted collector, it again has to be considered how much of the reflected radiation the collector receives: a horizontal surface clearly receives none of this radiation, and a vertical surface receives half of it. Using the earlier expressions (8) and (7), the expression for reflected radiation on a tilted collector, I_{RC} (W/m^2), becomes [4]

$$I_{RC} = \rho(I_{BH} + I_{DH})\frac{1-\cos(\Sigma)}{2} = \rho I_B[\sin(\beta) + C]\frac{1+\cos(\Sigma)}{2}. \qquad (10)$$

In some special occasions – for instance when the collector is surrounded by snow – the reflected radiation can boost the collector's performance significantly. In most cases, however, its portion is so small it can be completely neglected.[Chapter 7] [4]

To summarize, the total incident radiation on a solar collector tilted with an angle Σ is the sum of these three components: direct-beam, diffuse and reflected radiation (I_{BC}, I_{DC} and I_{RC}, respectively). When calculating radiation on a collector in an arbitrary location, all of them have to be taken into account.

2.3 The apparent motion of the sun

Resulting from the motion of the earth around the sun and its own axis, the position of the sun in the sky varies both daily and annually, as seen from the perspective of any point on the earth's surface. As figure 6 illustrates, the earth travels in space on an elliptical orbit around the sun, with its axis tilted – which furthemore causes the seasonal variations in the sun's path across the sky. Simultaneously the earth makes 24-hour full circles around its axis[1], causing the sun to rise and set every day. The apparent motion of the sun complicates the work of people in the field solar energy, as maximizing the energy production from a solar collector would require at least partial tracking of the sun. To get familiar with the subject, this section introduces first the seasonal variation of the altitude angle of the sun at solar noon, and then the daily variations in the sun's position.

[1]Actually, due to its elliptical orbit around the sun, the earth has to rotate $360,99°$ around its axis each day in order to stay synchronized [4, chapter 7].

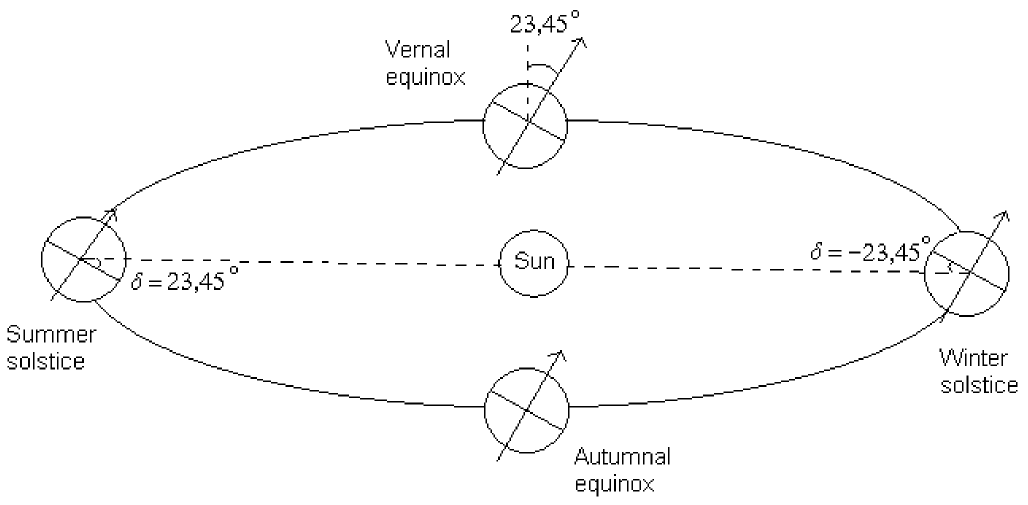


Figure 6: The earth's elliptical path around the sun. The site of the earth at solstices and equinoxes, as well as the solar declination δ at both solstices are also indicated.

2.3.1 Seasonal variations in the altitude of the sun at solar noon

Solar noon at any location is the time of the day when the sun is at its highest point. At that time sun shines (in the Northern Hemisphere) directly from south, *i.e.*, the sun is directly above the local meridian. The altitude angle of the sun at solar noon relative to the local horizontal depends on the solar declination at the time of the year and the latitude of the site. [4, Chapter 7]

Solar declination is defined as the angle between the plane of the equator and a line drawn from the centre of the sun to the centre of the earth (see figure 6). The earth's axis is tilted with an angle of 23.45° – thus the solar declination varies between the extremes of ±23.45° as the earth revolves around the sun. As illustrated in figure 6, the positive extreme is reached at summer solstice (approximately June 21) and the negative one at winter solstice (approximately December 21). At the equinoxes the solar declination is zero. Assuming the vernal equinox to be on day $n = 81$ (May 20), the value for solar declination on the nth day of the year can be estimated with a simple sinusoidal relationship [4]:

$$\delta = 23.45^\circ \sin\left[\frac{360^\circ}{365}(n-81)\right]. \tag{11}$$

The altitude angle of the sun at solar noon, β_N, can now be determined using the solar declination. According to figure 7, β_N at a latitude L is [4]

$$\beta_N = 90^\circ - L + \delta. \tag{12}$$

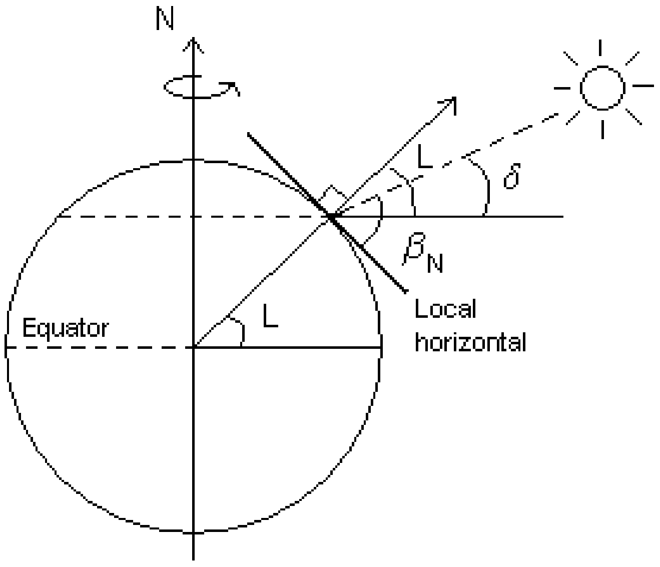


Figure 7: The altitude angle of the sun at solar noon, β_N, at latitude L and with solar declination δ.

Knowing these angles it is easy to estimate the best possible tilt angle for a solar collector – also the angle between the collector and the local horizontal. Firstly, facing the collector towards equator (due south in the Northern Hemisphere) is a good starting point, as the sun always reaches its highest point in the south of each location (in the north in the Southern Hemisphere). A suitable tilt angle would then be the angle at which the collector would be perpendicular to the sun's rays at solar noon. At equinoxes the solar declination is zero, thus tilting the collector so that it is parallel to the earth's rotation axis (see figure 7), *i.e.*, tilting the collector with an angle equal to the latitude of the location, would be reasonable, at least around the equinoxes. This is indeed a good angle to start with; though in the summer a bit smaller, and in the winter slightly bigger angles are more optimal. [4, Chapter 7]

2.3.2 Daily variations in the position of the sun

The position of the sun on an arbitrary day, at an arbitrary moment and at an arbitrary location is described with two angles, azimuth angle ϕ and altitude angle β, illustrated in figure 8. The azimuth angle is measured respective to the solar noon, and its values are by convention positive before noon, when the sun shines from the east, and negative after noon, when the sun shines from the west (see figure 8). The altitude and azimuth angles can be calculated using the following equations [4]:

$$\sin\beta = \cos L \cos\delta \cos H + \sin L \sin\delta \tag{13}$$

$$\sin\phi = \frac{\cos\delta \sin H}{\cos\beta}, \tag{14}$$

where H is the hour angle, that is the angle that the sun has to rotate before it is directly over the local meridian. The values for the hour angle are also positive in the morning, before the sun has crossed the local meridian, and negative in the afternoon. The earth rotates 360° in 24 hours, *i.e.*, 15° per hour; thus the values for the hour angle are +15° at 11.00 a.m. and −15° at 1.00 p.m. etc. [4, Chapter 7]

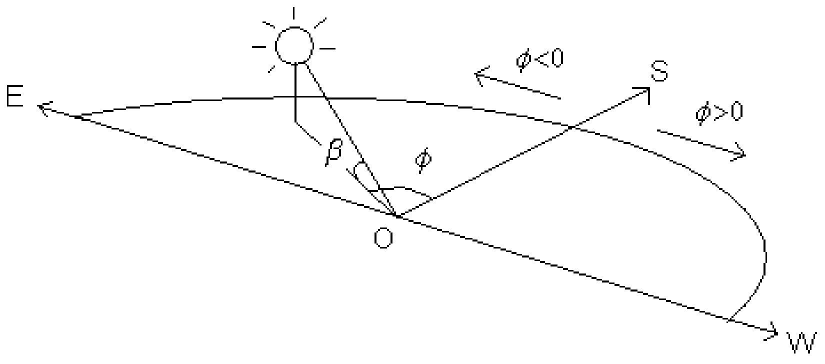


Figure 8: The altitude and azimuth angles of the sun, β and ϕ, respectively. O stands for observer.

A solar collector can be either fixed, or it can track the sun partially or entirely: it can be rotated either on one or two axes along the motion of the sun. Tracking increases the energy production of a solar collector significantly, but on the other hand a tracking system is complicated and expensive to install and maintain. Furthermore, electricity is required to run the trackers, which reduces the overall energy production of the system. In the case of flat-plate solar collectors, tracking has shown to have significant economical benefits only in large systems, in regions with high (especially high direct beam) annual radiation. For concentrating solar systems, however, the situation is quite specific – for those systems tracking is necessary. [1, Chapter 2]

3 The physics behind PV modules

Fundamentally, the operation of photovoltaic (PV) cells is based on the photoelectric effect. The photoelectric effect is the emission of electrons from the surface of a material (usually a metal) when light strikes the surface, and it was first discovered in 1887 by Heinrich Hertz. A correct explanation for the effect was not found until in 1905, when Albert Einstein postulated that light consists of quanta of light, called photons, and a photon with enough energy can give an electron energy sufficient to escape from the potential field of its nucleus. The number of escaped electrons depends on the number of photons, *i.e.*, the intensity of light, and energy of the electrons depends on the energy of the photons. [5, Chapter 38]

To exploit the photoelectric effect for electricity production we need a voltage difference to drive the liberated electrons – and this is where solid-state technology enters the picture. The means for moving the mobile electrons in the desired manner are provided by a *p-n junction diode*, that consists of two types of semiconductors. Therefore, the first actual photovoltaic solar cell was not built until in 1954 [1, Chapter 4], when the required solid-state technology was available.

In this chapter the discussion starts from the properties of semiconductors, from the basis of which we can determine the maximum possible amount of the sun's energy that can be captured by a certain semiconductor. Thereafter the p-n junction is explained, and from the p-n junction we proceed to PV cells and further to PV modules and arrays. Some attention is also paid to the impacts of temperature and irradiance on the performance of a PV module. The performance measurement of a PV module is then discussed, as in this thesis the measurements on PV modules are in the spotlight. Finally, some of the various PV module technologies are introduced.

3.1 Semiconductor physics

Semiconductors are characterized by their conductivity properties, that lie between those of conductors and insulators. The most common semiconductor is silicon, that is a group IV element in the periodic table of elements (see figure 9). This means that it has four electrons in its outermost shell, *i.e.*, it has four valence electrons, which determines its electrical properties. Another common semiconductor is germanium, that is also a group IV element. Compounds of elements for instance from groups III and V (gallium and arsenic), II and VI (cadmium and tellurium) and even I, III and VI (copper, inidium and selenide) are also used to produce semiconductors. Here semiconductor properties are discussed using the example of silicon. [4, Chapter 8]

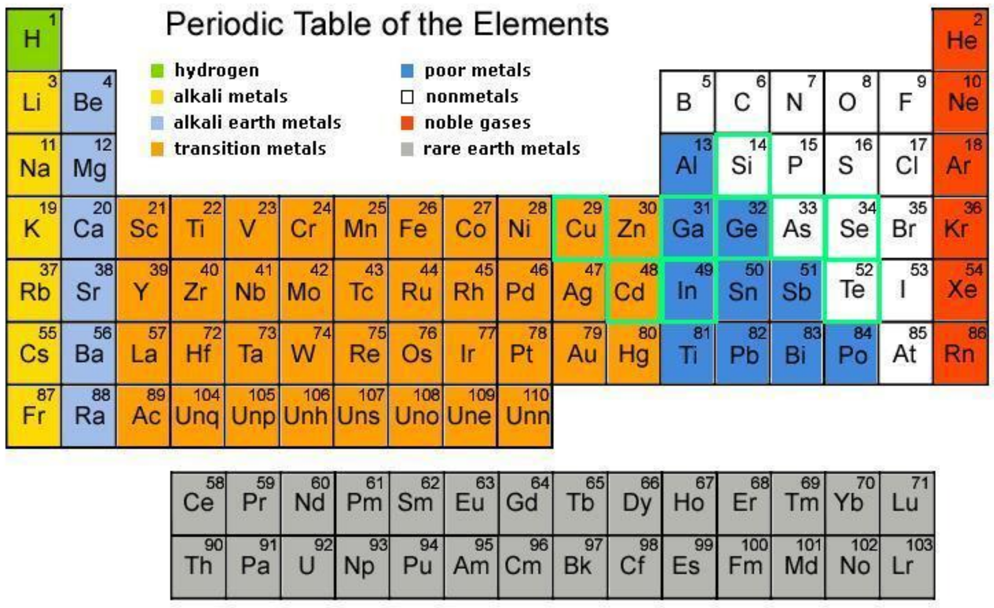


Figure 9: The periodic table of the elements. Semiconductors, and elements used as compounds to produce semiconductors, are highlighted with green. Image reproduced from [6].

In pure silicon crystal, the four valence electrons of a silicon atom are tied with strong covalent bonds to four adjacent silicon atoms. Therefore at zero temperature silicon is a perfect insulator: there are no free electrons to carry currents as there are in metals, but all the electrons are tied to their nuclei. As the temperature increases, some electrons gain enough energy to escape from the potential field of their nuclei and thus the conductivity of silicon increases. The conductivity properties of different materials can be described in terms of allowed energy levels and forbidden gaps between them; this subject is covered in section 3.1.1. [4, Chapter 8], [5, Chapter 42]

Semiconductors are very sensitive to impurities. This property can be exploited in changing their conductivity in a more favorable direction by adding suitable impurities, which is called *doping*. Semiconductors can be doped so that there is *excess* or *shortage* of electrons, to make *n-* or *p-type* semiconductors, respectively. When p- and n-type material are brought into contact, a *p-n junction* is formed. This junction has many favorable properties and is used widely in electronics, including photovoltaics. Doping and the p-n junction are discussed in more detail in sections 3.1.2 and 3.1.3. [4, Chapter 8], [5, Chapter 42]

3.1.1 Band structure and band gap energy

According to quantum mechanics, electrons in atoms have well-defined possible, discrete energy levels. As several atoms are brought into contact these levels spread out into so called bands. Depending on the distance of the atoms and the bonds between them, there might be bands of forbidden energy called band gaps between the atoms, or the bands may overlap, forming a continuum of allowed energy states in the material. The characteristics of insulators, conductors and semiconductors depend on their band structure. [7]

The bands of interest are the *valence band*, that is the highest completely filled band, and the *conduction band*. In the valence band all the energy states are occupied and hence its electrons are immobile, whereas in the conduction band there are plenty of unoccupied states for electrons to move in response to an applied electric field. To get to the conduction band, however, an electron has to jump over a gap – the forbidden band between the valence and conduction bands. Hence the conductivity properties of a material are dependent on the size of this gap, called the *band gap energy*. In conductors the valence and conduction bands overlap; thus there is lots of mobile electrons to carry a current already at lower temperatures. In insulators, the band gap energy at room temperature can be 5 eV or more, and in semiconductors it is around 1 eV (at room temperature). For instance silicon has a band gap energy of 1.12 eV. [5, Chapter 42]

In semiconductors not only the electrons in the conduction band can move and carry currents, but also the vacancies they leave to the valence band, called *holes*. When an electron is excited to the conduction band, a vacant energy state, a hole, is generated to the valence band. Another electron can move into this hole, and further a third electron may move to the vacant state of the previous electron and so on. This apparent motion of holes in the valence band contributes to the current like the motion of electrons in the conduction band. When an electric field is applied, the holes in the valence band move to opposite direction with respect to the electrons in the conduction band, although the moving charge carriers are actually electrons in both bands. This feature, observed in pure semiconductors, is known as *intrinsic conductivity*. [4, Chapter 8], [5, Chapter 42]

Then it must be considered how an electron can obtain the required energy to jump to the conduction band. Thermally is obviously one way, but in photovoltaics the energy is received from the photons of solar radiation. To excite an electron for instance in a silicon crystal, a photon with an energy of 1.12 eV is required. The energy of a photon is related to its frequency with the following expression [4]:

$$E = h\nu = h\frac{c}{\lambda}, \tag{15}$$

where h is the Planck's constant ($6.626 \cdot 10^{-34}$ Js), ν is the photon's frequency (Hz), λ is its wavelength (m) and c is the speed of light ($2.998 \cdot 10^{8}$ m/s). The speed of light is related to the frequency and wavelength with the expression $c = \nu\lambda$. [4, Chapter 8]

Using equation 15 it can be calculated that in silicon photons with wavelengths shorter than 1.11 µm are able to excite an electron to the conduction band. Photons with wavelengths longer than 1.11 µm cannot do this, but their energy is wasted as heat. On the other hand, as only the exact amount of 1.12 eV is utilized by the excited electron, photons with wavelengths shorter than the limit have excess energy that also heats the cell. This means that in the case of silicon 20.2% of the sun's energy is wasted due to photons with too long, and 30.2% due to photons with too short wavelengths, giving a theorethical upper limit of 49.6% for the efficiency of a single junction silicon solar cell (see figure 10). In real silicon solar cells, however, the highest efficiencies that have been obtained in laboratories are in the order of 25%. The remaining 20% is lost due to various reasons, such as [4, Chapter 8]:

- some of the photons are reflected from the surface of the cell and some pass right through the cell
- part of the generated electron-hole pairs are recombined before they contribute to the current

- the cell has some internal resistance.

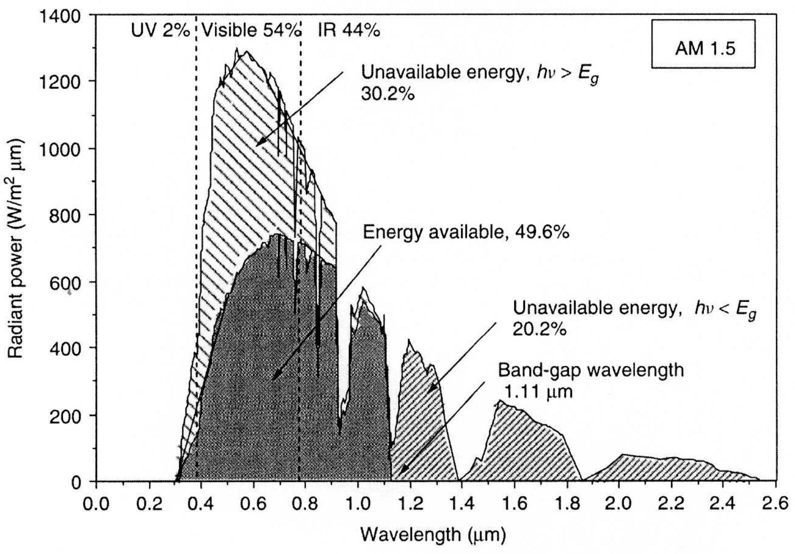


Figure 10: The solar spectrum at AM1.5. Indicated are the portion of the sun's energy that can be exploited by a silicon solar cell (dark area), and the portions of energy wasted for photons with excess energy (sparse stripes) and for photons with shortage of energy (dense stripes). Image reproduced from [4] (fig. 8.10).

Obviously, it is the size of the band gap of the material used that determines the theorethical upper limit for the efficiency of a solar cell. With lower band gap energy, there are more photons with the ability to excite electrons to the conduction band, resulting in a higher current; on the other hand, there are also more photons with excess energy that is wasted as heat. With higher band gap energy in turn, less electrons are excited, but the electrons have more energy and there are also less photons with excess energy to be dissipated. Thus, a smaller band gap yields more current and less voltage, and a higher band gap gives the opposite. The optimum band gap, that results in the highest possible power and efficiency, is estimated to be between

1.2 eV and 1.8 eV [4, Chapter 8] – the band gap of silicon is thus slightly too small. When new photovoltaic materials are developed, the size of the band gap is one of the primary concerns, as it will be seen in section 3.3.

3.1.2 Doping

As it was explained earlier, semiconductors are doped to improve their conductivity and to get the required ingredients for the p-n junction: the p- and n-type semiconductors. N-type silicon is produced by introducing a small portion of some group V element, typically phosphorus, into the silicon crystal. Typically a ratio of approximately one phosphorus atom per 1000 silicon atoms is used – already this is sufficient to change the conductivity properties of silicon significantly. A phosphorus atom takes place of a silicon atom in the crystal lattice, and out of the five valence electrons of phosphorus, four are tied with covalent bonds to the adjacent silicon atoms. The fifth electron, however, is very loosely bound, and requires very little energy to be excited to the conduction band; at room temperature the fifth electron is most probably found in the conduction band. What the fifth electron then leaves behind is a $+15e$ phosphorus nucleus surrounded by 14 electrons, *i.e.*, an ion with a net charge of $+e$. This ion is fixed in the crystal lattice – hence there is a fixed net positive charge and a free electron towards each ion. As group V elements donate electrons, they are called donors. This type of semiconductor is called an n-type semiconductor because of the mobile negative charge carriers. [4, Chapter 8], [5, Chapter 42]

To produce p-type semiconductor, group III elements are introduced to the semiconductor. Silicon is typically doped with boron, with approximate concentrations of one boron atom per ten million silicon atoms. Again, each boron atom substitutes a silicon atom in the silicon crystal, and is surrounded by four silicon atoms. Boron has three valence electrons that are all bound to the adjacent silicon atoms, but now an extra hole, a vacant energy state, is left next to the boron atom. This hole is easily filled by electrons from nearby atoms, and can therefore be thought as a mobile positive charge. As the hole is filled, the boron atom having a $+5e$ charge in its nucleus is surrounded with alltogether six electrons – thus a fixed ion with net charge of $-e$ is formed. As boron atoms accept electrons, they are called acceptors. A semiconductor doped with an acceptor is called p-type semiconductor because of its free positive charge carriers. [4, Chapter 8], [5, Chapter 42]

It is important to remember that despite their names, p- and n-type semiconductors are electrically neutral. The names merely refer to the type of majority charge carriers in these materials – electrons in n-type and holes in p-type semiconductors.

3.1.3 The p-n junction

When p- and n-type semiconductors are brought into contact, in the vicinity of the junction electrons from the n-side diffuse to the p-side and combine with the holes there. Doing so, electrons create immobile negative ions to the p-side and leave immobile positive ions behind in the n-side. This gives rise to an electric field that is directed from n-side to p-side and thus opposes the diffusion of electrons. Finally an equilibrium is obtained, and no diffusion of electrons occurs any more. Consequently, the p-n junction diode is divided into two regions: a depletion region and the quasi-neutral regions. As illustrated in figure 11, depletion region encompasses the region in the immediate vicinity of the junction, which is – due to the diffusion described above – depleted from charge carriers. Quasi-neutral regions in turn cover the regions "far" from the junction on both sides. In these regions the space charge density is assumed zero (hence the name quasi-neutral region) since no electrons have diffused from or to these regions to create positive or negative ions. [7, 8]

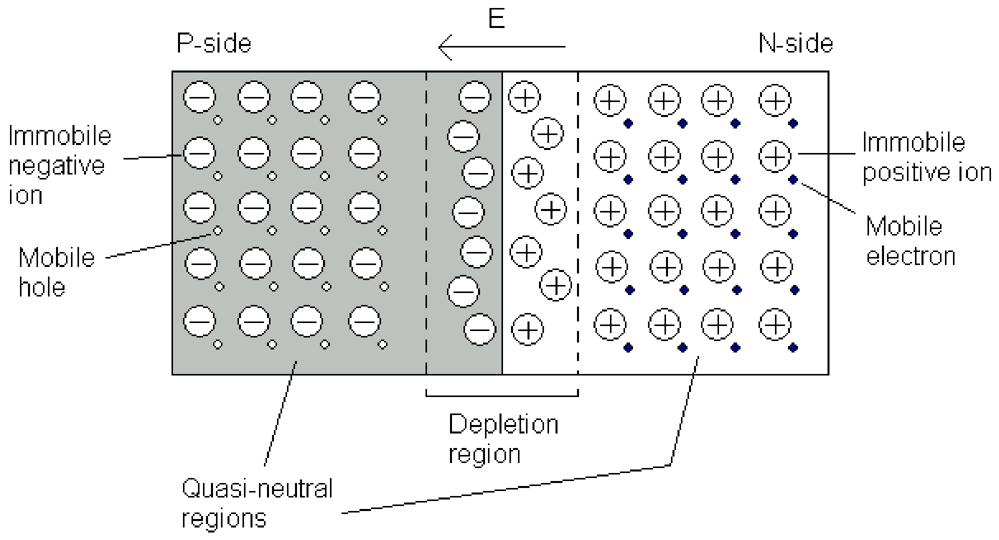


Figure 11: The p-n junction.

What is created by bringing together p- and n-type materials as described above is called a *p-n junction diode*[2]. The current-voltage characteristics of a diode are given by the Shockley diode equation [4]

$$I_d = I_s(e^{qV_d/kT} - 1), \tag{16}$$

where I_d is the current in the diode (A), I_s is a small reverse saturation current (A), q is the electron charge ($1.602 \cdot 10^{-19}$ C), V_d is the voltage across the terminals of the diode (V), k is the Boltzmann constant ($1.381 \cdot 10^{-23}$ J/K),

[2]In practice the p- and n-type materials cannot be coupled just by sticking them together, but usually the junction is produced by depositing some n-type material on the surface of a p-type material [5, Chapter 42].

and T is the junction temperature (K). The current-voltage relationship of a diode is displayed in figure 12. [4, Chapter 8]

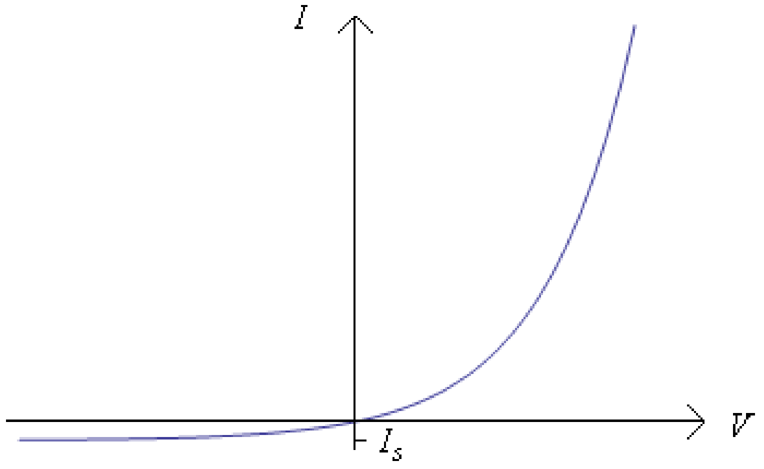


Figure 12: The current-voltage relationship of a p-n junction diode according to the Shockley diode equation (16). When the voltage is negative (reverse bias), the current tends to the saturation current, and when the voltage is positive (forward bias), the current increases rapidly.

When the voltage V_d applied across the diode is positive, *i.e.*, the applied electric field is directed from p-side to n-side, the diode is said to be forward biased. In this case the electric field of the depletion region is attenuated and therefore current flows readily across the junction: holes flow from p-side to n-side, and electrons do vice versa. The equivalent circuit for a forward-biased diode is shown in figure 3.1.3. When V_d is negative, the diode is reverse biased, and the electric field of the depletion region is strengthened. This field attempts to push electrons from p-side to n-side, and holes from n-side to p-side. There are, however, very few free electrons in the p-side and free holes in the n-side, and thus only the small saturation current will flow across the diode. The saturation current results from diffusion of so-called minority charge-carriers, which stand for holes in the n-side and electrons in the p-side. [5, Chapter 42]

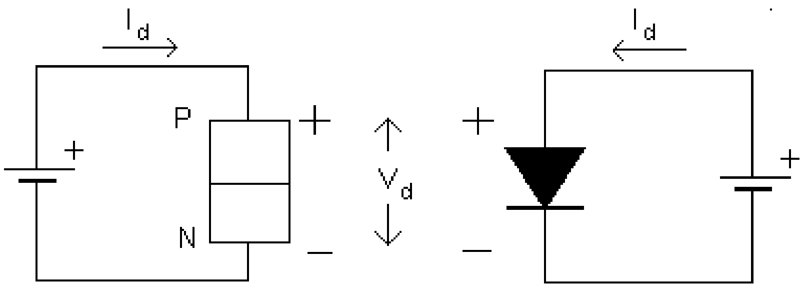


Figure 13: The equivalent circuit for a p-n junction diode in forward bias. Both the official symbol (on the right) and a schematic figure of the diode are shown.

3.2 From p-n junction to PV cells, modules and arrays

3.2.1 Electrical properties of a PV cell

When the p-n diode is exposed to light, electron-hole pairs are generated in silicon atoms in both sides of the diode, *i.e.*, electrons are excited from the valence band to the conduction band, leaving holes behind. If the pairs are generated in or reach the vicinity of the junction, they are exposed to the electric field of the depletion region, and electrons are pulled into the n-side and holes into p-side. As the charges cumulate on both sides of the diode, a voltage is generated to carry a current. If the different sides of the junction are electrically connected, electrons will flow from the n-side to the p-side, where they recombine with holes. [4, Chapter 8]

The electrical properties of a solar cell formed like this can be thought as those of a p-n junction diode in parallel with a current source, that delivers current proportionally to the incident solar flux. The equivalent circuit for a solar cell is shown in figure 14. To determine the current-voltage characteristics of a solar cell it is first necessary to introduce two parameters important in photovoltaics: short-circuit current I_{SC} and open-circuit voltage V_{OC}. Short-circuit current is the current delivered when the leads are shorted together. In this situation the voltage across the diode is zero, and thus no current flows through the diode but all the current flows through the shorted leads. Ideally, the short-circuit current is equal to the light-generated current. The open-circuit voltage in turn is the voltage when the leads are not connected, and in this situation the current is clearly zero. [4, Chapter 8]

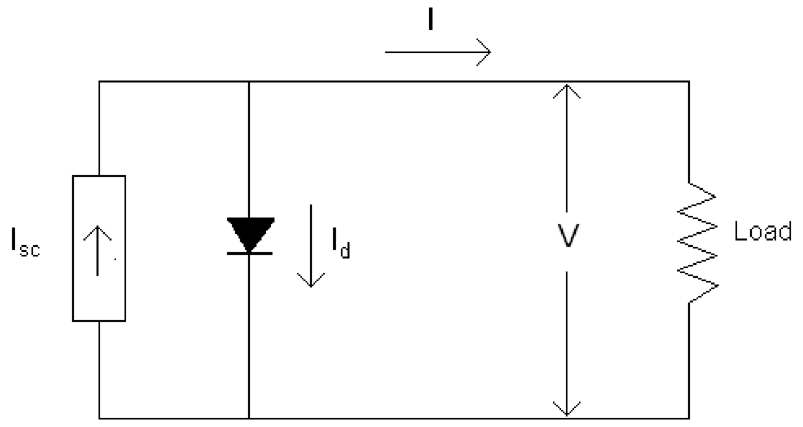


Figure 14: A simple equivalent circuit for a solar cell.

Hence, the actual current in a solar cell is the ideal current, *i.e.* the short-circuit current I_{SC} (A), minus the current that flows through the diode, I_d (A) [4]:

$$I = I_{SC} - I_d. \tag{17}$$

To get the relation of current and voltage in a PV cell we substitute the

Shockley diode equation (16) to the equation (17) above. This yields [4]

$$I = I_{SC} - I_0(e^{qV_d/kT} - 1), \tag{18}$$

which is just the diode equation reduced from the I_{SC}. Hence the graph for current-voltage characteristics of a solar cell, shown in figure 15, is as the graph for diode but inverted by convention and shifted up by I_{SC}. The open-circuit voltage V_{OC} is acquired from the equation (18) by setting the current to zero, which yields [4]

$$V_{OC} = \frac{kT}{q} ln\left(\frac{I_{SC}}{I_0} + 1\right). \tag{19}$$

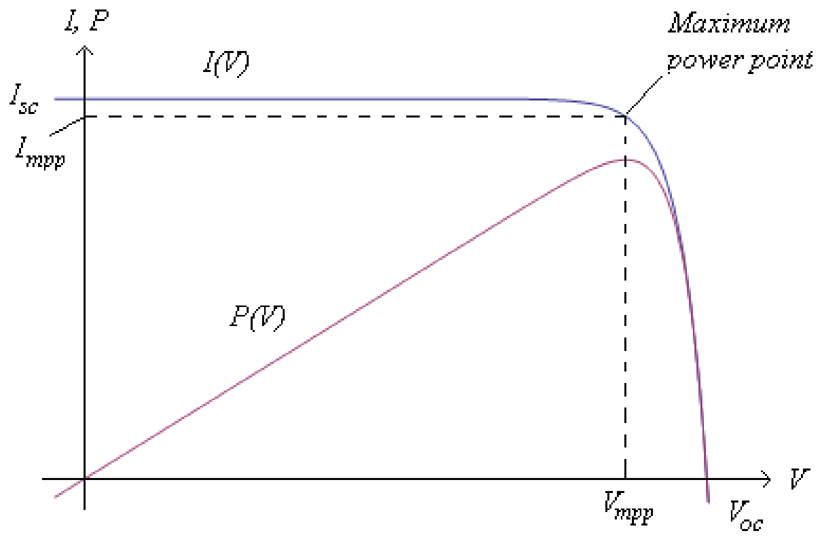


Figure 15: The current-voltage relationship of a solar cell ($I - V$ curve) accompained by power as a function of voltage.

In figure 15 also indicated are the power as a function of voltage (calculated as a product of current (equation (18)) and voltage), and the maximum power point current and voltage (I_{mpp} (A) and V_{mpp} (V)). I_{mpp} and V_{mpp} are the current and voltage combination that produces the maximum power. The maximum power point for the $I - V$ curve can be found using the maximum point of the power curve, or by fitting the largest area rectangle under the $I - V$ curve. The maximum power P_{max} (W) is calculated as a product of I_{mpp} and V_{mpp} [7]:

$$P_{max} = I_{mpp} V_{mpp}. \tag{20}$$

A parameter widely used in describing the performance of a PV module is its maximum power at Standard Test Conditions (STC, *i.e.*, irradiance level of 1000 W/m^2, module temperature of 25 °C and AM1.5G spectral irradiance distribution). Another parameter describing the module performance is the fill factor (FF), which is the ratio of P_{max} and the product of I_{SC} and V_{OC} [7]:

$$FF = \frac{P_{max}}{I_{SC} V_{OC}} = \frac{I_{mpp} V_{mpp}}{I_{SC} V_{OC}}. \tag{21}$$

Thus the fill factor is the ratio of the areas of two different rectangles, and describes how "rectangular" the $I - V$ curve is. Furthermore, the efficiency of a solar cell cell is defined as [7]

$$\eta = \frac{P_{max}}{P_{in}}, \tag{22}$$

where P_{in} is the power of the light incident on the cell (W). [4, Chapter 8]

3.2.2 From cells to modules and arrays

As the voltage output of one PV cell is only about 0.5 V, several cells are coupled in series in order to increase the ouput. An assembly of a number of cells coupled (typically 36) and encapsulated in an appropriate package is called a solar or PV module, that is the basic element for photovoltaic electricity production. The voltage output of a module is a sum of the voltages of its cells connected in series. To increase the current, cells are connected in parallel, and similarly the current output of a module is a sum of the currents of the idividual cells connected in parallel. In the $I - V$ curve of a module with a number of cells connected in series and parallel the curves of individual cells simply add up along the voltage and current axes, respectively. [4, Chapter 8]

To increase the output even more for a larger-scale electricity production system, several modules are connected in series and parallel to form a PV array. Again, connecting modules in series increases the voltage and connecting them in parallel increases the current, and the system design determines the most suitable combination. [4, Chapter 8]

3.2.3 Effects of temperature and irradiance on the module performance

The output of a module is obviously directly proportional to the incident irradiance; a high module temperature in turn impacts negatively on the performance of a module. Roughly speaking, the module current is dependent on the incident irradiance and independent on the module temperature, and for the module voltage the situation is reversed. The higher the intensity of radiation incident on a solar module, the more there are photons to excite electrons in the semiconductor and hence the higher is the current; the short circuit current is almost directly proportional to the incident irradiance. However, as may be seen from equation (19), the open circuit voltage has a logarithmic dependence on the I_{SC} and thus also on the irradiance. Thus the voltage increases only slightly with increasing irradiance. [4, Chapter 8]

When the module temperature increases, the reverse saturation current across the p-n junction increases and the band gap energy decreases. This results in a decrease in the module voltage: for crystalline silicon solar cells, for an increase of one degree celsius in module temperature the V_{OC} drops by about 0.37%. On the other hand, due to the smaller band gap there are more electrons exicted to the conduction band, and thus the module current increases slightly: I_{SC} increases approximately 0.05% per degree celsius. Consequently the maximum power output drops by about 0.5% per degree increase in temperature. Significantly, not only the ambient temperature affects the module temperature but also the incident irradiance: as was explained in section 3.1.1, a remarkable portion of the incident irradiance is wasted as heat. [4, Chapter 8], [1, Chapter 4]

3.2.4 Performance measurements of PV modules

Performance measurement of a photovoltaic solar module is "the measurement of the current-voltage relation, acquired at a known irradiance, temperature and spectral content" [9]. In other words, it means measuring the $I-V$ curve. As the module is illuminated, either indoors with a solar simulator or outdoors under real sunlight, it produces a certain current at a certain voltage, and to acquire all the points along the $I-V$ curve, we need to adjust the load in the circuit in order to vary the current in the range from I_{SC} to zero. This can be executed using either passive or active load.

Performance measurement passively load means increasing the load in the circuit using typically a resistive load. Alternatively, a capacitive or diode load can be used. According to Ohm's law $V = IR$; thus by inreasing the resistance we increase the voltage and move along the $I-V$ curve. The major drawback in this measurement method is its slowness: it requires a long period of constant irradiance and temperature conditions, which can be hard to obtain, as it will be seen in section 4. [9]

Measuring the $I-V$ curve with an active load means varying the current by applying a voltage ramp across the module. A voltage ramp stands for a varying reverse voltage that increases more or less uniformly from 0 V to V_{OC}. Unlike in the case of a passive load, using an active load the performance measurement can be executed very fast (in the order of below a millisecond), which is desirable especially in indoor performance measurements with flash simulators (see section 4.3). [9]

3.3 Different PV module technologies

There is a wide range of different photovoltaic technologies, and they are categorized in many different ways. The major categorization is done according to the thickness of the layer of the photovoltaic material used. In conventional crystalline silicon (c-Si) solar cells, the layer is relatively thick – in the order of $200 - 500$ μm. Another approach, referred to as thin-film technology, uses layers of only $1 - 10$ μm of the photovoltaic material. As the thin-film modules recquire much less of the semiconductor material, and their manufacturing process is also much less energy intensive, they ought to be cheaper to produce than the conventional c-Si modules. However, the efficiencies of thin-film modules have so far been lower than those of the conventional ones, and also their life-times have not been demonstrated. Nevertheless, thin-film technology has been improving a lot making it a more and more tempting option to replace the conventional technology. [4, Chapter 8]

3.3.1 Crystalline silicon technologies

The basis of a conventional thick c-Si cell is a wafer made of either single- or multi-crystalline silicon. The major drawback in this technology is the costly manufacturing process. Though silicon is one of the most abundant elements on the earth, in nature it never excists as pure silicon but as SiO_2

based minerals. The process for purifying the silicon is very energy-intensive, and so are the processes for producing the single- or multi-crystalline silicon crystal from the purified silicon. To produce single-crystalline silicon, an ingot composed of a single silicon crystal is formed by growing it from a purified silicon melt using a "seed crystal" (the Czochralski process). To produce multi-crystalline silicon, the silicon melt is cast into large crucibles. This process is less energy-intensive than the Czochralski process, and it is currently the most common method. An alternative approach is to grow a silicon ribbon from molten silicon. To make wafers, the ingot and the crucible need to be sawed into thin slices, which wastes a lot of silicon as saw dust called *kerf*; the ribbon on the other hand can be directly cut into rectangular cells, and the kerf losses are avoided. [4, 3, Chapter 8]

Once the wafers have been produced, they are doped to form the p-n junction. The dopants are usually added to the wafers in gaseous form. Furthermore, some surface treatment is required for the cells to minimize reflectance and maximize absorption of light. Finally, the electrical contacts are attached to the cells and the individual cells are wired together, and the whole assembly is encapsulated into an appropriate package. [4, Chapter 8]

The c-Si solar cells have reached the highest efficiencies among single-junction solar cells so far (see following section). The theorethical upper limit for the efficiency is almost 50%, and efficiencies of almost 25% have been reached in laboratories, as it was already mentioned in section 3.1.1. For modules in production, efficiencies of 14-17% have been reached. [4, Chapter 8]

3.3.2 Thin-film technologies

The manufacturing process of thin-film modules is quite different from that of the c-Si ones. Instead of making separate wafers, the semiconductor materials used in thin-film photovoltaics are deposited in gaseous form onto glass or metal substrates. The whole module is made in one time: To the glass superstrate first a transparent conductor and long metal bars that connect the adjacent cells are attached. Subsequently the different semiconductor layers are deposited in gaseous form, and finally the bottom conductor is attached. The straightforward manufacturing process makes thin-film modules especially suitable for mass production, which naturally brings their costs down. [4, Chapter 8]

As the layer of the photovoltaic material is very thin in thin-film modules, less photons are absorbed, and part of the light passes right through. This results in smaller efficiencies as compared to c-Si modules, but also opens up new opportunities. First of all, the semitransparent photovoltaic material can be deposited on glass to make windows that also produce electricity. Secondly, thin-film materials can be used to produce highly efficient multiple junction solar cells, where two or more junctions with different bandgaps are stacked on top of each other. The multijunction solar cells are constructed so that the uppermost layer exploits the shortest wavelengths (highest-energy photons) and the lowermost exploits the longest wavelengths. This is done by choosing semiconductor materials with higher band gaps to the uppermost layer, and materials with lower band gaps into the subsequent layers in descending order. [4, Chapter 8]

Most of the thin-film modules are currently made of amorphous silicon (a-Si), but also different compounds are used: Gallium Arsenide (GaAs), Cadmium Telluride (CdTe) and Copper Inidium Diselenide (CIS). Amorphous silicon, that is silicon with very little order in the atomic arrangement, has the advantage of the good availability of silicon. It is also very suitable for production of multijunction modules. However, the problem with a-Si modules is that their performance degrades significantly during the first months of operation: commercial modules have stabilized efficiencies of only 5-8%, though in laboratories stabilized efficiencies of 13% have been reached. The advantage of GaAs and CdTe is their close-to optimal band gaps (see section 3.1.1), 1.43 eV for GaAs and 1.44 eV for CdTe, resulting in relatively high efficiencies (see table 1). But these materials are not trouble-free either: gallium is a rarer element than silicon, which lifts its costs, and cadmium in turn is highly toxic, which causes safety problems in both manufacturing and usage of CdTe modules. The highest efficiencies so far among single-junction thin-films at both laboratories and industrial scale have been reached with CIS cells and modules [10]. Furthermore, the homogenous and opaque appearance of CIS modules renders them especially suitable for building integration of PV modules, which offers new opportunities for popularization of photovoltaics [10]. [4, Chapter 8]

The multijunction technology is also promising, and very high efficiencies have been reached with it. With multijunction GaAs-GaInP (Gallium Inidium Phosphorus) solar cells, 29.5% efficiencies have been reached and these cells are used in space applications and for concentrator PV. For multijunction a-Si, the theorethical upper limit for the efficiency is 42% and for modules in laboratories stabilized efficiencies of 11% have been reached. [4, Chapter 8]

Table 1: Different efficiency values for different module technologies. MJ refers to multijunction. The values are received from [4].

	c-Si	a-Si	GaAs	CdTe	CIS	MJ GaAs	MJ a-Si
Theorethical	49.6%	28%	29%				42%
Laboratory	25%	13%	20%	16%	20%	29.5%	11%
Commercial	14-17%	5-8%		9%	8-10%		

3.3.3 A glance over the past and future of the different technologies

In the infancy of the terrestrial photovoltaic industry in 1970s, c-Si modules were extremely expensive. That time there was lots of discussion about a new technology, probably thin-film, that would overcome the costly c-Si in near future. Now, 30 years later, c-Si is still dominating the markets with more than 90% share. Why? [3]

The costs of c-Si modules have been progressively reducing at a rate of 20% per every doubling in the cumulative production [3]. This has been a result of many different factors: Firstly, the efficiency of c-Si modules has been increasing, which naturally improves their profitability. Secondly, the wafer thickness has been decreasing, which has lead into cost reductions as less material is required. Furthermore, the manufacturing process of c-Si modules has developed, which encompasses for instance improvements in the sawing technology and automation of the process, making the production more economic. [3]

With the continuously decreasing prices and increasing production of c-Si modules, it is hard for a new technology to penetrate to the markets. The competing technology should have immediately high efficiency, long module life-time and ability to rapidly scale up the production – a hard goal to reach for a technology in pilot stage. However, there is still an opportunity for thin-films and other promising technologies. The ongoing reduction in the prices of c-Si modules will probably reach its limit within 10 years – it has already slowed down due to the current silicon shortage. Furthermore, as discussed in previous section, the thin-film technologies have some tempting opportunities to offer, including for instance the multijunction technology and a better ability for building integration, which improves their competitiveness. [3]

4 Measurements on PV modules at the ESTI

It is essential to thouroughly study all new module technologies being developed. Above all, the performance of the modules has to be studied under wide range of different conditions. Other properties to be tested before the modules can be introduced to markets are for instance their lifetime and long-term behaviour, as well as their mechanical endurance.

There are several laboratories dedicated for testing PV modules. The European Solar Test Installation (ESTI) at the Joint Research Centre (JRC) in Ispra, Italy, is one of them; other laboratories are for instance the National Renewable Energy Laboratory (NREL) in the U.S. Department of Energy and the Laboratory of Energy, Ecology and Economy (LEEE-TISO) in the University of Applied Sciences of Southern Switzerland (SUPSI), to mention but a few. To guarantee the quality of their results, some of the laboratories (including ESTI) are accredited according to quality standard IEC/ISO 17025: "General requirements for the competence of testing and calibration laboratories". The International Organization for Standardizaton (ISO) and the International Electrotechnical Commission (IEC, or Commission Electrotechnique Internationale, CEI) are international bodiestasked with creating a specialized system for standardization. In order to be accredited, the laboratory has to perform its tests according to internationally agreed and standardized methods. Between the laboratory and the ISO and IEC standards, there is a national accreditation body that audits the laboratory and ensures that it obeys the standards. The accreditation body that for instance ESTI uses is Cofrac[3]. [11, 12]

Different PV module types have earlier been ranked mainly according to their maximum output power (P_{max}) at Standard Test Conditions (STC, *i.e.*,

[3]Le Cofrac, `http://www.cofrac.fr/` (5/2008)

irradiance level of 1000 W/m^2, module temperature of 25 °C and AM1.5G spectral irradiance distribution). The STC power, however, does not tell the whole story: modules having equal STC powers may have very different long-term energy yields, even when located in the same place. Therefore an energy rating method has been developed for predicting the module performance and hence providing a more reliable tool for module comparison. [13]

This chapter discusses ESTI and the different measurements performed there. Focus is on the outdoor measurement system (section 4.2) and indoor performance measurements at STC as well as so-called performance surface measurements (section 4.3). Additionally the energy rating method being developed at ESTI, that is based on the indoor performance surface measurements, is introduced (section 4.4). These topics were essential in the realized study as it will be seen later in this thesis.

4.1 The European Solar Test Installation (ESTI)

The work at ESTI takes place under the Solar Electricity Action (SOLAREC) of the Renewable Energies Unit in the Institute of Energy (IE) of the JRC. The primary objective of ESTI is "to provide a sound and credible assessment of all aspects of Photovoltaic Solar Energy, assisting both policymakers and industry, but also standards organisations and national research agencies." [14]

The testing facilites of ESTI have been built up since 1977, and they comprise for example precision calibration facilities with two Large Area Pulsed Solar Simulators (LAPSS), an outdoor testing field and several climatic chambers for accelerated lifetime testing of the modules. In 1996 ESTI became the first solar testing and laboratory to obtain the status of accredited test laboratory for Photovoltaic devices under the EN45001 scheme. In

2001 this became the ISO 17025 scheme. [14]

In addition to testing, ESTI is accredited for calibration of photovoltaic devices. Field experience has lead into development of standardized test sequences such as CEI/IEC 61215: "Crystalline Silicon Terrestrial Photovoltaic (PV) Modules – Design Qualification and Type Approval", which enables identification of potential defects in a module type in an accelerated time period. This test sequence requires eight modules taken randomly from a production batch, or alternatively a prototype of a new design. Each module undergoes an individual testing sequence where its electrical, optical or mechanical characteristics are tested. A similar test sequence for thin film modules is documented in CEI/IEC 61646 : "Thin film Terrestrial Photovoltaic (PV) Modules – Design Qualification and Type Approval". The actual test results are treated confidentially, but ESTI maintains a public list of qualified module types. A module is regarded as qualified if it during and after the test sequence meets certain criteria, such as that there is no major visual defect (broken window, bubbles etc.) and that the degradation of its maximum output power at STC remains within certain limits. [15]

The modules are also treated confidentially while being tested at ESTI: to protect the name of the manufacturer the modules are assigned anonymous code names upon their arrival to ESTI. The code name is composed of two letters and two to three numbers, such as ju711 and by71.

4.2 Outdoor measurements

Rapid performance measurements under well-defined conditions are not sufficient for describing the actual performance of a PV module. Some features of different module technologies emerge only after a long-term measurement period under varying environmental conditions, *i.e.*, varying ambient temperature, wind speed and intensity and spectral distribution of irradiation. This is the motivation for the outdoor measurements performed at ESTI. The present section describes the testing setup, the monitoring devices for the modules and environment, and the monitoring data received from the setup.

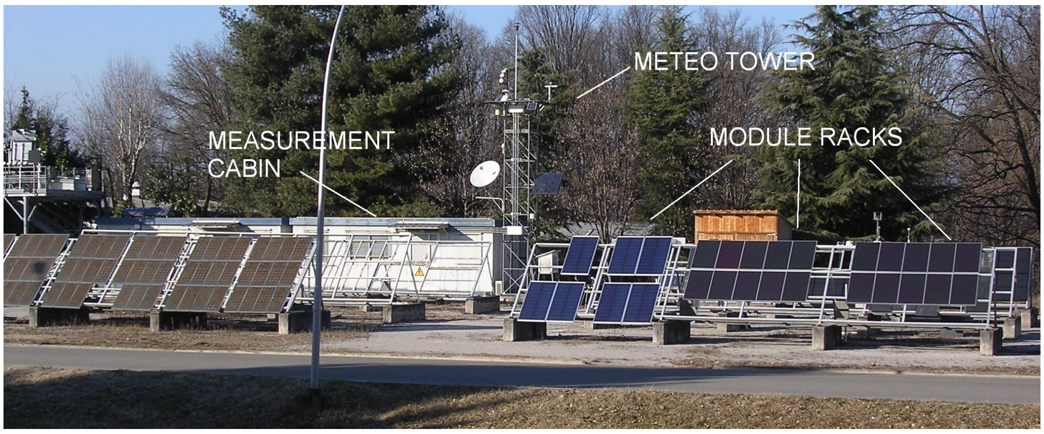


Figure 16: The outdoor measurement field of ESTI at the JRC in Ispra, Italy.

4.2.1 Testing setup

Figure 16 shows the outdoor test field of ESTI located at the Joint Research Centre (JRC) in Ispra, Italy (45° 48′ 42″ N and 08° 37′ 36″ E). There are several module racks that are equipped with adjustable supporting arms. Previously, the inclination of the racks used for some of the long term tests was adjusted once a season in order to keep the module plane approximately normal to incident irradiance at solar noon, but now they are kept fixed in accordance with the practise in the second edition of the IEC 61215 standard. A number of environmental sensors (see section 4.2.2) are located near the modules, and there is additionally a meteorological tower providing a wider range of environmental data. [16]

The modules and the sensors (environmental sensors and the ones measuring module temperatures) are connected via underground cables to the measurement cabin (see figure 17). In the cabin there are two Hewlett Packard dataloggers that measure all the electrical signals coming from the modules and the sensors, and six Kepco bipolar power supplies that provide the active load for measuring the $I-V$ characteristics – thus the performance of six modules can be measured simultaneously. To monitor the data from the dataloggers and to control the power supplies there is a computer running Microsoft Windows© with a dedicated software written with National Instruments LabVIEW©[4]. The modules are kept at maximum power point between the measurements.

A performance measurement is performed at regular intervals (usually every four minutes) as long as the incident irradiance is above 50 W/m^2. Duration of a measurement is approximately 5 seconds per module (or 10, depending on the number of points scanned), and it is performed for each module successively; that is, the 5-second measurement is performed every four minutes for all the modules in a certain order. At each measurement the software records the $I-V$ characteristics, environmental parameters (see the following section) and the module temperature. Additionally point to point current, voltage and irradiance values are recorded over the scan: typically 30 current, voltage and irradiance values are read during a single scan of 5 seconds duration. The software writes the data daily into three different text files (see section 4.2.3), and the files are transferred automatically to a local server for daily back-up. [16]

[4]NI LabVIEW (2008), `http://www.ni.com/labview/` (5/2008)

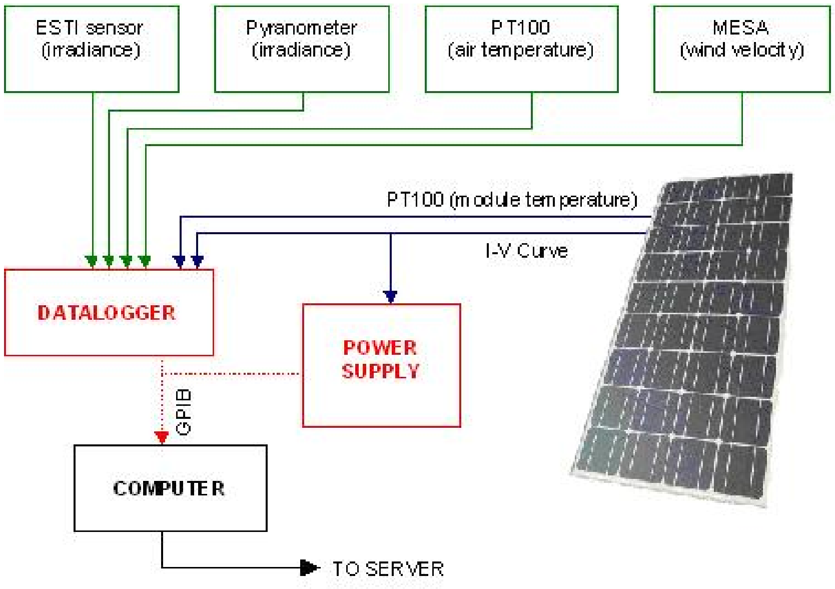


Figure 17: A schematic diagram of how the modules and sensors are connected to the instruments driving and recording the measurements outdoors. Image reproduced from [16] (fig. 4.6).

4.2.2 Environmental measurements

The environmental parameters that are recorded at each measurement are ambient temperature, wind velocity and irradiance. The environmental sensors, shown in figure 18, are all located near the module racks.

Ambient temperature is measured with a PT100 temperature sensor as shaded from the sun. PT100 is a resistance temperature detector: it is based on platinium that produces a well-known change in resistance with temperature. A similar device is used for measuring the module temperature and is attached to the back of the module. [16]

The wind velocity (speed and direction) is measured with a MESA Ultrasonic Anemometer [16]. The operation of an ultrasonic anemometer is based on measuring the propagation velocity of sound in air [17].

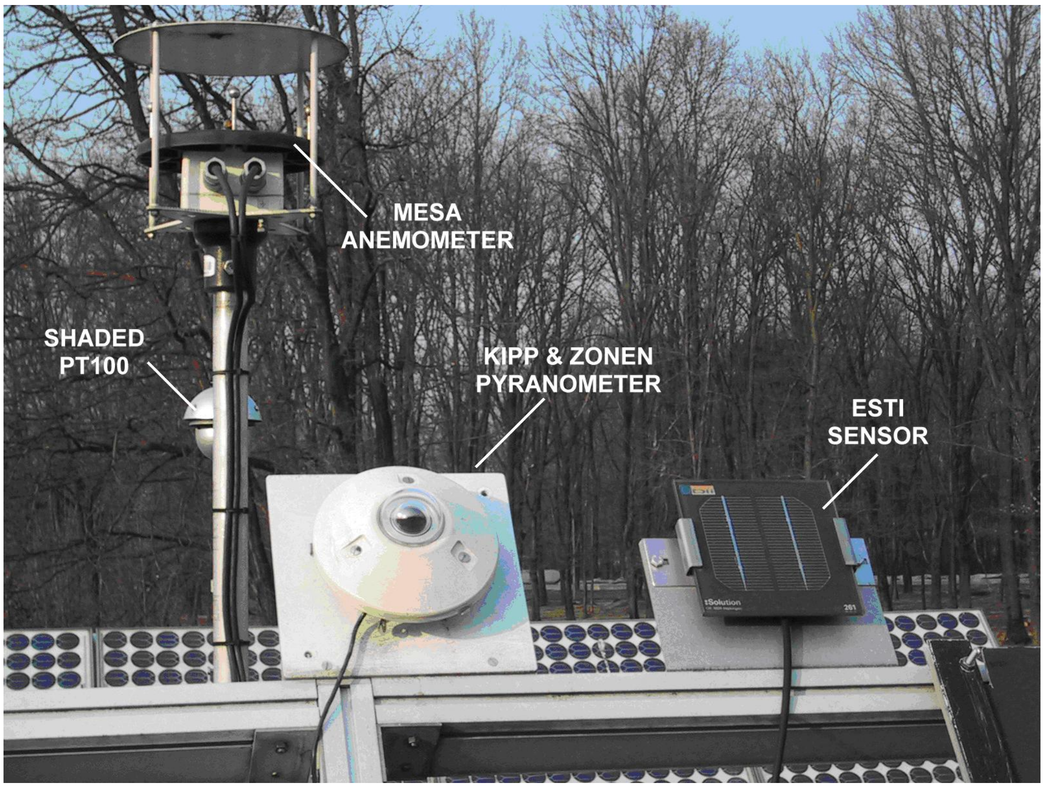


Figure 18: The environmental sensors used. Image reproduced from [16] (fig. 4.7).

The irradiance is measured with two different devices: a pyranometer and a reference cell, referred to as an ESTI sensor, both mounted coplanar with the modules. The ESTI sensor consists of two identical single-crystalline silicon solar cells; figure 19 shows the equivalent circuit for the sensor. The first cell is kept in short circuit to monitor the irradiance, and the second cell remains at open circuit. The first part is connected to a precision shunt resistor and the irradiance, G_{ESTI} (W/m^2), is defined as follows [16]:

$$G_{ESTI} = \frac{V_{SC}}{k} 1000, \tag{23}$$

where V_{SC} is the voltage across the shunt (mV), and k is a calibration factor (mV/Wm^{-2}). The open-circuit voltage (V_{OC}) measured from the second part can be used to make temperature correction to the irradiance. [16]

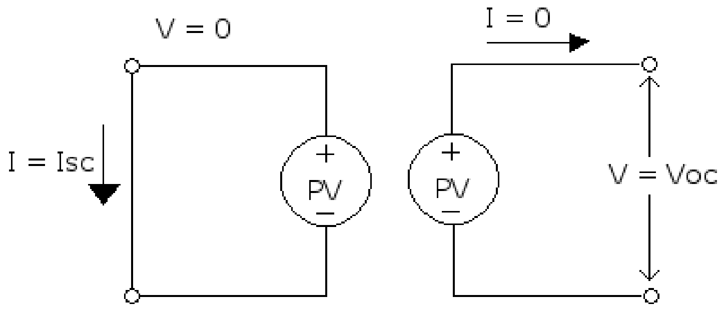


Figure 19: The equivalent circuit of an ESTI sensor. The ammeter and the voltmeter are excluded.

The CM11 pyranometer from Kipp & Zonen (see figure 20) is based on a thermal detector absorbing the solar irradiance. Hence it detects all the incident irradiance – all wavelengths and from all directions. The radiation energy is absorbed by a black painted ceramic disc (sensing element in figure 20) equipped with a thermopile imprinted on it. The border of the disc is in thermal contact with the pyranometer body, and the generated heat flows through the thermopile to the body that acts as a heat sink. The irradiance is determined from the temperature difference across the disc; the thermopile converts the difference into a voltage. The detector is shielded with two glass domes in order to prevent the effect of wind, rain and thermal radiation losses to the temperature rise. [18]

As these two irradiance meters are based on completely different technologies, they are also very different in their properties. First of all, their spectral responses are different: ESTI sensor detects much more narrow range of the solar spectrum (from 350 nm to 1200 nm) than the pyranometer, whose spectral range (from 305 nm to 2800 nm) is limited only by the transmittance

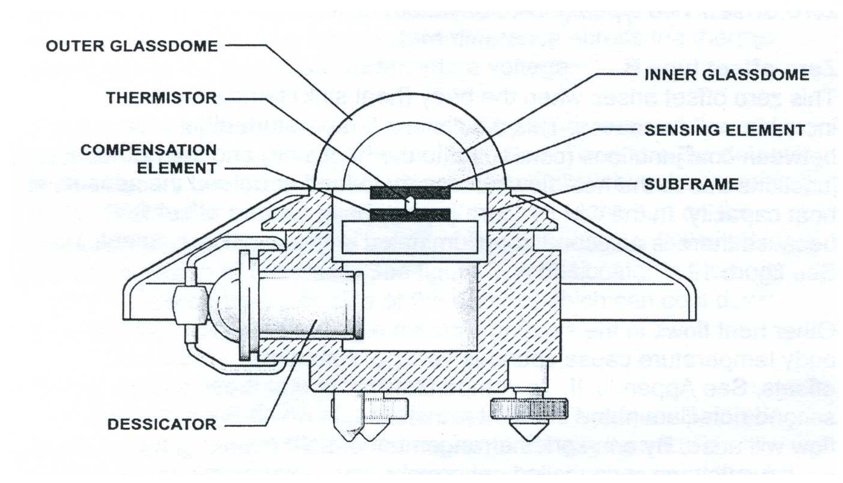


Figure 20: Approximate construction of the Kipp&Zonen CM11 pyranometer. Image reproduced from [18] (fig. 1).

of the glass. The spectral response of the ESTI sensor is similar to that of the c-Si modules, as they are based on similar technology. On the other hand, due to the wider spectral response the irradiance value received from the pyranometer corresponds more realistically to the actual energy received from the sun and might therefore be more correct for calculating for instance, module efficiencies. [16]

Another prominent difference between the ESTI sensor and the pyranometer is that the former reacts much faster to rapid changes in the irradiance (caused by clouds moving past the sun) than the pyranometer; *i.e.*, pyranometer has a much slower response time ($<$ 15 s [18]) than the ESTI sensor (instantaneous response). This feature is seen in figure 21, where point to point irradiances from both devices are plotted from a single measurement (the duration of a measurement is approximatey ten seconds), during which irradiance has changed rapidly. In the figure the ESTI irradiance rises and drops earlier than the pyranometer irradiance, and the changes in the latter are also less prominent. Because of this feature the ESTI sensor is usually preferred in measurements – it gives a more realistic value for the instantaneous irradiance. Furthermore, a pyranometer is more expensive than a c-Si reference device and it has also a lower output signal [16].

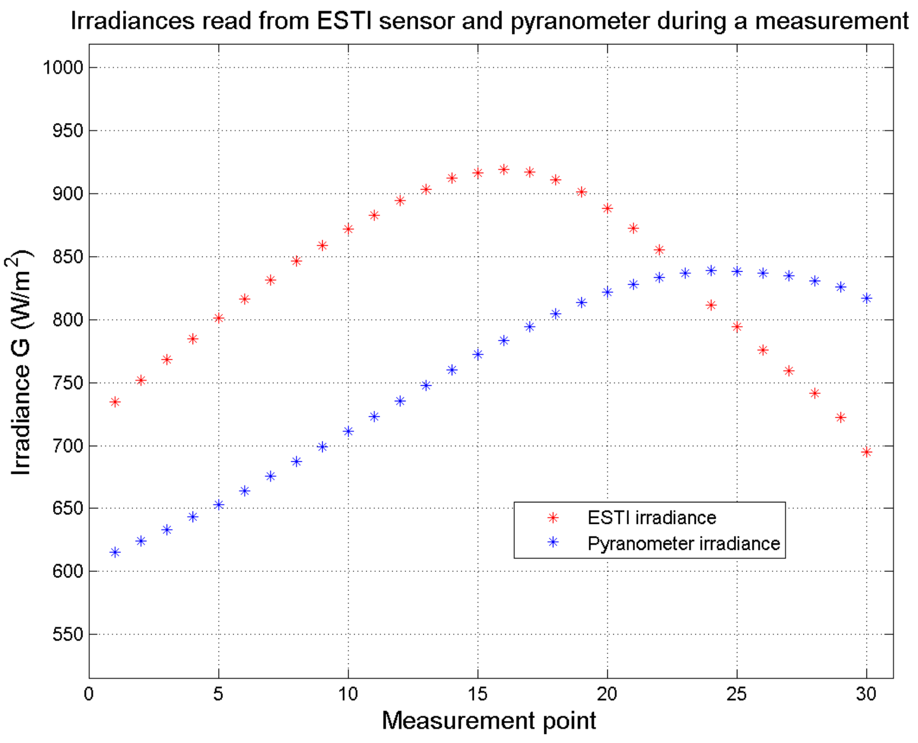


Figure 21: The point to point irradiances recorded from ESTI sensor and pyranometer during a measurement on 25.6.2007. The duration of a measurement is approximately 5 seconds.

Since an $I-V$ curve is defined at a specific irradiance level, large changes – say, more than 5% – in irradiance during the 5 -second scan of an $I-V$ curve result in faulty curve and thus in erroneous P_{max} values (see figure 22). This problem is treated by making corrections to the data after the scan; this is explained in more detail in the following section.

4.2.3 Monitoring data

The outdoor measurement system produces daily three different text files for each module being measured: enr-, ene- and raw-files. The content of these files is listed in tables 2-4, respectively. In the ene-files faults caused by changes in the irradiance during a measurement have been taken into account; in the enr-files they have not. The raw -files consist of a larger dataset than the first two.

Table 2: The parameters in the enr-file

Parameter	Explanation	Units
`Date`	Day of measurement	
`Time`	Local time on datalogger at the measurement	
`ESTI`	Irradiance from ESTI sensor,	
	measured at the beginning of the scan	W/m^2
`Pyran`	Irradiance from pyranometer,	
	measured at the beginning of the scan	W/m^2
`Tmod`	Module temperature	°C
`Tamb`	Ambient temperature	°C
`Isc`	Short-circuit current	A
`Voc`	Open-circuit voltage	V
`Pmax`	Maximum power	W
`Impp`	Maximum power point current	A
`Vmpp`	Maximum power point voltage	V
`FF`	Fill factor	

In the enr-files there is one line for each measurement of uncorrected data. This means that the possible changes in irradiance have not been considered, but $I-V$ characteristics found in the enr-file are based on the original values received from the data logger. As explained in section 4.2.1, the $I - V$ curve scan lasts approximately 5 seconds, and in addition to current and voltage values, point to point irradiances are recorded during the scan. The irradiance values recorded in enr-files are those measured at the beginning of the scan.

The ene -files were introduced in 2005 in order to correct erroneous $I - V$ characteristics resulting from significant changes in irradiance during the $I - V$ curve scan. In the ene-files there is also one line of data for each measurement. On each line there are three different irradiance values: the average value of the irradiances recorded during the $I - V$ curve scan from both ESTI sensor and pyranometer, and the minimum irradiance of the scan from the ESTI sensor, which is used as the corrected irradiance. The $I - V$ curve is corrected by shifting the current values, in accordance with IEC 60891, with a coefficient proportional to the correction made to the ESTI irradiance. The $I - V$ characteristics found on each line of an ene-file are extracted from the new, corrected $I - V$ curve. In figure 22, an example of original and corrected $I - V$ curves is plotted. These curves are from the same measurement as the irradiances plotted in figure 21.

The reason to correct the $I - V$ curve according to the minimum value of ESTI irradiance of the scan (not for instance according to the average value) is that the power supplies always perform the $I - V$ curve scan according to the initial irradiance value, that is, the voltage bias used to bring the current down is scaled according to the irradiance measured at the beginning of the scan. If a value other than the minimum ESTI irradiance is chosen for the correction, the corrected $I - V$ curve might not necessarily reach the current zero-crossing and hence the V_{OC}. Furthermore, the reason to do the correction according to the minimum value of ESTI sensor, not pyranometer, is the

Table 3: The parameters in the ene-file

Parameter	Explanation	Units
`Date`	Day of measurement	
`Time`	Local time on datalogger at the measurement	
`ESTI Irr Ave`	The average irradiance of the scan from ESTI sensor	W/m^2
`Pyran Irr Ave`	The average irradiance of the scan from pyranometer	W/m^2
`Corrected to Irr`	The minimum irradiance of the scan from ESTI sensor	W/m^2
`Irr Change`	The difference between minimum and maximum value of the scan of ESTI irradiance	%
`ESTI Volts Ave`	Open circuit voltage of the ESTI sensor (average of the scan)	V
`Tmod`	Module temperature	°C
`Tamb`	Ambient temperature	°C
`Isc`	Short-circuit current from the corrected $I-V$ curve	A
`Voc`	Open-circuit voltage from the corrected $I-V$ curve	V
`Pmax`	Maximum power from the corrected $I-V$ curve	W
`Impp`	Maximum power point current from the corrected $I-V$ curve	A
`Vmpp`	Maximum power point voltage from the corrected $I-V$ curve	V
`FF`	Fill factor from the corrected $I-V$ curve	

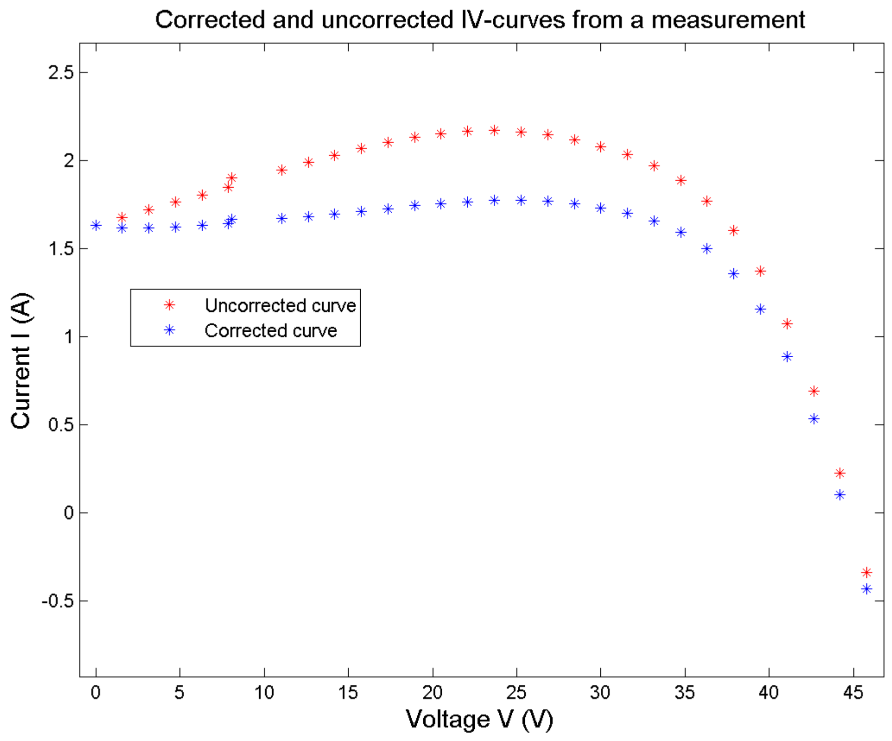


Figure 22: The original and corrected IV-curves from the same measurement as the irradiances plotted in figure 21.

slow response time of pyranometer – its readings may not tell the true story about the irradiance at the time of the measurement. However, as mentioned earlier, the pyranometer irradiance might represent more realistically the actual amount of irradiance coming from the sun. Examples of attempts at finding a correct pyranometer value for each measurement are presented in section 5. [19]

The body of the raw-files differs from that of the ene- and enr-files. For each measurement there is firstly one line with nine uncorrected parameters accompained with two corrected parameters: `ESTI`, `Pyran`, `Tmod`, `Tamb`, `Isc`, `Voc`, `Pmax`, `Impp`, `Vmpp` and `FF` (see table 2), and `Corrected to Irr` and `Irr Change` (see table 3). Additionally there is for each measurement six columns of point to point data from the scan, explained in table 4. The raw -files are useful when one wants to go back to individual measurements and see what actually happened during the $I - V$ curve scan – for instance the graphs in figures 21 and 22 were reconstructed from raw-file data.

Table 4: The point to point data found in the raw-file.

Parameter	Explanation	Units
`Module Volts`	Point to point module voltage	V
`Module Current`	Initial point to point module current	A
`ESTI Irr`	Point to point irradiance from ESTI sensor	W/m^2
`Pyran Irr`	Point to point irradiance from pyranometer	W/m^2
`ESTI Volts`	Point to point open circuit voltage from ESTI sensor	V
`Corrected module current`	Corrected point to point module current	A

4.3 Indoor performance measurements

Indoor measurements at ESTI cover a wide range of different measurements testing various module features as mentioned in section 4.1. In this section focus is on the basic performance measurement at standard test conditions (STC) and on so-called performance surface measurements. The first-mentioned is the main characterization measurement for modules at ESTI and the latter measurement is crucial for the energy rating method and thus also for the analysis presented in this thesis (see sections 4.4 and 5).

The performance measurements at STC and the performance surface measurements are both executed using flash solar simulators, which consist of electric lamps simulating the terrestrial solar radiation. In the simulators, xenon lamps are used as their spectrum is rather similar to that of the sun. However, the spectrum is clearly not exactly the same as the sun, and in steady-state solar simulators, ageing of the lamp and filter instability further distort the similarity. Flash simulators are more common since the service life of the lamps increases when the measurement time is minimized, although the principal reason for using a flash simulator is economic – it is vastly expensive to construct and operate a large area steady-state solar simulator. Nevertheless, there are some problems involved in this technique: Firstly, the spectrum of the lamp changes as it warms up. Secondly, the flash technique requires a fast (a few milliseconds) sweep of the $I-V$ curve, which means that a rapid voltage bias is applied to the module – and the high bias rate may lead to erroneous measurement for some module types, for example, those exhibiting large capacitance. [16]

4.3.1 Performance measurements at STC

The solar simulator used in the STC performance measurements is a Spectrolab Large Area Pulsed Solar Simulator (LAPSS). It contains two xenon lamps and generates an irradiance pulse whose intensity increases rapidly, stays approximately uniform for 1 ms and then decreases rapidly. The $I-V$ curve measurement is executed simultaneously with the flash from the lamp by applying a voltage ramp across the module with a Kepco power supply.

The module being tested is placed vertically on a rack, at a distance from the solar simulator at which the irradiance is approximately uniform on each cell of the module and the intensity of the irradiance is 1000 W/m^2 during the peak of the pulse. The irradiance is measured using a c-Si reference cell (ESTI sensor) placed co-planar next to the module, and the module temperature is measured with a PT100 temperature sensor attached to the back of the module, as in outdoor measurements. The measurements are performed in a room having black surfaces to exclude the effect of reflections, and the room temperature is kept at $25 \pm 2°C$. [16]

The performance measurement at STC is performed according to standard CEI/IEC 60904-1 "Measurement of photovoltaic current-voltage characteristics", which sets limitations for instance for the spectral response and placement of the reference cell relative to the module being tested, and to the accuracy of temperature, voltage and current measurements. [16, 20]

The reference cell, that is basically a c-Si cell, and the test module are often made of different materials. Hence their spectral responses can be very different, as figure 23 illustrates. This defect is treated by using an offset called *spectral mismatch factor* for correcting the $I - V$ curve. In the standard CEI/IEC 60904-7: "Computation of Spectral Mismatch Error Introduced in the Testing of a Photovoltaic Device" (1995) the spectral mismatch factor is defined as "the error in the measured I_{SC} of a solar cell due to differences between the light source spectral irradiance and the reference spectral irradiance, and the difference between the spectral responses of the test and reference devices" [16]. At ESTI, the spectral mismatch factor is determined experimentally for a particular module and reference cell combination, using a dedicated measurement set-up. [16]

4.3.2 Performance surface measurements

In the indoor performance surface measurements of the ESTI the module performance is measured not only at STC, but at a wide range of module temperature and irradiance values. The solar simulator used in these measurements is Pasan LAPSS, that consists of only one xenon lamp. It

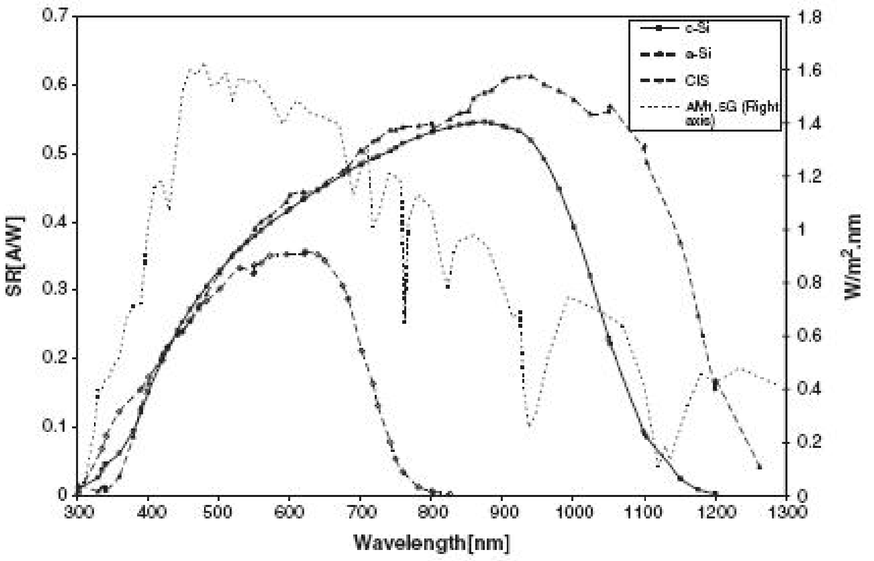


Figure 23: Spectral responses of different module types (c-Si, a-Si and CIS) and the AM1.5G spectrum. ESTI sensor spectral response is very similar to the c-Si response. Image reproduced from [21] (fig. 3).

generates a pulse whose intensity increases rapidly to a peak and then decays slowly in approximately 20 ms. The shape of the pulse is exploited in the measurements: the $I - V$ curve measurement is performed at different irradiance levels by adjusting the timing of the measurement relative to the phase of the pulse. To adjust the module temperature the module is placed in a temperature controlled chamber having a quartz window. The module temperature is varied from 25 °C to 60 °C, and the $I - V$ curve measurement is performed at different irradiances in the range of $50 - 1000\text{W/m}^2$ at each temperature. Similarly to the performance measurements at STC, the irradiance in performance surface measurements is also measured with an ESTI sensor, placed co-planar next to the module, and the module temperature is measured with a PT100 temperature sensor attached to the back of the module.

Using these measurements a power matrix, *i.e.*, P_{max} as a function of module temperature and irradiance, is created. Onto the power matrix a surface is fitted (see figure 24), called the performance surface, using a three-dimensional data-analysis software[5]. The fitting function is chosen arbitrarily, as a compromise between good fit and a reasonable number of parameters (usually 3-6 parameters are used); hence the function has no physical meaning. This function then serves as the empirical equation for maximum power, that is used in predicting the performance of the module in energy rating (see the following section and section 5.4). [16, 22]

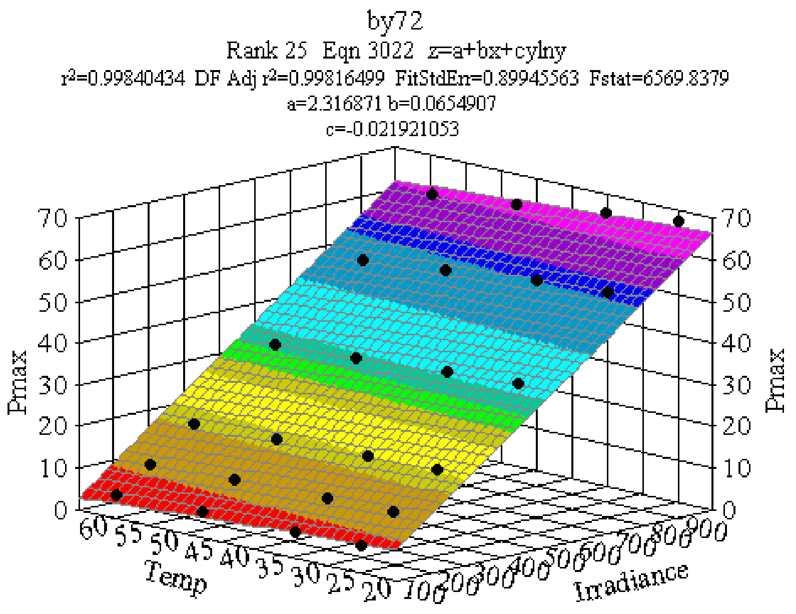


Figure 24: The performance surface for module by72. The fitting equation and the numerical values of its coefficients, and the goodness of the fit are also indicated.

[5]SYSTAT Software Inc., TableCurve3D (2002), `http://www.systat.com/products/TableCurve3D/` (5/2008)

4.4 Energy rating

There are great differences in energy yields of different PV module technologies, and a certain module type might suit to certain environmental conditions better than some other module. The choice of the module type is of utmost importance when planning a PV-based electricity production plant – not forgetting the financial motives: even a 1% difference in energy production can be worth of 4-6 euros per kW$_p$ per year [13]. Energy rating is developed to provide a more reliable and exact tool for module comparison than the module STC power used thus far. It stands for predicting module performance at a certain location using a performance surface that gives the module output as a function of environmental parameters for the location. In this section, some methods used in energy rating are introduced, focusing on the method being developed at ESTI. Finally, the validity of the ESTI method for both c-Si and thin film modules is evaluated. [13]

4.4.1 Energy rating methods

There are several methods proposed for the energy rating, and the objective of the testing laboratories and the standardization organizations is to standardize the most suitable method – a draft of IEC International Standard 61853, "Performance Testing and Energy Rating of Terrestrial Photovoltaic (PV) Modules", is under development. The methods differ on how the data is acquired for generating the power matrix and what kind of mathematical models are used for fitting the surface [13, 23].

In some proposals, the data for the power matrix is acquired using outdoor measurement data, which has given good results in predicting module performances [23]. In this method, however, a measurement period of approximately one year is required to cover sufficient range of irradiance and temperature conditions, and the resulting performance surface is specific for the location where it has been measured. The ESTI energy rating method employs the performance surface generated indoors, in which case only a few hours are needed to acquire a sufficient range of data. [22]

The performance surface determined from the indoor measurements in the ESTI method is a function of irradiance and module temperature. As the prediction is based on using ambient temperature and irradiance data of a specific location, a transformation from ambient to module temperature is required. The module temperature is estimated as follows [22]:

$$T_{mod} = \frac{NOCT - T_{ref}}{G_{ref}} G + T_{amb}, \tag{24}$$

where $NOCT$ is the Nominal Operating Cell Temperature, that is by definition the module temperature at ambient temperature of $T_{ref} = 20$ °C and irradiance of $G_{ref} = 800$ W/m^2. The method for measuring the NOCT, and also the expression above, are described in the standard CEI/IEC 61215: "Crystalline silicon terrestrial photovoltaic (PV) modules – Design qualification and type approval". [22]

As already mentioned, only ambient temperature and irradiance data from the location in question are required as input parameters in the ESTI energy rating method. This is one of the greatest advantages of the method, since historical records for the data is readily available for many locations. This however retains the assumption that the module performance is dominated by these two parameters – for instance the effect of spectral variations and relation of direct and diffuse radiation are neglected. [22]

4.4.2 Evaluating the ESTI energy rating method

To verify the ESTI energy rating method a comparison between the predicted and actual outdoor measured energy production is made. The predicted and measured energy production values are calculated by integration from the respective instantaneous P_{max} -values received either from the performance surface or from the outdoor measurements (see section 5 for details). For crystalline silicon modules, the results have been good: in some studies, the difference between the predicted and actual energy production has been less than 1% [22]. For thin film modules, the predictions have not been as successful [21]. Similar results were obtained in this study (see section 6).

The main reason for the poor success in energy rating of thin film modules is that their indoor performance differs from their outdoor performance; and once the performance surface used in energy rating is measured indoors, this certainly affects the prediction. The discrepancy between the outdoor and indoor performance results primarily from the instability of some thin film materials in relation to the pre-measurement conditions [24]. At ESTI, modules are kept in the dark prior to indoor measurements, and this leads to a reduction in the performance for some thin film technologies, in particular for CIS modules. While being measured outdoors, however, the modules are continuously under light exposure and there is no such effect. One possible solution for this problem is light-soaking, which means preconditioning the modules by exposing them to light (20 min under intensity of 800 W/m^2 and temperature of 40 °C) prior to indoor measurements [24].

Another factor distorting the indoor and outdoor performance of thin film modules and thus making the energy rating more difficult is the effect of spectral variations. This covers differences between the spectral responses of the PV device and the reference device, and differences between the AM1.5G reference spectrum used in indoor measurements and the actual spectrum, that varies in the course of each day. Thus, employing only total irradiance and module temperature as an input in the ESTI energy rating method might be insufficient: for thin film modules, a more sophisticated method considering also the effect of spectral variations might be necessary. [21]

5 Creating an application for outdoor data analysis

The outdoor measurement system executes a performance measurement for the modules at regular intervals (usually every four minutes) throughout each day, as long as the solar irradiance is above 50 W/m^2 on the plane of the modules. On a good day irradiance may be above this limit from 7 a.m. until 8 p.m., and up to five modules are usually measured simultaneously. Hence a vast amount of data is generated in a short period.

To be able to make module performance analysis over long time periods – several months or years – proper data storage is of utmost importance. When situated in a database, all the data is found in the same place in a well-defined order and it is easy to retrieve arbitrary datasets. To analyze the data, a dedicated program was written with MATLAB by the author. MATLAB was chosen for this task because of its flexibility and versatility. This chapter discusses the database system used in our application, the basic features of MATLAB, the interface required between the two, and finally the actual program written with MATLAB.

5.1 The database system

Put simply, a database is a stored set of logically interrelated data. The database system used in our setup was a *relational database*, in which the data is stored in separate tables. Each table has a certain number of fields, and the data is entered, retrieved and modified by rows, that are called records of the table. Each row has an equal number of fields. Variables of a field share certain common features, such as name and data type. The data type can be for instance integer, double precision number, character string or date or time, to mention but a few. [25, 26, chapter 3]

Figure 25 illustrates the structure and contents of our database, named *solarec*. There was one table for each module, and each table contained the same 29 well-defined fields. The data originated from the three output files of the outdoor measurements (see section 4.2), and the data was transferred to the database by Thomas Huld (researcher at the JRC), who maintained the database.

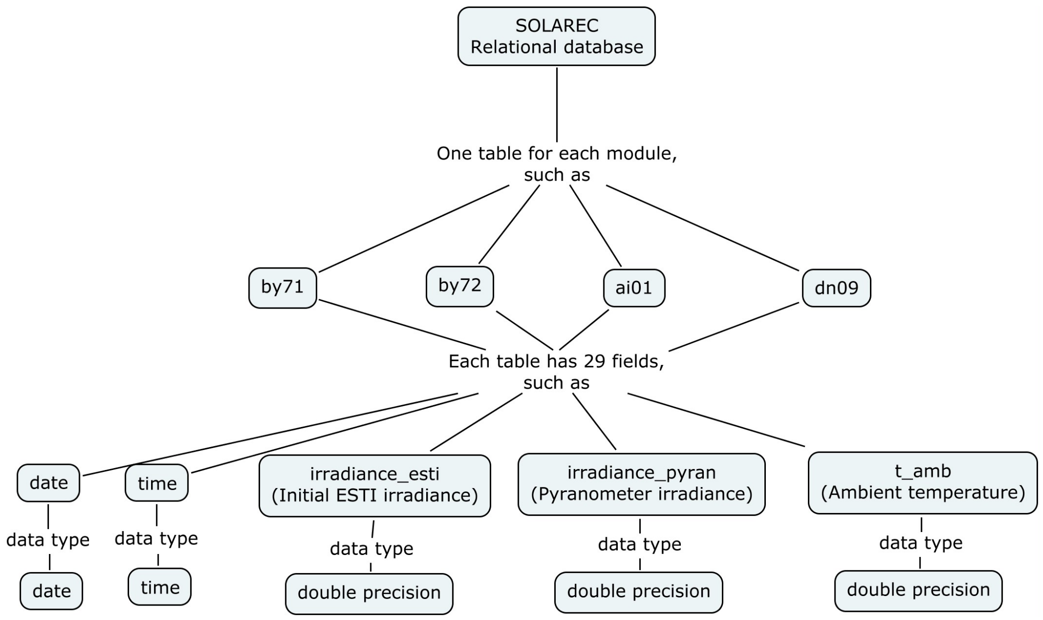

SOLAREC
Relational database
One table for each module, such as
by71
by72
ai01
dn09
Each table has 29 fields, such as
date
time
irradiance_esti (Initial ESTI irradiance)
irradiance_pyran (Pyranometer irradiance)
t_amb (Ambient temperature)
data type
date
time
double precision
double precision
double precision

Figure 25: A block diagram of the structure of the database *solarec*. A complete list of the fields found in each table can be found in appendix A.

A database is created and modified with a database management system. In our setup the management system used was PostgreSQL [27], which is an open source Object-Relational Database Management System (ORDBMS). It is an extension of the more traditional Relational Database Management Systems (RDBMS). As enhancements to the straight relational model, the ORDBMS supports such things as arrays (multiple values in a single column), inheritance (child-parent relationships between tables) and functions (programmatic methods invoked by SQL statements). [26, chapter 3]

Interaction with the database, *i.e.*, adding, retrieving, modifying and removing records of data, takes place via a specific interface using query commands in Strutured Query Language (SQL). The interfaces utilized here for communication with the database were *pgAdmin*, that is an open source adiminstration and development platform for PostgreSQL[6], and a dedicated interface for MATLAB, discussed in section 5.3. The first-mentioned was used mainly to check from which modules and time periods there were data in the database.

The structured query language (SQL) consists of structured statements that resemble simple sentences in English. At the beginning of a statement there is always a command, that is the verb of the statement, describing the action to be taken. In addition this statement usually contains one or more clauses, that are formal modifiers that further define the action and hence the function of the statement. A very simple example of an SQL statement is

```
SELECT date, pmax FROM by71
```

which selects the fields `date` and `pmax` from the table `by71`. [26, chapter 3]

5.2 Introduction to MATLAB

MATLAB is a high performance technical computing environment created by The MathWorks Inc. that was founded in 1984 [28]. Currently the MATLAB system involves five main sections[7] [29]:

[6]pgAdmin III, `http://www.pgadmin.org/` (6/2008)

[7]The MATLAB version used here is 7.4.0.287 (R2007 a), released in January 2007.

- Desktop tools and development environment, which is a set of tools providing help for using MATLAB functions and dealing with the system in general. Most of them are graphical user interfaces – such as MATLAB desktop, that is illustrated in figure 26.

- The MATLAB mathematical function library containing both elementary and advanced functions.

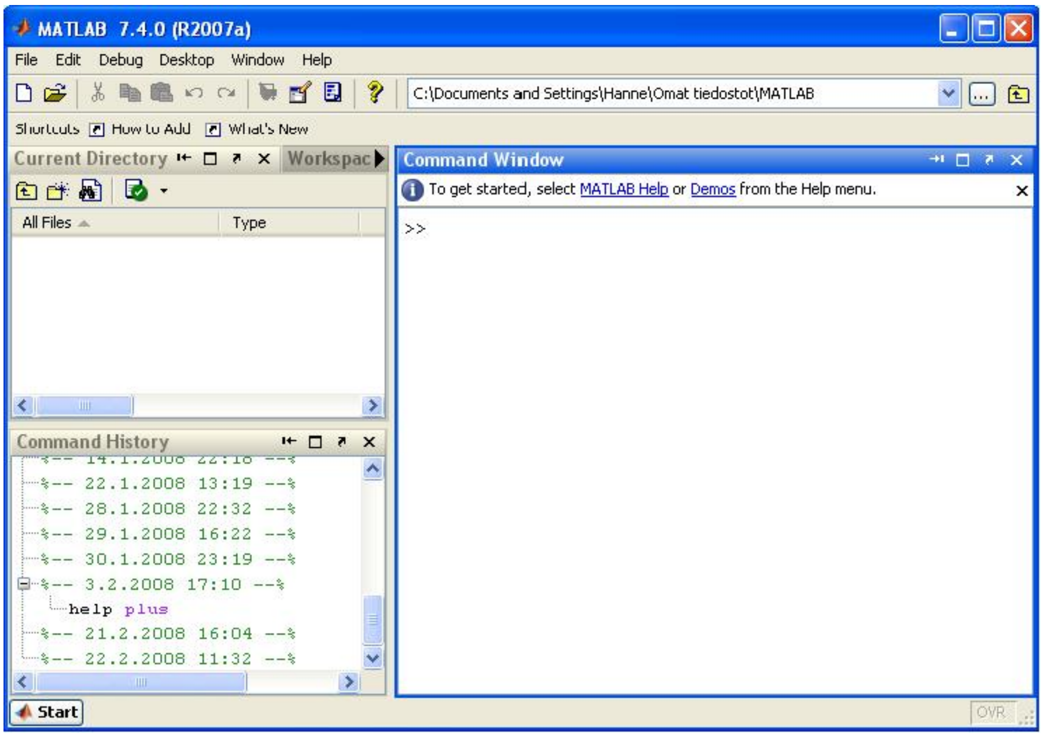
- MATLAB language, which is a high-level matrix/array programming language.

- A wide range of graphics tools for making highly customized 2D an 3D graphs.

- The MATLAB external interfaces/API (Application Program Interface) enabling the user to write C or Fortran routines that interact with MATLAB.

To perform computations with MATLAB, the user writes the task using MATLAB statements and built-in functions into an *m-file* – a text-file with the extension *.m* – which is then executed with a single command. MATLAB provides its own m-file editor, which simplifies the writing and evaluation of the code. Simple calculations can be also performed directly from the MATLAB command window.

M-files are divided according to their contents into two groups: *scripts* and *functions*. A script simply excutes a series of MATLAB statements without accepting inputs or returning outputs, and stores its variables into a workspace common with other scripts and the MATLAB command line interface. A function in turn accepts input arguments and returns output arguments, and stores variables into its own internal workspace – a feature


MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
C:\Documents and Settings\Hanne\Omat tiedostot\MATLAB
Shortcuts How to Add What's New
Current Directory
Workspac
All Files
Type
Command History
Command Window
To get started, select MATLAB Help or Demos from the Help menu.
Start
OVR

```
>>
```

```
%-- 14.1.2008 22:18 --%
%-- 22.1.2008 13:19 --%
%-- 28.1.2008 22:32 --%
%-- 29.1.2008 16:22 --%
%-- 30.1.2008 23:19 --%
%-- 3.2.2008 17:10 --%
   help plus
%-- 21.2.2008 16:04 --%
%-- 22.2.2008 11:32 --%
```

Figure 26: The MATLAB desktop.

that makes using functions favorable when program efficiency is important. In the program described in section 5.4, two main scripts were written to perform the analysis, and these scripts employed several functions. [29]

Another fact to keep in mind while improving the performance of a MATLAB program is that MATLAB was originally written to work with matrix software – its basic data element is an array[8]. For instance minimizing the number of `for` and `while` loops by using approporiate vector and matrix operations instead improves the performance of the code significantly. [29]

[8]The name MATLAB actually comes from MATrix LABoratory.

5.3 Interface between MATLAB and the database

In order to connect to the database and retrieve data from it, certain tools are required. In our application, these tools were provided by MATLAB. PostgreSQL cannot run functions written with MATLAB, but it does support several other programming languages, such as Java, Python and C [27]. MATLAB in turn cannot directly run functions written in other programming languages; therefore an interface is needed for MATLAB to be able to interact with the database.

MATLAB provides routines to execute C or Fortran subroutines in the form of *MEX-files* (MATLAB Executable), which are produced from C or Fortran source code, but which MATLAB can directly load and execute as if they were built-in MATLAB functions. The source code for a MEX-file consists of two parts: the computational routine, containing the code that performs the desired operations (in our case accessing the database) and a gateway routine that interfaces MATLAB with the computational routine. [29]

Suitable MEX-files were found on the internet rather than being built by the team. An engineering organization DerTech[9] had constructed a MEX-file library (called *pgmex*) that provided an interface for a PostgreSQL C library (called *libpq*); these libraries together provided the required access to the database. To use the libraries from MATLAB, the MEX-file library was saved into a new toolbox directory in MATLAB, and the C library was saved into the directory where Windows assumed these types of libraries to be saved.

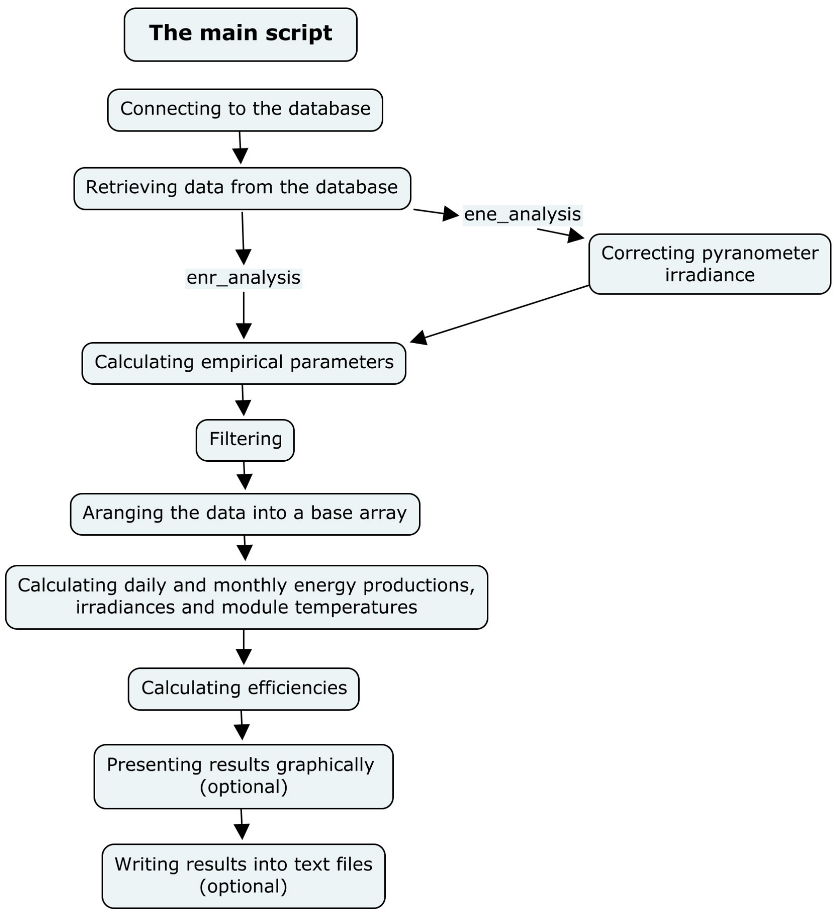
[9]DerTech, LLC, `http://www.dertech.com` (1/2008)

5.4 The MATLAB program

The main tasks set for our MATLAB program are to calculate the measured and predicted energy productions and efficiencies on a daily and monthly basis for different PV modules, using outdoor measurement data located in the *solarec* database. The predictions are based on the empirical equation for maximum power (P_{max}), received from the indoor performance surface measurements (see section 4.3). Before the actual calculations, the data retrieved from the database is filtered in order to eliminate faulty data points. In the end, the program presents the results graphically and fromulates them into text files.

The starting point for the program was another MATLAB program written by Susana Iglesias (an earlier trainee at the JRC). The main tasks for this former program were essentially the same though it was somewhat restricted: it retrieved the data for analysis from text files, and was restricted to analyzing data from only three different crystalline silicon modules one at a time. Comparatively, the new program is able to analyze data from whichever modules have data in the database, simultaneously when necessary. Additionally, the former program did not include any filters.

The program encompasses two main scripts that employ several dedicated functions. The structure of the program is illustrated in figure 27. Most of the tasks presented in this figure are executed in separate functions instead of the main scripts, as this clarifies the scripts and above all improves the program performance significantly. The two main scripts are named `ene_analysis` and `enr_analysis` (appendices B.1 and C.1), written for analyzing data from ene-files and enr-files, respectively (see section 4.2). The tasks performed by these scripts are slightly different, and obviously they analyze different data, but they mainly employ the same functions. Following the structure in the figure 27, the following sections describe in more detail the contents of the program: what the scripts actually do and why, what functions they employ and how the program communicates with the user at each of the stages shown in the figure.



The main script
Connecting to the database
Retrieving data from the database
ene_analysis
Correcting pyranometer irradiance
enr_analysis
Calculating empirical parameters
Filtering
Aranging the data into a base array
Calculating daily and monthly energy productions, irradiances and module temperatures
Calculating efficiencies
Presenting results graphically (optional)
Writing results into text files (optional)

Figure 27: The structure of the MATLAB program.

Connecting to the database

The first task for both main scripts is to create a connection to the database, using the above mentioned MEX-files as an interface. The actual interface function used to make the connection from within MATLAB is called pgconnectdb, taking as an input a string connstr containing name, host and user of the database as follows:

```
connstr = 'host=emu.jrc.it dbname=solarec user=thomas';
myconn = pqconnectdb(connstr);
mystat = pqstatus(myconn)
```

If the connection to the database is created successfully, the interface function pqstatus returns CONNECTION_OK and mystat = 0 to the command window of MATLAB. This code snippet is found at the beginning of both main scripts (appendices B.1 and C.1).

Retrieving data from the database

Once the connection between MATLAB and the database is enabled the program uses the received variable myconn to retrieve the required data from the database. This is performed either by function getdata_db_ene or getdata_db_enr (appendices B.2 and C.2), depending on the script being executed (ene_analysis or enr_analysis, respecively).

The actual parameters that are retrieved for the analysis are fixed. The function getdata_db_ene retrieves for instance date, time, corrected P_{max} and ESTI irradiance, change in ESTI irradiance and ambient and module temperatures; the function getdata_db_enr in turn retrieves date and time, the uncorrected P_{max} and ESTI and pyranometer irradiances and so forth. However, the time span (the first and the last date) as well as the number and names of the modules to be analyzed are to be defined by the user. In this task MATLAB built-in function input is utilized: input prints the defined question to the command window and waits for input from the user. The functions subsequently build a string exeStr that contains an

SQL statement for retrieving the fixed parameters according to the limits set by the user. To construct the string, MATLAB built-in function `sprintf` is employed. The string is then used by the interface function `pqexec` to access the data. The following loop of the function `getdata_db_enr` (appendix C.2) illustrates how this happens for each of the `fN` modules being analyzed.

```
for i=1:fN
   name = ['Name of the module ' num2str(i) '? '];
   fmod{i} = input(name,'s'); % Cell array for names of the modules
   exeStr = sprintf('SELECT date, time, pmax, irradiance_esti,
            irradiance_pyran, t_mod, t_amb
            FROM %s
            WHERE date≥''%s'' AND date≤''%s''
            ORDER BY date,time',
            fmod{i},Dstart,Dend);
   res(i) = pqexec(myconn, exeStr);
   nFields(i) = pqnfields(res(i));
   nRows(i) = pqntuples(res(i));
end
```

The variable `res` received from `pqexec` is used later in the function to retrieve the data for each module row by row. The size of the dataset (number of fields and rows of data, `nFields` and `nRows`) retrieved for each module is displayed to the command window of MATLAB. The number of columns is always identical for all the modules, since the same parameters are retrieved for all the modules. The number of rows usually varies as different modules are not measured for exactly the same time periods.

Correcting pyranometer irradiance

The next task to perform within the script `ene_analysis` is to calculate a corrected value for the pyranometer irradiance, employing the function `pyran_correction` (appendix B.3). The motivation for calculating a corrected pyranometer value was interest in seeing if it makes a difference whether the energy predictions and efficiencies are calculated using corrected ESTI or pyranometer irradiances. There had been no previous attempts to make this kind of correction.

The corrected pyranometer value is calculated only for measurement points in which the change in irradiance measured with the ESTI sensor is less than 5%. This threshold value was chosen after examining the irradiance and $I - V$ curves of different measurement points of one module and studying the number of measurement points discarded with different treshold values. With higher threshold values the change in irradiance was so big and the $I - V$ curve was so distorted that correcting the pyranometer value was not reasonabe, and with lower threshold values there were simply too many measurement points discarded. Figure 28 shows the relative amount of measurement points deleted with different threshold values for module ai01 for each month in 2006.

The correction to the pyranometer irradiance is made using the average irradiance values of the measurement scan for both ESTI and pyranometer irradiances, $G_{pyr,ave}$ and $G_{ESTI,ave}$ (W/m^2) (see section 4.2.3), and the minimum value of the scan for ESTI irradiance ($G_{ESTI,min}$) as follows:

$$G_{pyr,corr} = \frac{G_{pyr,ave}}{G_{ESTI,ave}} G_{ESTI,min}.$$

This correction is made only for measurement points having less than 2% change in ESTI irradiance. For measurement points with a change of 2-5%

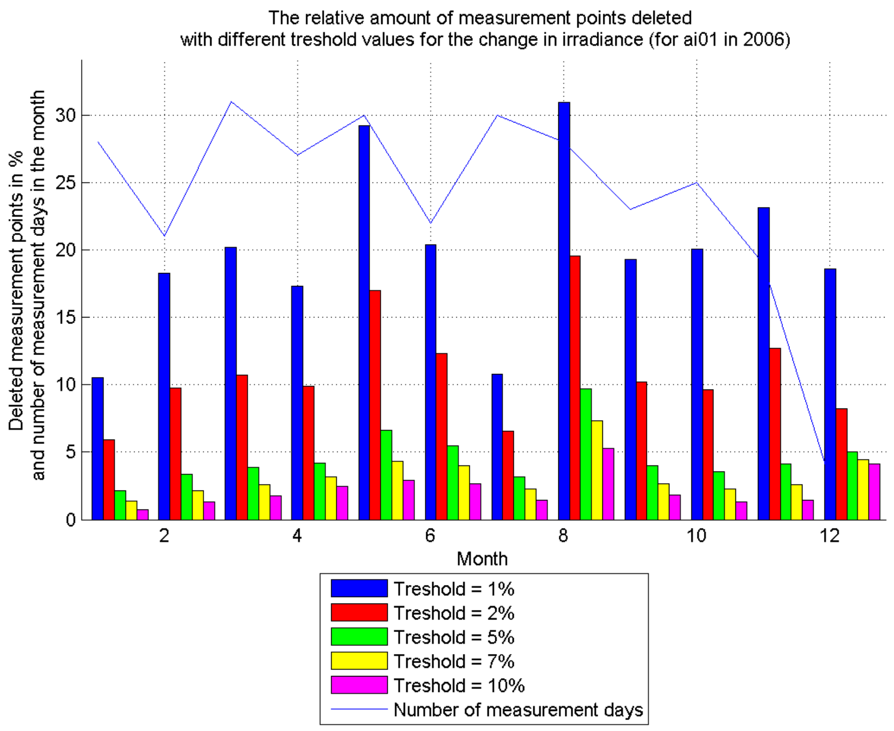


Figure 28: Relative amount of measurement points filtered out with different threshold values for the relative change in ESTI irradiance for module ai01 for each month in 2006. The number of measurement days in each month is also indicated.

the function searches for a measurement point with an irradiance change of less than 2% within 6 steps from the measurement point considered. If it finds such a point, it performs the correction using the $G_{pyr,ave}$ and $G_{ESTI,ave}$ values of the new point, and the $G_{ESTI,min}$ value of the original point. The remaining points are set to zero. After the corrections the function displays for each module being analyzed a histogram of the number of steps required for a point having an irradiance change of 2-5% to find a point with less than 2% change.

This task is not performed within the script `enr_analysis`, since the ESTI and pyranometer irradiance values used by this script are uncorrected.

Calculating empirical parameters

The next task in both scripts is to calculate the empirical parameters for comparing the measured and predicted results. At first an empirical module temperature ($T_{mod,emp}$) is calculated using ambient temperature and irradiance data in order to perform predictions from the basis of these parameters only. For this purpose function `noct_estim` (appendix D.1), employed by both main scripts, was written.

The function first defines the value for NOCT (Nominal Operating Cell Temperature) according to equation (24) introduced in section 4.4. The function plots the difference of measured ambient and module temperatures (T_{amb} and T_{mod}) as a function irradiance, and constructs a linear fit to this set of points using MATLAB built-in function `polyfit`. The plots – an example is shown in figure 29 – are displayed to the user. According to the equation (24) the expression for the linear fit becomes

$$T_{mod} - T_{amb} = \frac{NOCT - T_{ref}}{G_{ref}} G, \tag{25}$$

where $T_{ref} = 20$ °C and $G_{ref} = 800$ W/m^2. The NOCT can thus be calculated from the slope of the fit. The empirical module temperatures are thereafter calculated using the same equation with the defined NOCT, and irradiance and ambient temperature data. The value for NOCT is usually around $40 - 50$ °C [22, 4].

The function `polyfit` used in the fitting is based on the least-squares method, and therefore in the actual calculations the fit does not necessarily intercept the origin as it is should according to the equation (25) above.

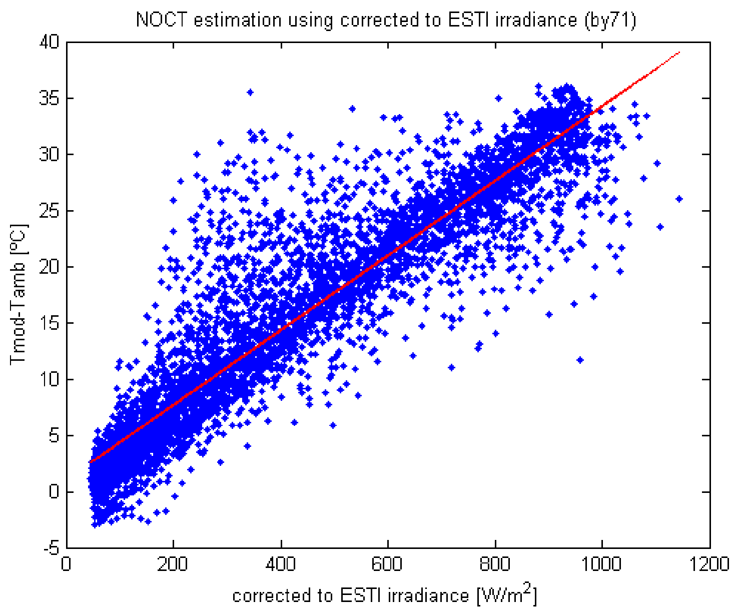


Figure 29: Plot of the difference of ambient and module temperatures as a function of ESTI irradiance for estimating the NOCT for module by71. The NOCT can be calculated from the slope of the straight line according to equation (25).

This is also seen in figure 29, and it is undoubtedly a source of error in the calculations. In the results (see section 6), however, it is seen that the module temperature estimations based on this method are good.

With the empirical module temperature having been calculated, the scripts subsequently calculate the values for empirical P_{max}, using the empirical equation for $P_{max}(G, T_{mod})$ received from indoor performance surface measurements. Altogether four different empirical maximum powers are calculated:

- `Pmax_emp_es`, calculated using the ESTI irradiance and measured module temperature

- `Pmax_empT_es`, calculated using the ESTI irradiance and empirical module temperature
- `Pmax_emp_py`, calculated using the pyranometer irradiance and measured module temperature
- `Pmax_empT_py`, calculated using the pyranometer irradiance and empirical module temperature.

The script `ene_analysis` uses corrected and `enr_analysis` uses uncorrected irradiance values. The scripts read the empirical equation for P_{max} for each module from a text file **`empirical_data.txt`** (appendix E), located in the same directory as the scripts. To read the file, MATLAB built-in function `textscan` is employed. In addition to the equation and its parameters, different measured STC powers and the surface area of the module are also retrieved from and therefore need to be written into the text file. The STC powers are later used to calculate normalized energy productions in order to perform module comparison (see page 82), and the module area is used in efficiency calculations.

Filtering

At this point all the required data is available for filtering. The scope of filtering is to remove defective measurement points from the dataset, originating for example from errors in the measurement system. The functions written to perform the filtering are `data_filter_ene` and `data_filter_enr` (appendices B.4 and C.3); the former is employed by `ene_analysis` and the latter by `enr_analysis`. The former function filters out, *i.e.*, sets to zero, measurement points if some of the following occurs:

1. The measured P_{max} is zero.

2. The measured P_{max} is negative.

3. One of the empirical P_{max} values listed in previous section is either negative or complex. This may occur when module temperatures are low (near 0 °C), as the performance surface does not cover low temperatures and the fitted performance equation sometimes does not behave well for extrapolated data.

4. The relative difference between measured and empirical P_{max} is more than 50%.

5. The ESTI irradiance has changed by more than 5% (or another threshold value set by the user) during the measurement.

6. The measured P_{max} is bigger than any of the STC P_{max} values.

In `data_filter_enr`, the filtering is performed similarly except for the fifth point, which is replaced with a filter taking out points in which the relative difference between ESTI and pyranometer irradiances is more than 30%.

The measurement points that are set to zero in the filtering are all the P_{max} values (measured and empirical ones) and both irradiances. The filtered measurement points are flagged in a separate array; the value of an element in the array tells the reason why the point has been filtered. The number of discarded points on each day and month are later calculated using this array, and the results are written in the same text files as the other results in order to evaluate how much data has been discarded and for what reasons.

Arranging the data into a base array

As explained in section 4.2, the outdoor measurement system performs all measurements, scanning the $I-V$ curve and recording all the parameters, at regular intervals through each day. An interval divisible into 60 minutes (typically four minutes) is always chosen, and the measurement times are synchronized with the beginning of the hour – thus the measurement time points of the system are always known when the interval is known. The interval has however not always been the same. For the past couple of years it has been set to 4 minutes, but earlier also intervals of 5 and 3 minutes have been in use.

When several modules are measured simultaneously, which is often the case, the system performs the measurements for the modules successively in a certain order: starting at the well-known measurement point, it performs the measurement for the first module (the duration of a scan for one module is approximately 5 seconds) and then moves to the next module and repeats the measurement for it and so on, up to the last module. The whole procedure is repeated at the next measurement point. Consequently, the actual measurement times written in the output files are slightly different for all the modules being measured. When analyzing several modules simultaneously with the program, a common time stamp corresponding to the original measurement time points of the system has to be created to simplify the computations and to be able to make reliable module comparison. For this task, function `base_matrix` (appendix D.2) was written.

Firstly, the function finds out which measurement interval has been in use in the time period being analyzed. It calculates the mode, *i.e.*, the most frequent, measurement interval of the whole time period, and displays the days in the period that had this interval. The days that had an interval other than the mode interval are also displayed, accompanied with their intervals – when analyzing longer time periods, the measurement interval might have changed in between. At this stage the application asks the user, whether or not to use the mode measurement interval in further calculations. If the measurement interval is not the same for the whole period the analysis should be repeated in shorter periods with approximately equal measurement intervals. There are often also several days with zero measurement interval, since the function takes into account all the days between the starting and ending dates of the time span defined in the beginning, even though there may have been non-measurement days in between.

Table 5: An illustrative table of how the parameters of one module are located into the base array, assuming a measurement interval of four minutes. The actual base array is 3-dimensional – one 2D array for each module. Here only a few parameters are shown, and their values are randomly chosen.

Date	Time	P_{max} [W]	$P_{max,emp,ESTI}$ [W]	G_{ESTI} [W/m^2]	...
01.06.2007	6.00	0	0	0	...
01.06.2007	6.04	0	0	0	
01.06.2007	6.08	0	0	0	
...	...	...	...	...	
01.06.2007	14.00	75	78	755	
01.06.2007	14.04	82	84	790	
...	...	...	...	...	
01.06.2007	19.52	24	25	82	
01.06.2007	19.56	20	22	59	
01.06.2007	20.00	0	0	0	
02.06.2007	6.00	0	0	0	...
02.06.2007	6.04	0	0	0	
...	...	...	...	...	

Once the measurement interval is chosen, the function builds an array where all the possible measurement points between 6 a.m. and 20 p.m. reside with the chosen interval for each day concerned. The data points from each module are then located in their correct places: if a measurement point of a module resides within half the measurement interval from a certain measurement point of the system, the corresponding data points are located in accordingly with this measurement point. Table 5 illustrates the idea of the base array.

Calculating daily and monthly values

The daily and monthly energy productions, irradiances and module temperatures are calculated using the function `energycalc` (appendix D.3). Once ordered into the base array, where the number of measurement points within a day is fixed, it is easy to perform calculations for the data on a daily basis using appropriate matrix operations. The parameters to be evaluated are extracted from the base array and reshaped into arrays having one column for each day using the MATLAB built-in function `reshape`. For instance the measured P_{max} is extracted from the base array and arranged to a new 3D array in the following manner:

```
Pmax = reshape(base_arr(:,2,:),M_d,nDays,N);
```

The resulting 3D array has as many rows as there are measurement points within a day (`M_d`), as many columns as there are days (`nDays`) and the number of modules being analyzed (`N`) as the third dimension. The daily measured and empirical energy productions are thereafter calculated from the corresponding maximum power values (P_{max}, W) and the measurement interval dt (h) defined earlier with by summation:

$$E = \sum_i P_{max,i} dt,$$

where E is the daily energy production value (Wh). Finally, the monthly energy production values are calculated by summing the daily values.

In addition to the energy productions, the function `energycalc` similarly calculates the daily and monthly average module temperatures and daily and monthly irradiations. The number of points filtered for different reasons and the total number of measurement points are also calculated for each day and month.

The daily values are stored into 4D arrays – one 3D array for each module – for later use. The 3D arrays, illustrated in figure 30, have 12 rows (one for each month) and 31 columns (one for each day of the month); the size of the third dimension depends on the number of years being analyzed. For simplicity the number of columns and rows is fixed although all months do not have 31 days and there are not necessarily measurements from all of the 12 months – thus there is usually many zeros in these 4D arrays. The monthly values in turn are stored into 3D arrays having 12 rows, the number of columns equal to the number of years, and the size of the third dimension equal to the number of modules being analyzed.

Calculating efficiencies

For calculating the measured and empirical efficiencies for each measurement point, day and month, function `effcalc` was written. The function uses the irradiances, and measured and empirical maximum powers and energy productions calculated earlier. The efficiencies are calculated as

$$\text{Eff}(\%) = \frac{E_{module}}{E_{sun} \cdot A_{module}} \cdot 100\%, \tag{26}$$

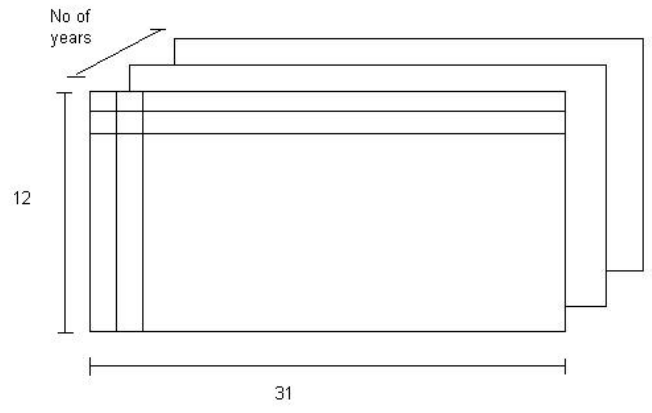


Figure 30: The dimensions of the 3D array used to store the different daily values (energy productions, irradiations, average module temperatures etc.).

where E_{module} is the energy or power produced by the module, E_{sun} is the energy or power coming from the sun per unit area and A_{module} is the surface area of the module. A_{module} was retrieved earlier from the text file `empirical_data.txt` (see page 74). The daily and monthly efficiencies are stored into 4D and 3D arrays similar to the ones introduced in the previous section to be used later in plotting the results and writing them into text files.

Presenting the results graphically

Three different functions were written to make three types of graphs. Both main scripts employ these same functions. The graphs are optional for the user: utilizing the MATLAB built-in function `input`, the user is asked at each point if he or she wishes to construct certain types of graphs. Examples of the graphs are presented in section 6.

The first function, `plotter` (appendix D.4), uses the 4D and 3D arrays built earlier to construct bar plots where measured energy productions and efficiencies are compared with the predicted ones on a daily and monthly basis. This is performed for each module separately, using both irradiances. In addition to these graphs, the function also plots daily average energy productions and so called monthly equalized energy productions for each month. The daily average values are calculated by dividing the monthly total output with the number of *measurement* days in the month, and the monthly equalized values are monthly outputs calculated by multiplying the daily average value of each month with the *total* number of days in the month. The purpose for plotting the monthly equalized values is to give an estimate for the monthly energy production even if there were several non-measurement days during the month. The program saves all the graphs directly in *png* format into a folder named *plots*, that is created in the same directory with the m-files of the program (if necessary). For clarification, the names under which the different graphs are saved are displayed to the command window.

The second function, `plotter_meas_only` (appendix D.5), also constructs bar plots on a daily and monthly basis, however, only for measured values: for the measured energy productions and separately for both irradiances. The energy production values used by this function are not, however, the same as used by the previous function: these values are calculated using maximum powers in which gaps of one or two measurement points, resulting from filtering, are filled with average values of the surrounding measurement points. The purpose of plotting these "filled" energy production values is to find out, how much the filtering has actually affected the calculated energy productions. As earlier, the graphs are saved into the folder *plots*, and the names under which they are saved are displayed to the user.

The third function, `mod_comparison` (appendix D.6), constructs bar plots in which the daily average energy productions of each month of the different modules being analyzed are compared. Monthly irradiances and monthly average module temperatures of the modules are also compared. In order to compare modules of different types and sizes accurately, the daily average energy production values of different modules are compared as normalized to each module's STC P_{max} values (retrieved earlier from the text file **`empirical_data.txt`**). This means that the daily average energy production values of each month and module are divided with each module's STC P_{max}, also called watt-peak (W_p), values. In all these graphs only days when data from all the modules is available are considered in order to make accurate comparisons. Again, the graphs are saved into the folder *plots*, and the names under which they are saved are displayed to the user.

Furthermore, the functions that make plots of daily values – `plotter` and `plotter_meas_only` – use another function called `month_teller` (appendix D.8) which identifies the literal names of different months. The literal names are required in the titles of the plots, and for the file names when saving the plots.

The functions `plotter` and `mod_comparison` construct several different bar plots, each with something in common. To avoid repetition in the code and thus improve the program performance, so called nested functions are used within these functions to make the actual bar plots. A nested function is a function written in the same m-file with another function, and it can be used only by this parent function. For instance in the function `plotter`, the daily and monthly energy productions and efficiencies are plotted, as well as daily average and monthly equalized energy productions. In all of these plots the measured values are compared with the two different empirical values (see section 6). In context with each separate plot type, the function defines the lengths and labels of vertical and horizontal axes, titles etc. and passes them to the nested function called `bar_plotter` to make the actual plots. For more details of this method, see [29] and appendices D.4 and D.6.

Writing the results into text files

As for constructing the graphs, writing the results into text files is also optional for the user. To save the calculated daily and monthly results, both main scripts employ the function `data_writer` (appendix D.7). The 4D and 3D arrays where the daily and monthly values reside are first reshaped into appropriate arrays, and then written into text files using MATLAB built-in function `fprintf`. The text files are saved into a folder *results*, created in the same directory with the m-files of the program (if necessary). The user defines the names with which the text files are saved.

6 Results

This chapter presents results obtained with the MATLAB program described in the previous section. Firstly, the outdoor performances of two different modules, one crystalline Silicon (c-Si) and one thin film Copper Inidium Diselenide (CIS) module, are compared with their predicted performances in order to test the ESTI energy rating method (see section 4.1). The predicted performances have been calculated using the empirical equation for maximum power as a function of incident irradiance and module temperature ($P_{max}(G, T_{mod})$) received for each module from the indoor performance surface measurements (see section 4.3). Predictions based on irradiances measured with both ESTI sensor and pyranometer are presented.

Secondly, the energy yields of three different modules – one c-Si and two CIS – are compared with each other. These modules are shown mounted on the test rack in figure 31.

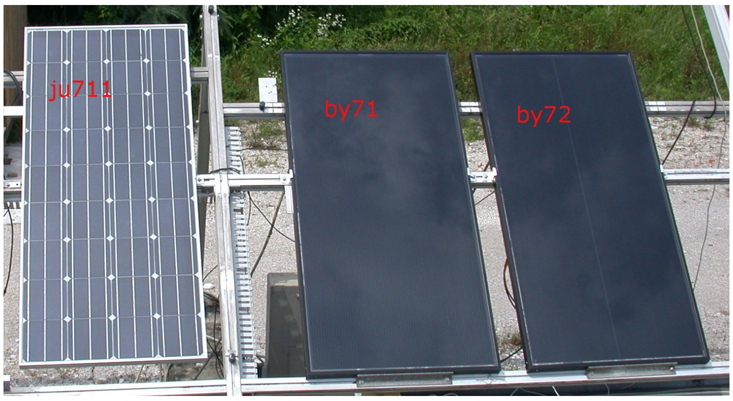


Figure 31: The studied modules: one c-Si module and two CIS modules, denoted by ju711, by71 and by72, respectively.

6.1 Comparing measured and predicted energy productions and efficiencies

6.1.1 C-Si module ju711

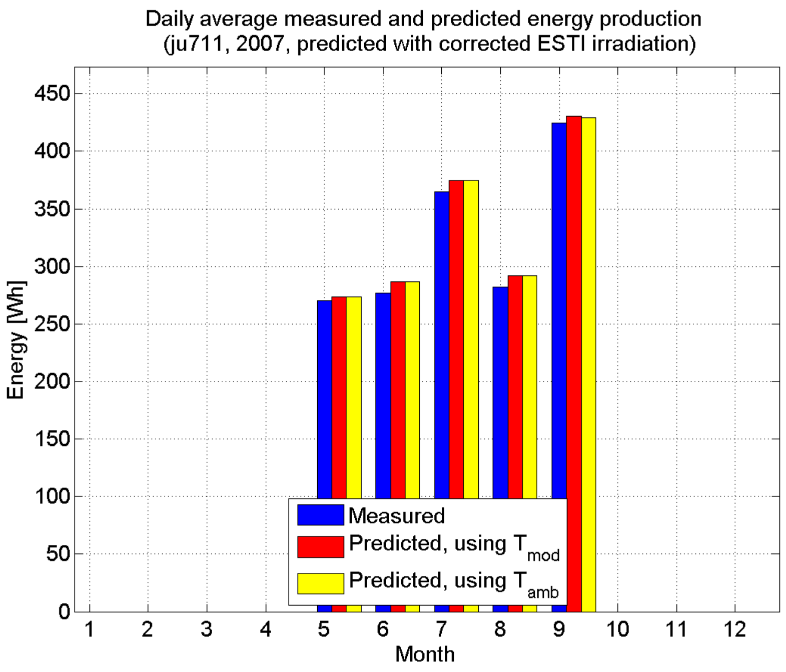
The daily average measured energy productions of the c-Si module ju711 are compared with the predicted ones for months of May to August in 2007 in figure 32. As explained in section 5.4, the daily average energy production is the monthly total energy production divided by the number of measurement days in the month. In the two graphs in figure 32, as well as in the later graphs, there are bars of three different colours representing the three different energy production values:

- The blue bar represents the actual, measured energy production.
- The red bar represents the predicted energy production calculated using measured module temperatures and irradiances.
- The yellow bar represents the predicted energy production calculated using estimated module temperatures and measured irradiances. The estimated module temperatures are calculated according to equation (24) introduced in section 4.4.

In the corresponding graphs for efficiencies, the blue bar represents the measured efficiency, calculated using measured energy production values and certain irradiation values, and the red and yellow bars represent the predicted efficiencies, calculated using the two different predicted energy production values described above and the corresponding irradiation values. The daily and monthly efficiencies are calculated according to the equation (26) introduced in section 5.4, thus by using the input and output *energies* of the module rather than by integrating instantaneous efficiencies over a day or month. For instance, the daily efficiencies are calculated using the daily energy production values of the module and the daily irradiations.

It can be seen from the graphs in the figure 32 that it does not make any significant difference, whether the estimated or measured module temperature is used in the prediction. The differences between the two predicted energy production values is in the order of one per mille in all the studied months in both graphs. Similar results were obtained with all the studied modules.

The difference between the two graphs in figure 32 is that in the upper graph the predicted energy productions are calculated using the corrected ESTI irradiance (see section 4.2.3) and in the lower graph using the corrected pyranometer irradiance. Calculation of the corrected pyranometer irradiance was explained in section 5.4. The predictions based on pyranometer irradiance overestimate the energy production clearly: the predicted values are on average 5% higher than the measured values. The predictions based on esti irradiances in turn are on average only 2% higher than the measured values. This was partially expected: irradiances measured with the pyranometer are in general higher as the pyranometer "sees" a wider portion of the solar spectrum than the ESTI sensor (see section 4.2). However, both devices are calibrated to give the same reading at AM1.5G; still, the pyranometer tends to give higher readings especially in the morning and in the evening when the altitude angle of the sun is small (equivalently, the air mass ratio is big). Furthermore, the module and the ESTI sensor are in this case made from the same material, thus the prediction based on the ESTI sensor is assumed to be more realistic. Nevertheless, it is not self-evident whether the differences between the predictions are due to these reasons only or whether the method for calculating the corrected pyranometer irradiance overestimates



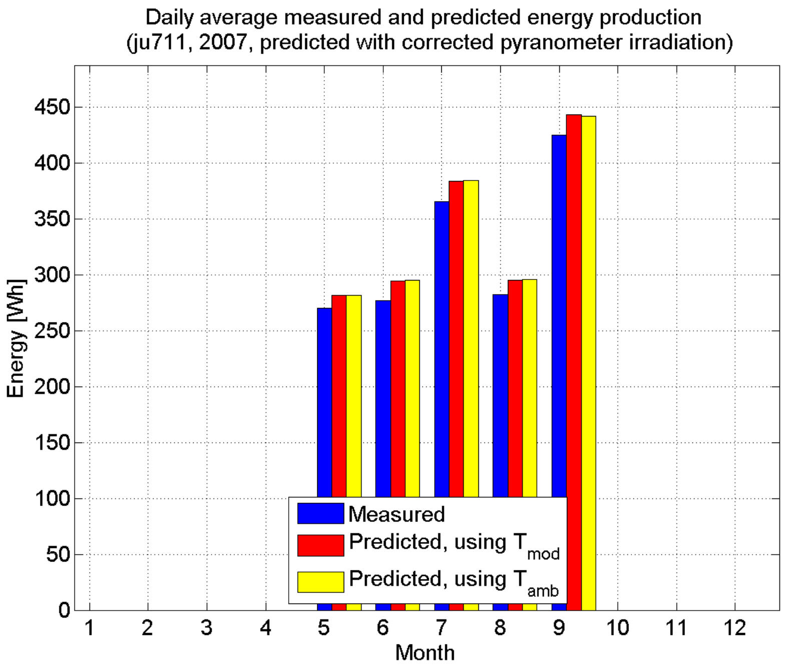


Figure 32: The daily average measured and predicted energy production for ju711 from May to September in 2007. In the upper graph, the predictions are calculated using corrected ESTI irradiance and in the lower graph using corrected pyranometer irradiance.

the values.

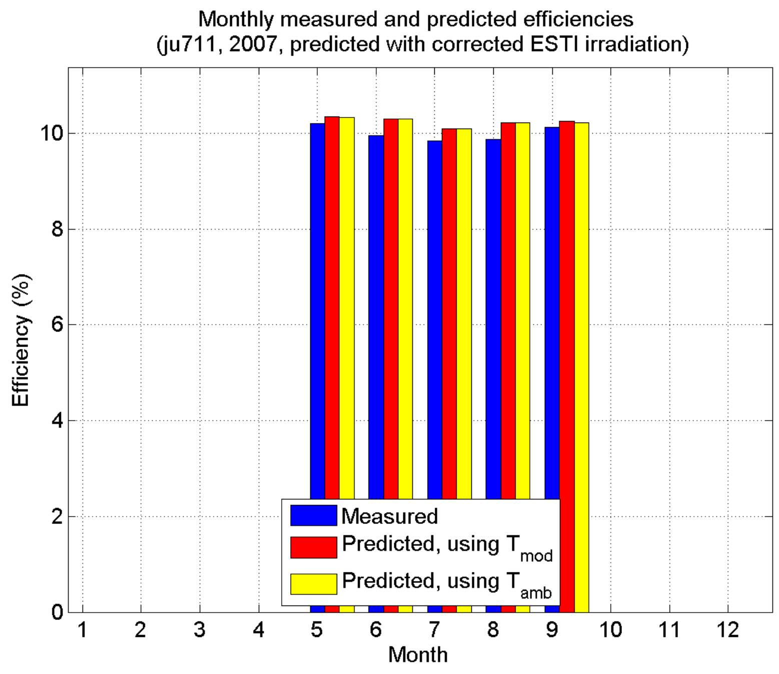
Figure 33 shows, respectively, the monthly efficiencies for ju711. Efficiencies calculated using pyranometer irradiations (lower graph in figure 33) are in general lower than the ones calculated with ESTI irradiations (upper graph in figure 33): measured efficiencies are on average 2% higher when calculated using ESTI irradiations. This makes sense, as the irradiance values measured with the pyranometer are higher as noted previously. Furthermore, the behaviour discovered in the previous graphs is also seen in these graphs: predicted efficiencies based on ESTI irradiance are closer to the measured values than the predictions based on pyranometer irradiance.

Figure 34 presents results for daily actual and predicted energy productions in June 2007. The daily energy production naturally varies significantly due to altering environmental conditions. Again, the predictions based on ESTI irradiance (upper graph in figure 34) are closer to the actual energy production values than the predictions based on pyranometer irradiance (lower graph in figure 34).

6.1.2 CIS module by71

Figures 35-37 show the corresponding results for CIS module by71: the actual and predicted daily average energy productions (figure 35) and monthly efficiencies (figure 36) from May to August, inclusive, in 2007, and the actual and predicted daily energy productions in June 2007 (figure 37). Again, the predictions based on corrected ESTI irradiances and the predictions based on corrected pyranometer irradiances are shown in separate graphs. Results for only one CIS module are presented because the results for the other CIS module mentioned were essentially the same.

In all of these graphs it is seen that for this module the actual outdoor



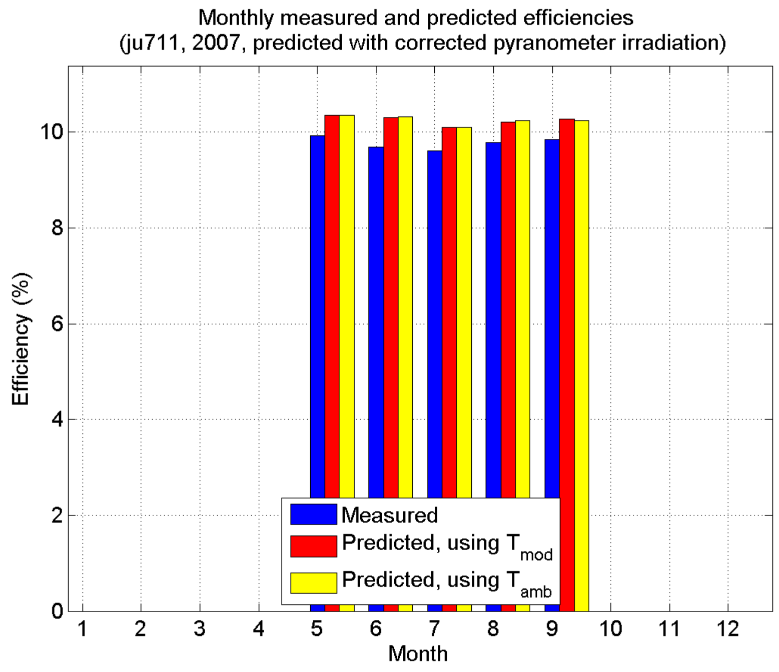
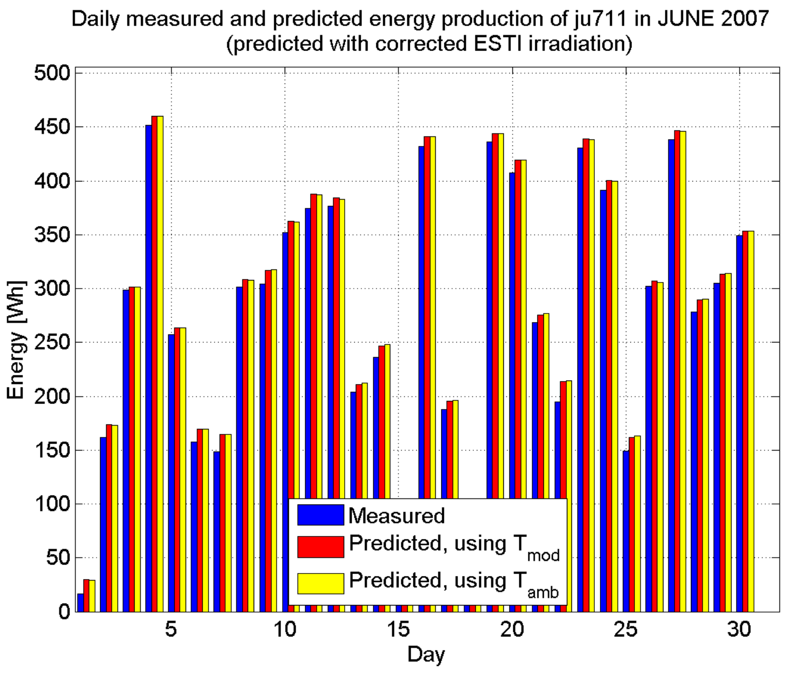


Figure 33: The monthly measured and predicted efficiencies for ju711 from May to September in 2007, calculated using corrected ESTI (upper graph) and pyranometer (lower graph) irradiances.



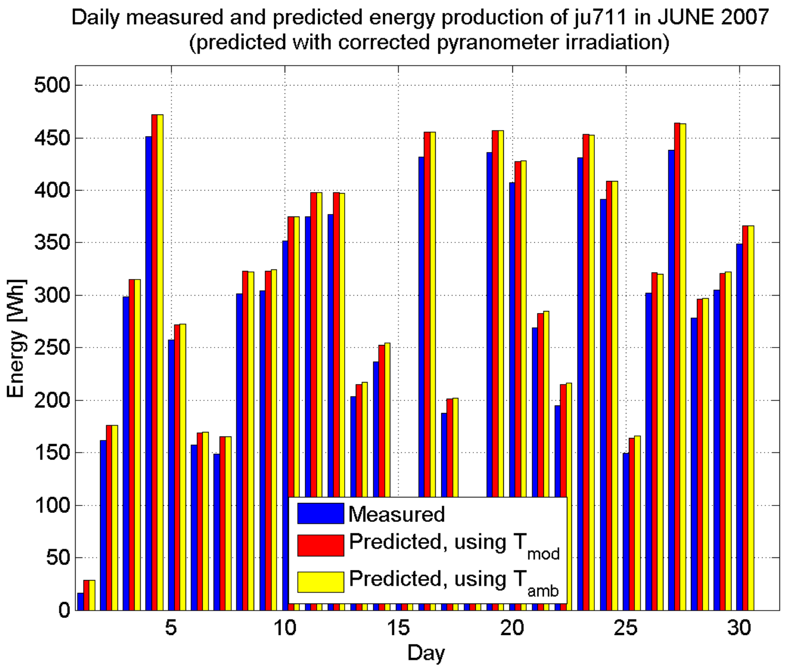
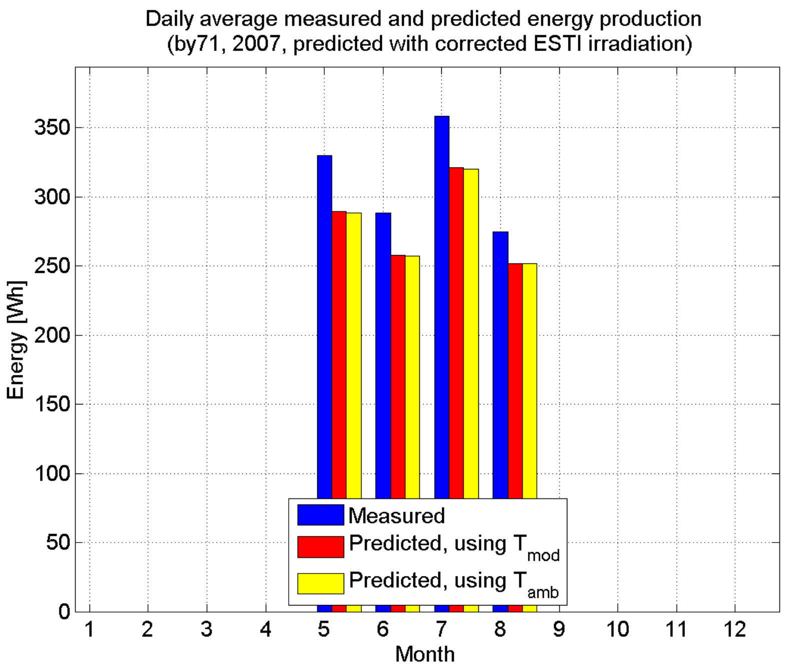


Figure 34: The daily measured and predicted energy production for ju711 in June, 2007. In the upper graph, the predictions are calculated using corrected ESTI irradiance and in the lower graph using corrected pyranometer irradiance.



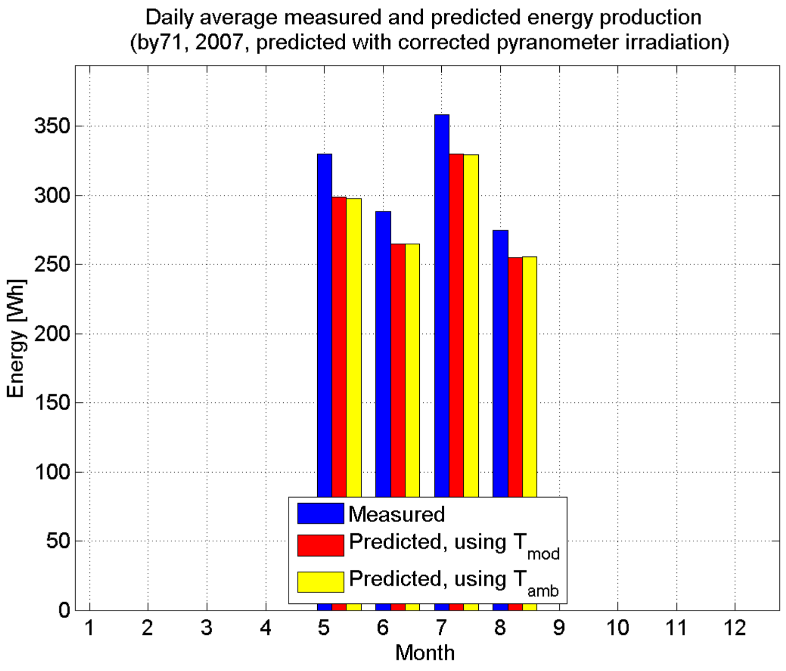


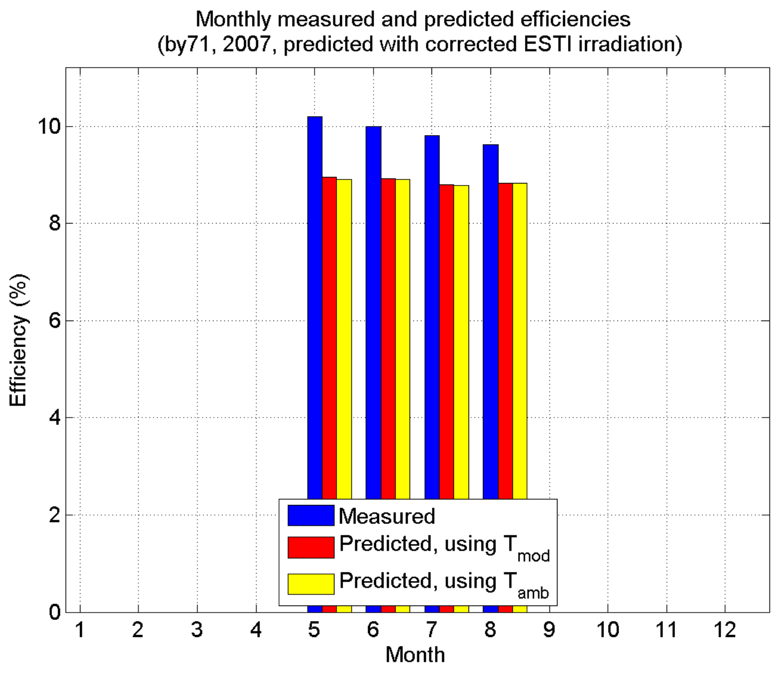
Figure 35: The daily average measured and predicted energy production for by71 from May to August in 2007. In the upper graph, the predictions are calculated using corrected ESTI irradiance and in the lower graph using corrected pyranometer irradiance.

measured performance is much higher than the predicted performance (based on indoor measurements). For the daily average energy production values, the predicted values based on ESTI irradiance are on average 11% higher and the predicted values based on pyranometer irradiance are on average 8% higher than the measured values. The difference being smaller with predictions based on pyranometer irradiance is explained with the higher irradiance values received from the pyranometer, which was discussed already in the previous section. The main reason for such a large difference between the measured and predicted performance is the discrepancy between indoor and outdoor performance of CIS modules, resulting from the effect of different pre-measurement conditions on these modules (see section 4.4). Similar behaviour was seen when data from other CIS modules was analyzed.

6.2 Module comparison

Table 6 displays numerical values for the measured daily average energy productions from May to August inclusive in 2007. Included are the c-Si module ju711 and the CIS module by71 discussed previously, and additionally another CIS module, denoted by by72. In addition to the energy yields, the number of measurement days and relative amount of measurement points discarded in filtering of the data are also indicated in order to see how these parameters affect the differences in the energy yields. The monthly average module temperatures are also given.

According to the table, the CIS module by72 has the highest daily average energy yield in all the months except for August, where the c-Si module has the highest and by72 has the worst output. The c-Si module has clearly the worst performance of all the three modules in May and June, and in July by71 is performing worst, though the results for July are somewhat unreliable



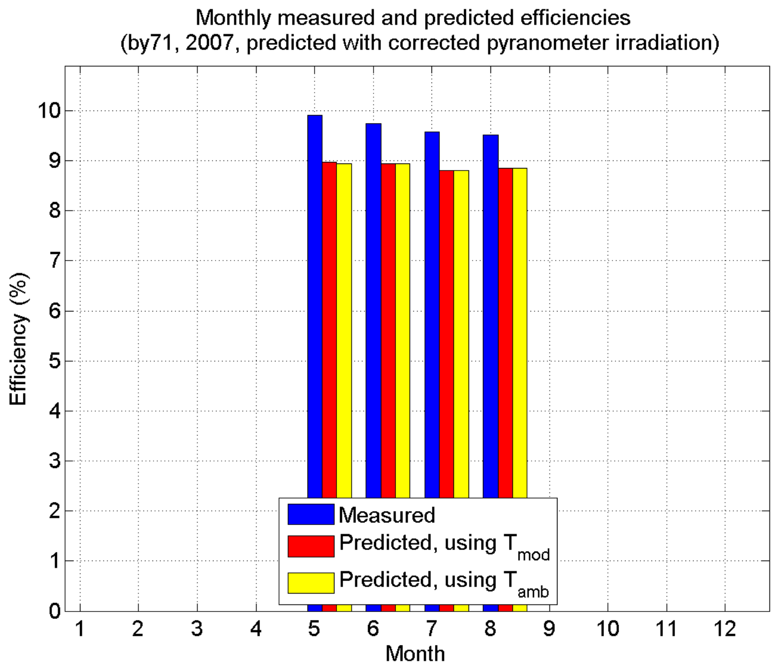
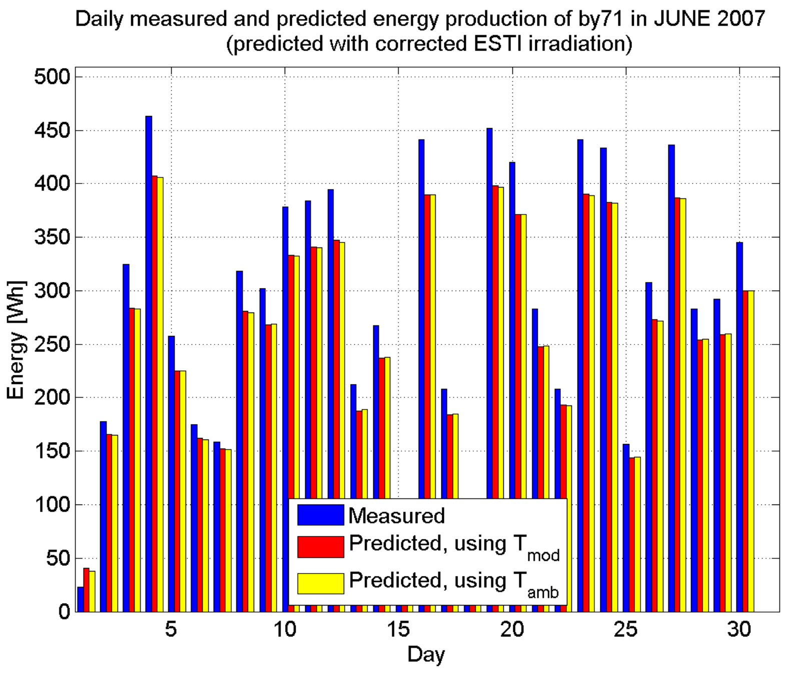


Figure 36: The monthly measured and predicted efficiency for by71 from May to August in 2007, calculated using corrected ESTI (upper graph) and pyranometer (lower graph) irradiances.



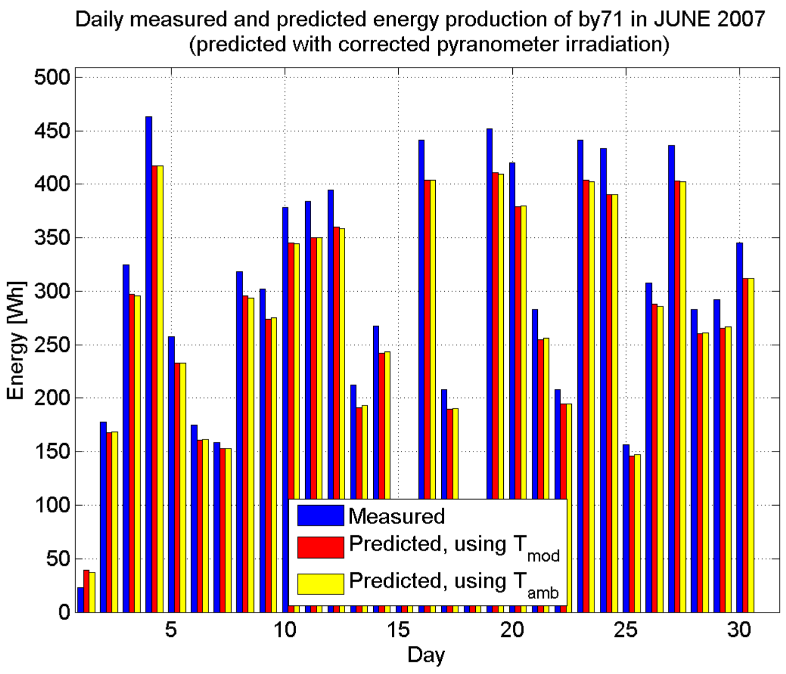


Figure 37: The daily measured and predicted energy productions for by71 in June, 2007. In the upper graph, the predictions are calculated using corrected ESTI irradiance and in the lower graph using corrected pyranometer irradiance.

as the CIS modules were measured for only half the month. In general, it still seems that from these four months July has been the best month for all the modules – this can be seen also in the graphs for individual modules in the previous sections. The reason for this is simple: in July the level of irradiation was high. The same applies for May, which was also a good month for all the modules.

The amount of points filtered seems to vary a lot between the modules, which causes uncertainity in the results. Reason for this is difficult to explain, as the weather conditions – and thus the irradiance stability – are the same for all the modules. For instance in May, a remarkably high number of points had been filtered from by72 as compared to the other modules. Further examination of the data revealed that indeed the majority of the filtering had occurred due to large changes in irradiance during the measurements. This applied for all the modules, and for all the months, though the number of points filtered in May for this reason was clearly the highest for by72. The days of especially high amount of points filtered due to this reason seemed to show low output from all the modules; thus it migh be a pure coincidence that during the measurements of by72 the irradiance conditions have been especially unstable. Or, there might also be some flaws in the data acquisition or in the program that have not been detected. However, it is obvious that this kind of filtering distorts the results towards performance at higher irradiances.

It is important to notice that direct module comparison from the basis of the energy production values found in the table 6 may be misleading since all the modules are not similar. In order to compare the performance of modules of different types and sizes accurately, in figure 38 the energy productions have been normalized to each module's watt-peak, *i.e.*, the STC maximum

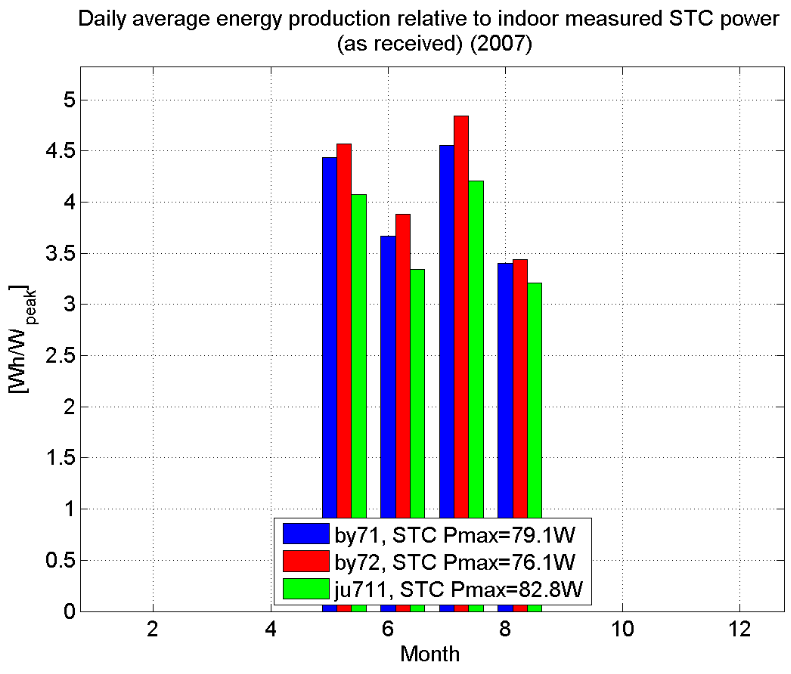
Table 6: Daily average energy productions, the number of measurement days and relative amount of measurement points filtered out, and the monthly average module temperatures for c-Si module ju711 and CIS modules by71 and by72 for months of May to August, 2007.

Month		ju711	by71	by72
May	Daily average energy yield (Wh)	270	330	348
	Number of measurement days	12	10	9
	Amount of points filtered (%)	10.6	14.3	20.5
	Monthly average module temperature (°C)	32.2	34.7	35.2
June	Daily average energy yield (Wh)	277	288	295
	Number of measurement days	30	30	30
	Amount of points filtered (%)	18.0	15.2	12.3
	Monthly average module temperature (°C)	35.6	38.3	38.9
July	Daily average energy yield (Wh)	365	358	367
	Number of measurement days	30	14	14
	Amount of points filtered (%)	9.7	9.4	15.6
	Monthly average module temperature (°C)	41.9	42.9	43.3
August	Daily average energy yield (Wh)	282	274	248
	Number of measurement days	28	28	25
	Amount of points filtered (%)	18.5	13.5	20.1
	Monthly average module temperature (°C)	38.0	41.1	40.8

power. However, different watt-peak values have been measured for these modules, and the choice of the value used in the normalization naturally affects the results. Here, two different watt-peaks are used: one measured indoors upon each module's arrival to the ESTI, and one determined by extrapolation from outdoor measurement data. Extrapolation had to be used because the STC conditions (25 °C module temperature at an irradiance of 1000 W/m^2) are hardly ever met outdoors – the module temperature is always higher than 25 °C at an irradiance of 1000 W/m^2. Therefore the outdoor watt-peak value was determined by linear extrapolation using the outdoor measurement data for module temperatures, irradiances and P_{max}:s.

For the c-Si module ju711, the difference between the two watt-peaks is very small – only 1%. In turn, for the CIS modules by71 and by72 the outdoor watt-peaks are 5% and 9% higher, respectively, than the indoor watt-peaks. Hence in the lower graph in figure 38, where outdoor watt-peak is used for the normalization, the differences between all the three modules are small. In the upper graph, where the indoor watt-peak is used for the normalization, the c-Si module has a much lower performance than the CIS modules. Differences between the two CIS modules are rather small, but in general by72 has performed slightly better than by71. As already seen in table 6, August is an exception: in that month the c-Si module dominates and additionally by72 has performed worse than by71. A reason for this might be that in August by72 has had three days less measurements than the other modules, and a further examination of the data revealed that these three days were days of relatively high output for the other modules.

From the basis of both the plots and the numerical values, it seems that the CIS modules (especially by72) are performing better than the c-Si module. The fact that the numerical values, that have not been normalized, gave



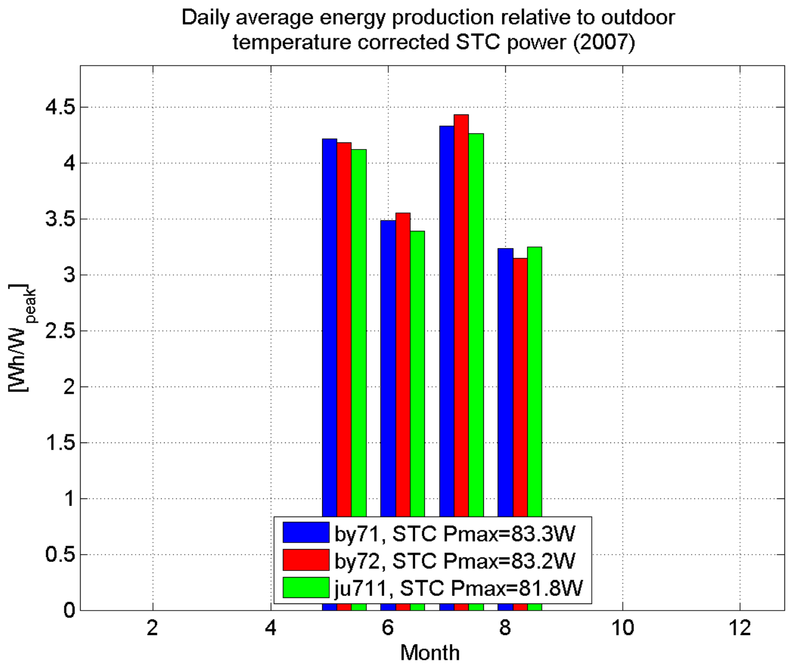


Figure 38: The daily average energy productions of one c-Si (ju711) and two CIS modules (by71 and by72) as normalized to watt-peak values (STC P_{max}:s) measured indoors as received (upper graph) and extrapolated from outdoor data (lower graph). Included are months from May to August in 2007.

similar results as the plots, is explained with the close to equal watt-peak values of these modules (indicated in figure 38). In general, thin film modules have lower efficiencies and hence lower energy yields than c-Si modules as discussed in section 3.3. The reason for the CIS modules dominating the c-Si module in almost every month of this study may be due to this particular c-Si module being an unusually bad module; it does not seem to have a particularly good efficiency (similar to that of the CIS modules), as it can be seen by comparing figures 36 and 33. Furthermore, this study covers only the warmer summer months. The output power of a PV module has an inverse relationship with its temperature (see section 3.2.3), and the negative power temperature coefficient of c-Si modules is typically about twice that of CIS modules [19]. However, from table 6 it can be seen that ju711 has actually had a few degrees lower module temperature than the CIS modules – probably due to its slightly better efficiency (less energy is dissipated as heat). Furthermore, from the basis of the module temperatures it seems that the warmest months have been July and August, and there is no drop in the performance of ju711 in these months. Still, to see if the temperature really affects the discrepancies between the modules, the study should be extended to cover also the colder winter months.

7 Conclusions

The purpose of this thesis was to introduce a MATLAB program developed for analyzing outdoor measurement data of PV modules. The program was developed to serve the outdoor measurement field of the European Solar Test Installation (ESTI), located at the Joint Research Centre (JRC) in Ispra, Italy. Simultaneously, a PostgreSQL database was created for storing the measurement data, and the program therefore had to include an interface to the database to access the data. The program was written not only to analyze the outdoor performance of different modules but also to compare the outdoor measured performance with a predicted performance based on indoor measurements. This was a part of the verification of an energy rating method of PV modules, being developed at ESTI.

In general, the program fulfilled the requirements: it is an efficient and fairly flexible tool for analyzing and predicting the performance of different PV modules being measured at ESTI, and also for evaluating the energy rating method. It also provided valuable information about problems with the outdoor measurement system, as several filters had to be included in the program to deal with erroneous measurement data. The usage and development of the program continued after the author's work at ESTI had finished.

It was discovered that the method used for energy rating is valid for conventional crystalline silicon (c-Si) modules, but for the copper inidium diselenide (CIS) modules some problems were identified. This problem has been encountered earlier as well [21], and it results primarily from the discrepancy between the indoor and outdoor performance of CIS modules [24]. Therefore it has been proposed that for the energy rating to be implemented correctly for CIS modules, a method with correct pre-conditioning of the modules prior to the indoor measurements is required. Alternatively, the method could employ outdoor data for the performance surface, but as explained previously, this would entail much longer measurement times [30].

In addition to comparing the outdoor measured performance of different modules to their predicted performances, the program was used for directly comparing the outdoor performance of different modules with each other. To do this, the modules' energy yields had to be normalized to each module's watt-peak value, *i.e.*, the STC maximum power. Different watt-peak values are measured for a module during its measurement period at ESTI, and it was found that the choice of the watt-peak value used in the normalization has a strong effect on the results. In the studies two CIS modules were compared with a c-Si module, and when the indoor measured watt-peak values were used for normalisation the CIS modules appeared to perform significantly better than the c-Si module. However, when the outdoor watt-peak values were used, the CIS modules tended to perform only slightly better. This is due to the indoor values not being correctly pre-conditioned as discussed in the preceding paragraph. Since only the hotter summer months were considered in this study the CIS modules would be expected to outperform somewhat the c-Si module as the effect of temperature on the modules performance is different for different module technologies. For a more robust comparison, a longer time period, ideally a full year, covering also the cooler winter months should be considered.

Furthermore, a particular problem encountered in the outdoor measurements is that irradiance sometimes changes quite significantly during the individual performance measurements of a module, resulting in distorted $I-V$ curves and hence faulty estimates of instantaneous maximum power. The irradiance is measured with two different devices that react to the changes in different manners: a reference cell and a pyranometer. The reference cell (specifically an ESTI sensor is employed at ESTI), is often preferred as it has a rapid response time and its spectral response is closer to that of the modules. The pyranometer in turn has a wider spectral response, and therefore might correspond more realistically to the actual amount of energy received from the sun. However, due to its slow response time, the instantaneous values received from the pyranometer may not be correct when the irradiance is rapidly changing (typically 30 irradiance values are read during a single $I-V$ scan of 5 seconds duration). A routine was therefore created in the program, used for calculating a corrected pyranometer value. So far very little attention has been given to this problem in the field of outdoor performance measurement of photovoltaics – and it would be interesting to go more deeply into the subject.

Finally, some comments on possible future developments of the program. Besides irradiance data, evaluation of other environmental data, such as ambient temperature and wind speed, could have been conducted as well in order to study their effect on module performance. Then, at present, to use and understand fully the program the user has to acquire a rather high level of MATLAB skills. This is without a doubt a major drawback for the program; it is generally not very user-friendly. Another clear drawback is the simplistic method the program uses for filtering the data: the faulty measurement points are merely replaced with zeros. The effect of filtering on the final results also remains quite hidden – although the program calculates and stores the number of filtered points, further evaluation would be necessary in order to understand the impact that filtering has had. The program is also rather sensitive to faults and irregularities in the measurement data; to make it less sensitive the program should be tested with a wider range of data. Finally, the program performance could surely be improved for instance by further minimizing the number of calculations in the main scripts and transferring them to separate functions instead.

To conclude, in the future the program should be developed to be more efficient and robust, more sophisticated in for example its filtering methods and more user-friendly. It could also include more evaluation of different kinds of environmental data.

References

[1] V. Quaschning. *Understanding Renewable Energy Systems.* Earthscan Canada, 2005.

[2] B. A. Sandén. The economic and institutional rationale of PV subsidies [Electronic Version]. *Solar Energy*, 78:137–146, 2005.

[3] R. M. Swanson. A Vision for Crystalline Silicon Photovoltaics. *Progress in Photovoltaics: Research and Applications*, 14:443–453, 2006.

[4] G. M. Masters. *Renewable and Efficient Electric Power Systems.* John Wiley & Sons, 2004.

[5] H. D. Young and R. A. Freedman. *University Physics with Modern Physics.* Pearson Addison Wesley, San Fransisco, CA, USA, 11th edition, 2004.

[6] Elements Database. Periodic Table of Elements. Retrieved August, 2008 from `http://www.elementsdatabase.com/`.

[7] M. A. Green. *Solar Cells. Operating Principles, Technology and System Applications.* The University of New South Wales, Kensington, NSW, Australia, 1998.

[8] Department of Physics C.R. Nave (Georgia State University and Astronomy). Hyperphysics, 2005. Retrieved October, 2007 from `http://hyperphysics.phy-astr.gsu.edu/hbase/hframe.html`.

[9] Discussions with E. D. Dunlop at the ESTI. 25.9.2008.

[10] A. Virtuani, W. Zaaiman, and H. Müllejans. Modified Visual Appearance of Cu(In,Ga)(Se,S)$_2$ Thin Film Solar Modules for Building Inte-

grated Photovoltaics. In *Proceedings of the 22nd European Photovoltaic Solar Energy Conference (EU PVSEC)*, Milan, Italy, 2007.

[11] International Organization for Standardization (ISO). *ISO/IEC 17025: General requirements for the competence of testing and calibration laboratories.* International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2005.

[12] R. Kenny. A personal e-mail message, received at 23.4.2008.

[13] R. Kröni, S. Stettler, G. Friesen, D. Chianese, R. Kenny, and W. Durisch. Energy Rating of Solar Modules. Technical report, Swiss Federal Office of Energy (SFOE), 2005.

[14] E. Dunlop. ESTI description. Retrieved April, 2008 from `http://sunbird.jrc.it/solarec/index.htm`.

[15] Solar Electricity Action (SOLAREC) European Communities. SOLAREC Home, 12.11.2007. Retrieved April, 2008 from `http://sunbird.jrc.it/solarec/`.

[16] D. Anderson. *Energy Rating of Photovoltaic Modules.* PhD thesis, University of Strathclyde, United Kingdom, 2002.

[17] Thies Clima. Ultrasonic Anemometer 2D. Retrieved May, 2008 from `http://www.thiesclima.com/usanemo_e.htm`.

[18] Kipp & Zonen, Delft, The Netherlands. *Instruction Manual of the CM11 pyranometer and the CM14 albedometer*, 2004.

[19] Discussions with R. Kenny at the ESTI. 9.1. and 15.1.2008.

[20] International Electrotechnical Commission (IEC). *IEC 60904.1: Measurement of photovoltaic current-voltage characteristics.* International Electrotechnical Commission (IEC), 2006.

[21] R. P. Kenny, A. Ioannides, H. Müllejans, W. Zaaiman, and E. D. Dunlop. Performance of thin film PV modules. *Thin Solid Films*, 511-512:663–672, 2006.

[22] R. P. Kenny, E. D. Dunlop, H. A. Ossenbrik, and H. Müllejans. A Practical Method for the Energy Rating of c-Si Photovoltaic Modules Based on Standard Tests. *Progress in Photovoltaics: Research and Applications*, 14:155–166, 2006.

[23] R. P. Kenny, G. Friesen, D. Chianese, A. Bernasconi, and E. D. Dunlop. Energy Rating of PV modules: comparison of methods and approach. In *Proceedings of the 3rd World Conference on Photovoltaic Solar Energy Conversion*, Osaka, Japan, 2003.

[24] A. Virtuani, D. Pavanello, R. P. Kenny, M. Nikolaeva-Dimitrova, and E. D. Dunlop. Comparison of Indoor and Outdoor Performance Measurements on CIS Thin Film Solar Modules. In *Proceedings of the 22nd European Photovoltaic Solar Energy Conference (EU PVSEC)*, Milan, Italy, 2007.

[25] A. Ekonoja, T. Lahtonen, and J. Mäntylä. Henkilökohtaisen tiedonhallinnan perusteet, 6.11.2003. Retrieved March, 2008 from `http://appro.mit.jyu.fi/doc/tiedonhallinta/`.

[26] J. C. Worsley and J. D. Drake. *Practical PostgreSQL.* O'Reilly, 2002.

[27] PostgreSQL Global Development Group. PostgreSQL homepage, 2008. Retrieved January, 2008 from `http://www.postgresql.org`.

[28] The MathWorks Inc. The MathWorks Home, 2008. Retrieved January, 2008 from `http://www.mathworks.com`.

[29] The MathWorks Inc. The MathWorks Support, 2008. Retrieved January, 2008 from `http://www.mathworks.com/support/`.

[30] R. P. Kenny, T. Huld, A. Virtuani, M. Nikolaeva-Dimitrova, and H. Kauko. Comparison of Outdoor Performance Measurements on CIS and c-Si Modules. Poster presented at the 23rd European Photovoltaic Solar Energy Conference (EU PVSEC), September 2008.

Appendices

A Fields in the Solarec database

The following table lists the fields in the Solarec -database, accompained with their data types. The original source files for the data are also indicated. To see the explanations for the names of the fields, see section 4.2.3.

Table 7: The fields in the database that originate from the enr-files, accompained with their data types.

Source file	Field name	SQL data type
enr	`date`	`date`
	`time`	`time`
	`irradiance_esti`	`double precision`
	`irradiance_pyran`	`double precision`
	`t_amb`	`double precision`
	`t_mod`	`double precision`
	`isc`	`double precision`
	`voc`	`double precision`
	`pmax`	`double precision`
	`impp`	`double precision`
	`vmpp`	`double precision`
	`fillfactor`	`double precision`

Table 8: The fields in the database that originate from the ene and raw-files, accompained with their data types. The square brackets refer to (double precision) data arrays.

Source file	Field name	SQL data type
ene	`esti_irr_ave`	`double precision`
	`pyran_irr_ave`	`double precision`
	`irr_corr`	`double precision`
	`irr_change`	`double precision`
	`esti_voltage`	`double precision`
	`isc_corr`	`double precision`
	`voc_corr`	`double precision`
	`pmax_corr`	`double precision`
	`impp_corr`	`double precision`
	`vmpp_corr`	`double precision`
	`ff_corr`	`double precision`
raw	`vmod`	`double precision[]`
	`imod`	`double precision[]`
	`irr_iv_esti`	`double precision[]`
	`irr_iv_pyran`	`double precision[]`
	`v_iv_esti`	`double precision[]`
	`imod_iv_corrected`	`double precision[]`

B The script and functions for analyzing data from ene-files

B.1 The main script: ene_analysis

```
%*************************************************************%
% This is the main script for analyzing outdoor ene —data
% retrieved from the database 'solarec'.
% Author: Hanne Kauko
% Date: 13.2.2008
%*************************************************************%

close all
clear all

%% CONNECTING TO THE DATABASE
% Using the interface functions (pq...) to create connection to the
% database

connstr = 'host=emu.jrc.it dbname=solarec user=thomas';
myconn = pqconnectdb(connstr);
mystat = pqstatus(myconn) %#ok<NOPTS>


%% GETTING THE DESIRED DATA FROM THE DATABASE
% Using the function getdata_db_ene to retrieve the desired data from the
% database and arrange it to appropriate arrays.
% Here G_esti is the corrected to ESTI irradiation (in W/m2).
```

```
[N, nYears, nRows, nDays, Dstart, Dend, mod, l, time,...
     G_esti, G_esti_av, esti_change, G_pyr_av, Pmax, Tmod, Tamb] ...
     = getdata_db_ene(myconn);


%% CORRECTING THE PYRANOMETER IRRADIANCE
% Using the function pyran_correction to correct the value of pyranometer
% irradiance.
% Here G_pyr is the corrected to pyranometer irradiation (in W/m2).

G_pyr = pyran_correction(G_pyr_av, G_esti_av, G_esti,...
    esti_change, N, nRows, mod);



%% EMPIRICAL ANALYSIS

% Calculating the empirical Tmod with the function NOCT_estim
% Using the corrected to ESTI irradiation
G_str = 'corrected to ESTI';
Tmod_emp_es = NOCT_estim(Tmod, Tamb, G_esti, N, mod, nRows, G_str);
% Using the corrected to pyranometer irradiation
G_str = 'corrected to pyranometer';
Tmod_emp_py = NOCT_estim(Tmod, Tamb, G_pyr, N, mod, nRows, G_str);


% To calculate the empirical Pmaxes, the empirical equation and
% parameters, as well as the module area and different STC outputs,
% must be stored in the text file 'empirical_data.txt' residing in
% the same folder with the M-files.

Pmax_emp_es = zeros(1,1,N); Pmax_emp_py = zeros(1,1,N);
Pmax_empT_es = zeros(1,1,N); Pmax_empT_py = zeros(1,1,N);
Pmax_STC_emp = zeros(1,N);
```

```
Pmax_STC_lapss = zeros(1,N);
Pmax_STC_out = zeros(1,N);
Pmax_STC_label = zeros(1,N);
mod_area = zeros(1,N);
match=0;

%Opening 'emprical_data.txt' and retrieving the required data from it.
fid = fopen('empirical_data.txt');
C = textscan(fid,'%s %s %f %f %f %f %f %f %f %f %f %f',...
    'commentStyle','%');
[a,b,c,d,e,f] = C{3:8};
for n=1:N
    for row=1:length(C{1})
        if length(char(C{1}(row)))==length(mod{n}) &&...
                all(char(C{1}(row))==mod{n})==1
            match = 1;
            Pmax_eq = inline(vectorize(char(C{2}(row))),...
                'Irr','T','a','b','c','d','e','f');
            Pmax_emp_es(1:nRows(n),:,n) = ...
                Pmax_eq(G_esti(1:nRows(n),:,n),...
                Tmod(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_empT_es(1:nRows(n),:,n) = ...
                Pmax_eq(G_esti(1:nRows(n),:,n),...
                Tmod_emp_es(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_emp_py(1:nRows(n),:,n) = ...
                Pmax_eq(G_pyr(1:nRows(n),:,n),...
                Tmod(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_empT_py(1:nRows(n),:,n) = ...
                Pmax_eq(G_pyr(1:nRows(n),:,n),...
                Tmod_emp_py(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_STC_emp(n) = Pmax_eq(1000,25,...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_STC_lapss(n) = C{10}(row);
            Pmax_STC_out(n) = C{11}(row);
            Pmax_STC_label(n) = C{12}(row);
            mod_area(n) = C{9}(row);
        end
    end
end
if  match==0
    disp(['The empirical data for this module has not been stored',...
        'in the file empirical_data.txt.'])
    return
end
fclose(fid);


%% FILTERING
% Using the function data_filter_ene to filter the data.
% The filtered points are marked in an array 'flag', and the
% value of flag(i,j) tells the reason why the point has been filtered
% according to the following:
% flag(j,i) = 1, if Pmax was zero in the original data
% flag(j,i) = 2, if Pmax<0 in the original data
% flag(j,i) = 3, if Pmax_emp or Pmax_empT <0 or complex in the original
% data
% flag(j,i) = 4, if abs(Pmax-Pmax_emp)/max(Pmax,Pmax_emp) > 0.5
% flag(j,i) = 5, if the (ESTI) irradiance has changed more than 5% during
% the measurement
% flag(j,i) = 6, if Pmax>Pmax_STC_max (the maximum STC Pmax) in the
% original data
```

```
%Calculating the maximum STC Pmax
Pmax_STC_max = zeros(N,1);
for n=1:N
    Pmax_STC_max(n) = max([Pmax_STC_emp(n) Pmax_STC_lapss(n) ...
        Pmax_STC_out(n) Pmax_STC_label(n)]);
end

[Pmax, Pmax_emp_es, Pmax_empT_es, Pmax_emp_py, Pmax_empT_py,...
    G_esti, G_pyr, flag, nPoints_deleted] ...
    = data_filter_ene(Pmax, Pmax_emp_es, Pmax_empT_es, Pmax_emp_py,...
    Pmax_empT_py, esti_change, G_esti, G_pyr, Pmax_STC_max, N, nRows);


%% CREATING A BASE ARRAY
% Using the function 'base_matrix' to create a 'base_array', in which
% every possible measurement point for the concerned time period is
% included, and the data form each module recides at their correct points.
% 'base_matrix' calculates the mode measurement interval for the time
% period, and uses this to create a common time stamp for all the modules
% discussed. 'nMeas_day' tells the (fixed) number of measurement
% points within a day.

% Arranging the  original and filtered data into an array
Data = [Pmax Pmax_emp_es Pmax_empT_es Pmax_emp_py Pmax_empT_py ...
    G_esti G_pyr Tmod flag nPoints_deleted];

[base_array, nMeas_day, dt_in_minutes] = ...
    base_matrix(nDays, Dstart, Dend, time, Data, N, 1, nRows);


%% ENERGY CALCULATIONS
% Using the function energycalc to calculate the daily and monthly
% energies from each different Pmax, and daily and monthly irradiation.
% The daily sums are stored into 4-nYears arrays with the postfix _day,
% consisting of 12x31 matrices. The 3rd dimension is the no of years of
% measurement and the 4th is the no of modules analyzed.
% The monthly sums are stored into 3D arrays consisting 12x(no of years)
% of matrices, 3rd dimension being the number of modules analyzed.

[E_day, E_month, E_fill_day, E_fill_month, E_emp_es_day, E_emp_es_month,...
    E_empT_es_day, E_empT_es_month, E_emp_py_day, E_emp_py_month, ...
    E_empT_py_day, E_empT_py_month, G_esti_day, G_esti_month, ...
    G_pyr_day, G_pyr_month, Tmod_av_day, Tmod_av_month, flag_array,...
    date_array]...
    = energycalc (base_array, nDays, nYears, Dstart, Dend,...
    N, nMeas_day, dt_in_minutes);


%% EFFICIENCY CALCULATIONS

% Calculating the efficiencies using measured outputs and corrected to
% ESTI irradiance
[Eff_es, Eff_es_day, Eff_es_month] = ...
    effcalc (Pmax, E_day, E_month, G_esti, G_esti_day, G_esti_month, ...
      mod_area, nYears, N);

% Calculating the efficiencies using measured outputs and corrected to
% pyranometer -irradiance
[Eff_py, Eff_py_day, Eff_py_month] = ...
    effcalc (Pmax, E_day, E_month, G_pyr, G_pyr_day, G_pyr_month, ...
    mod_area, nYears, N);

% Calculating the efficiencies using empirical outputs and corrected to
% ESTI -irradiance.
[Eff_emp_es, Eff_emp_es_day, Eff_emp_es_month] = ...
    effcalc (Pmax_emp_es, E_emp_es_day, E_emp_es_month, ...
    G_esti, G_esti_day, G_esti_month, mod_area, nYears, N);
[Eff_empT_es, Eff_empT_es_day, Eff_empT_es_month] = ...
    effcalc (Pmax_empT_es, E_empT_es_day, E_empT_es_month, ...
    G_esti, G_esti_day, G_esti_month, mod_area, nYears, N);

% Calculating the efficiencies using measured outputs and corrected to
% pyranometer -irradiance
[Eff_emp_py, Eff_emp_py_day, Eff_emp_py_month] = ...
    effcalc (Pmax_emp_py, E_emp_py_day, E_emp_py_month, ...
    G_pyr, G_pyr_day, G_pyr_month, mod_area, nYears, N);
[Eff_empT_py, Eff_empT_py_day, Eff_empT_py_month] = ...
    effcalc (Pmax_empT_py, E_empT_py_day, E_empT_py_month, ...
    G_pyr, G_pyr_day, G_pyr_month, mod_area, nYears, N);


%% PLOTTING THE RESULTS

Dstart_vec = datevec(Dstart);
year_vec = zeros(1,nYears);
for k=1:nYears
    year_vec(k)=Dstart_vec(1)+(k-1);
end

%% Using function 'plotter' to plot the monthly and daily measured and
%% predicted energy productions and efficiencies.

% Using corrected ESTI irradiation
choice = input(['\n Do you want to make graphs where measured ',...
'\n values are compared with predicted ones,',...
'\n using values calculated with corrected ESTI irradiance',...
'\n(for each module separately) (y/n)? '],'s');
if choice=='y'
    irrad_str='corrected ESTI';
    irrad_fn='ESTIcorr';
    plotter(E_month, E_emp_es_month, E_empT_es_month,...
        Eff_es_month, Eff_emp_es_month, Eff_empT_es_month,...
        E_day, E_emp_es_day, E_empT_es_day, Eff_es_day, Eff_emp_es_day,...
        Eff_empT_es_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end

% Using corrected pyranometer irradiation
choice = input(['\n Do you want to make graphs where measured',...
'\n values are compared with predicted ones,',...
'\n using values calculated with corrected pyranometer irradiance',...
'\n(for each module separately) (y/n)? '],'s');
if choice=='y'
    irrad_str='corrected pyranometer';
    irrad_fn='pyr_corr';
    plotter(E_month, E_emp_py_month, E_empT_py_month,...
        Eff_py_month, Eff_emp_py_month, Eff_empT_py_month,...
        E_day, E_emp_py_day, E_empT_py_day, Eff_py_day, Eff_emp_py_day,...
        Eff_empT_py_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end


%% Using function 'plotter_meas_only' to plot just daily and monthly energy
%% productions and irradiances, making no comparison with the predicted
%% values

% Plotting the daily and monthly energy productions, using the energy
% calculated with Pmax_fill having gaps missing because of filtering
% filled with average ones.
choice = input(['\n Do you want to make graphs with',...
    ' measured (filled) energy production only (y/n)? '],'s');
if choice=='y'
    tit = 'energy production';
    ylbl = 'Energy [Wh]';
    filename = 'meas_energy';
    plotter_meas_only(E_fill_day, E_fill_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end

% Plotting the daily and monthly corrected ESTI irradiations
choice = input(['\n Do you want to make graphs with corrected',...
    '\n ESTI irradiations only (y/n)? '],'s');
if choice=='y'
    tit = 'corrected ESTI irradiance';
    ylbl = 'Irradiance [Wh/m^2]';
    filename = 'ESTIcorr';
    plotter_meas_only(G_esti_day, G_esti_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end

% Plotting the daily and monthly corrected pyranometer irradiations
choice = input(['\n Do you want to make graphs with corrected',...
    '\n pyranometer irradiations only (y/n)? '],'s');
if choice=='y'
    tit = 'corrected pyranometer irradiance';
    ylbl = 'Irradiance [Wh/m^2]';
    filename = 'pyr_corr';
    plotter_meas_only(G_pyr_day, G_pyr_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end

%% Using the function 'mod_comparison' to make module comparison plots.

choice = input(['\n Do you want to make module',...
    'comparison plots (y/n)? '],'s');
if choice=='y'
```

```
    irrad_str={'corrected ESTI', 'corrected pyranometer'};
    irrad_fn={'ESTIcorr','pyr_corr'};
    mod_comparison(Pmax_STC_lapss, Pmax_STC_out,...
        Pmax_STC_emp, Pmax_STC_label, E_day, G_esti_day, G_pyr_day,...
        Tmod_av_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end


%% SAVING THE DATA
% Using the function 'data_writer' to write the daily and monthly data into
% text files.

choice = input(['\n Do you want to write the results',...
    'into text-files (y/n)? '],'s');
if choice=='y'
    data_writer (E_day, E_month, E_emp_es_day, E_emp_es_month,...
        E_empT_es_day, E_empT_es_month, E_emp_py_day, E_emp_py_month,...
        E_empT_py_day, E_empT_py_month, ...
        Eff_es_day, Eff_es_month, Eff_py_day, Eff_py_month,...
        Eff_emp_es_day, Eff_emp_es_month, Eff_emp_py_day,...
        Eff_emp_py_month, Eff_empT_es_day, Eff_empT_es_month, ...
        Eff_empT_py_day, Eff_empT_py_month, G_esti_day, G_esti_month,...
        G_pyr_day, G_pyr_month, Tmod_av_day, Tmod_av_month, flag_array, ...
        mod, nYears, N, date_array);
end

%%
pqclear(res);
pqfinish(myconn);
```

B.2 The function getdata_db_ene

```
function [fN, fnYears, nRows, fnDays, Dstart_ser, Dend_ser, fmod, fl,...
    ftime, fG_esti_cor, fG_esti_av, fG_change, fG_pyr_av, ...
    fPmax, fTmod, fTamb]...
    = getdata_db_ene(myconn)

%-----------------------------------------------------------------------%
% This function retireves the required .ene -data from the database for
% user-defined time period and set of modules
% and arranges the data into appropriate arrays.
%-----------------------------------------------------------------------%

% Defining the time period to be analyzed
text_Dstart = 'The starting date (yyyy-mm-dd) of the data analysis: ';
text_Dend = 'The last date (yyyy-mm-dd) to be considered in the analysis: ';
Dstart = input(text_Dstart,'s');
Dend = input(text_Dend,'s');

% Converting the starting and ending dates into serial number format
Dstart_ser = datenum(Dstart,'yyyy-mm-dd');
Dend_ser = datenum(Dend,'yyyy-mm-dd');
fnDays = Dend_ser-Dstart_ser+1; % No of days in the dataset

% Converting the starting and ending dates into vector format (in order to
% calculate the no. of years of measurement)
Dstart_vec = datevec(Dstart_ser); Dend_vec=datevec(Dend_ser);
fnYears = Dend_vec(1)-Dstart_vec(1)+1; % No of years of measurement

% Defining the modules to be analyzed
fN = input('How many different modules are compared? ');

fmod = cell(1,fN);
res = zeros(1,fN,'uint32');
nFields = zeros(1,fN);
```

```
nRows = zeros(1,fN);

% Defining the data to be retrieved
for i=1:fN
   name = ['Name of the module ' num2str(i) '? '];
   fmod{i} = input(name,'s'); % Cell array containing names of the modules
   text = ['SELECT date,time,pmax_corr,irr_corr,irr_change,',...
        'esti_irr_ave,pyran_irr_ave,t_mod,t_amb FROM %s',...
        'WHERE date≥''%s'' AND date≤''%s'' ORDER BY date,time'];
   exeStr = sprintf(text,fmod{i},Dstart,Dend);
   res(i) = pqexec(myconn, exeStr);
   nFields(i) = pqnfields(res(i));
   nRows(i) = pqntuples(res(i));
end

%Displaying the size of the dataset
disp('Number of columns in the dataset of each module: ')
disp(nFields)
disp('Number of rows in the dataset of each module: ')
disp(nRows)

fl = max(nRows);
ftime = zeros(fl,6,fN);
fPmax = zeros(fl,1,fN);
fG_esti_cor = zeros(fl,1,fN);
fG_esti_av = zeros(fl,1,fN);
fG_pyr_av = zeros(fl,1,fN);
fG_change = zeros(fl,1,fN);
fTmod = zeros(fl,1,fN);
fTamb = zeros(fl,1,fN);

% Retrieving the data using the variable 'res' recieved above and
% creating the time and data arrays
```

```
for i=1:fN
    for j=1:nRows(i)
        timeStr = [pqgetvalue(res(i),j-1,0) ' ' pqgetvalue(res(i),j-1,1)];
        % Dates and times of each measurement:
        ftime(j,:,i) = datevec(timeStr,'yyyy-mm-dd HH:MM:SS');
         % Corrected Pmax (W)
        fPmax(j,:,i) = pqgetvalue(res(i),j-1,2);
        % Corrected to ESTI irrad., (W/m2)
        fG_esti_cor(j,:,i) = pqgetvalue(res(i),j-1,3);
         % Change in the ESTI irrad. during meas. (%)
        fG_change(j,:,i) = pqgetvalue(res(i),j-1,4);
        % Average ESTI irrad. (W/m2)
        fG_esti_av(j,:,i) = pqgetvalue(res(i),j-1,5);
        % Average pyranometer irrad. (W/m2)
        fG_pyr_av(j,:,i) = pqgetvalue(res(i),j-1,6);
        % Module temp. (C)
        fTmod(j,:,i) = pqgetvalue(res(i),j-1,7);
         % Ambient temp. (C)
        fTamb(j,:,i) = pqgetvalue(res(i),j-1,8);
    end
end
```

B.3 The function pyran_correction

```
function fG_pyr_cor = ...
    pyran_correction(fG_pyr_av, fG_esti_av, fG_esti_cor, festi_change, ...
    fN, fnRows, fmod)

%------------------------------------------------------------------------%
% This function calculates a corrected value for pyranometer irradiance.
% The correction is done only for measurement points, where ESTI
% irradiance has changed less than 5% (this was found a proper limit by
% looking the IV-curves and irradiance curves from measurements with
% different changes in the ESTI irradiance).
% If ESTI change is < 2%, the correction is done by
%   G_pyr_cor(i) = G_pyr_av(i)./G_esti_av(i)*G_esti_cor(i);
% If ESTI change is between 2% and 5%, the nearest point j where
% ESTI change < 2% within 3 steps is found, and the correction is then
% done by
%   G_pyr_cor(i) = G_pyr_av(j)./G_esti_av(j)*G_esti_cor(i).
%------------------------------------------------------------------------%

fG_pyr_cor = zeros(size(fG_pyr_av));
step_count = zeros(length(fnRows),fN);

for n=1:fN
    for i=1:fnRows(n)
        if festi_change(i,1,n)≤2
            fG_pyr_cor(i,1,n) = fG_pyr_av(i,1,n)./fG_esti_av(i,1,n)...
                *fG_esti_cor(i,1,n);
            step_count(i,n)=1;
        elseif 2 < festi_change(i,1,n) && festi_change(i,1,n) < 5
            if festi_change(i+1,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i+1,1,n)./...
                    fG_esti_av(i+1,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=2;
            elseif festi_change(i-1,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i-1,1,n)./...
                    fG_esti_av(i-1,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=2;
            elseif festi_change(i+2,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i+2,1,n)./...
                    fG_esti_av(i+2,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=3;
            elseif festi_change(i-2,1,n)≤2
```

```
                fG_pyr_cor(i,1,n) = fG_pyr_av(i−2,1,n)./...
                    fG_esti_av(i−2,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=3;
            elseif festi_change(i+3,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i+3,1,n)./...
                    fG_esti_av(i+3,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=4;
            elseif festi_change(i−3,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i−3,1,n)./...
                    fG_esti_av(i−3,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=4;
            elseif festi_change(i+4,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i+4,1,n)./...
                    fG_esti_av(i+4,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=5;
            elseif festi_change(i−4,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i−4,1,n)./...
                    fG_esti_av(i−4,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=5;
            elseif festi_change(i+5,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i+5,1,n)./...
                    fG_esti_av(i+5,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=6;
            elseif festi_change(i−5,1,n)≤2
                fG_pyr_cor(i,1,n) = fG_pyr_av(i−5,1,n)./...
                    fG_esti_av(i−5,1,n)*fG_esti_cor(i,1,n);
                step_count(i,n)=6;
            end
        end
    end
    figure;
    hist(step_count(:,n),6)
    tit_text = ['Histogram for the steps to be taken from a point',...
```

```
        '\n having a change in ESTI irradiance between 2%% and 5%%',...
        '\n to a point with lt. 2%% change',...
        '\n(%s; the total number of points is %d)'];
    tit = sprintf(tit_text,fmod{n},fnRows(n));
    title(tit)
    xlabel_text = ['Number of steps (in the first bin there are',...
        'the points with no steps to be taken)'];
    xlabel(xlabel_text)
    ylabel('Count')
    yMax = fnRows(n);
    axis([0 6 0 yMax]);
end
```

B.4 The function data_filter_ene

```
function [fPmax, fPmax_emp_E, fPmax_empT_E, fPmax_emp_P, fPmax_empT_P,...
    fG_esti, fG_pyr, fFlag, fnPoints_deleted] ...
           = data_filter_ene(fPmax, fPmax_emp_E, fPmax_empT_E, ...
           fPmax_emp_P, fPmax_empT_P, fG_change, fG_esti, fG_pyr, ...
           fPmax_STC_max, fN, fnRows)

%-----------------------------------------------------------------------------%
% This function filters out bad data points, i.e. sets empirical and
% measured Pmaxes and irradiance to zero if there is something strange
% in the data. The filtered points are marked in an array 'flag', and the
% value of flag(i,j) tells the reason why the point has been filtered
% according to the following:
%
% flag(j,i) = 1, if Pmax was zero in the original data
% flag(j,i) = 2, if Pmax<0 in the original data
% flag(j,i) = 3, if Pmax_emp or Pmax_empT <0 or complex in the
% original  data (this may occur at low temperatures if the empirical
% equation is 'bad')
% flag(j,i) = 4, if abs(Pmax−Pmax_emp)/max(Pmax,Pmax_emp) > 0.5
% flag(j,i) = 5, if the (ESTI) irradiance has changed more than 5%
% during the measurement
% flag(j,i) = 6, if Pmax>Pmax_STC_max (the maximum STC Pmax) in the
% original data
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−%

text = ['The treshold value for the change in ESTI irradiance',...
    'for filtering the data is by default 5%.'];
disp(text)
choice = input('Do you want to set another treshold (y/n)? ','s');
if choice=='y'
    G_th = input('Select the value for the treshold (in %): ');
else
    G_th = 5;
end

fFlag = zeros(size(fPmax));
fnPoints_deleted = zeros(size(fPmax));

for i=1:fN
    for j=1:fnRows(i)
        if fPmax(j,:,i) == 0
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 1;
        end
```

```
if fPmax(j,:,i) < 0
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0;
    fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0;
    fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0;
    fG_pyr(j,:,i) = 0;
    fFlag(j,:,i) = 2;
end
if fPmax_emp_E(j,:,i)<0 || isreal(fPmax_emp_E(j,:,i))==0 || ...
        fPmax_empT_E(j,:,i)<0 || isreal(fPmax_empT_E(j,:,i))==0 ...
        || fPmax_emp_P(j,:,i)<0 || isreal(fPmax_emp_P(j,:,i))==0 ...
        || fPmax_empT_P(j,:,i)<0 || isreal(fPmax_empT_P(j,:,i))==0
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0;
    fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0;
    fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0;
    fG_pyr(j,:,i) = 0;
    fFlag(j,:,i) = 3;
end
if abs(fPmax(j,:,i)-fPmax_emp_E(j,:,i))/...
        max([fPmax(j,:,i) fPmax_emp_E(j,:,i)]) > 0.5 || ...
        abs(fPmax(j,:,i)-fPmax_emp_P(j,:,i))/...
        max([fPmax(j,:,i) fPmax_emp_P(j,:,i)]) > 0.5
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0;
    fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0;
    fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0;
```

```
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 4;
        end
        if fG_change(j,:,i) > G_th
            fPmax(j,:,i) = 0;
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 5;
        end
        if fPmax(j,:,i) > fPmax_STC_max(i)
            fPmax(j,:,i) = 0;
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 6;
        end
        % Calculating total the number of points deleted by filtering
        % (necessary to do separately, as there might be some overlapping
        % in the flags)
        fnPoints_deleted(j,:,i) = nnz(fPmax(j,:,i)==0);
    end
end
```

C The script and functions for analyzing data from enr-files

C.1 The main script: enr_analysis

```
%***********************************************************%
% This is the main script for analyzing outdoor enr —data
% retrieved from the database 'solarec'.
% Author: Hanne Kauko
% Date: 13.2.2008
%***********************************************************%

close all
clear all

%% CONNECTING TO THE DATABASE
% Using the interface functions (pq...) to create connection to the
% database

connstr = 'host=emu.jrc.it dbname=solarec user=thomas';
myconn = pqconnectdb(connstr);
mystat = pqstatus(myconn) %#ok<NOPTS>


%% GETTING THE DESIRED DATA FROM THE DATABASE
% Using the function getdata_db_ene to retrieve the desired data from the
% database and arrange it to appropriate arrays.

[N, nYears, nRows, nDays, Dstart, Dend, mod, l, time,...
```

```
    G_esti, G_pyr, Pmax, Tmod, Tamb] = getdata_db_enr(myconn);


%% EMPIRICAL ANALYSIS

% Calculating the empirical Tmod with the function NOCT_estim
% Using ESTI irradiation
G_str = 'ESTI';
Tmod_emp_es = NOCT_estim(Tmod, Tamb, G_esti, N, mod, nRows, G_str);
% Using pyranometer irradiation
G_str = 'pyranometer';
Tmod_emp_py = NOCT_estim(Tmod, Tamb, G_pyr, N, mod, nRows, G_str);

% To calculate the empirical Pmaxes, the empirical equation and
% parameters, as well as the module area and different STC outputs,
% must be stored in the text file 'empirical_data.txt' esiding in
% the same folder with the M-files.

Pmax_emp_es = zeros(1,1,N); Pmax_emp_py = zeros(1,1,N);
Pmax_empT_es = zeros(1,1,N); Pmax_empT_py = zeros(1,1,N);
Pmax_STC_emp = zeros(1,N);
Pmax_STC_lapss = zeros(1,N);
Pmax_STC_out = zeros(1,N);
Pmax_STC_label = zeros(1,N);
mod_area = zeros(1,N);

%Opening 'emprical_data.txt' and retrieving the required data from it.
fid = fopen('empirical_data.txt');
C = textscan(fid, '%s %s %f %f %f %f %f %f %f %f %f %f', ...
    'commentStyle','%');
[a,b,c,d,e,f] = C{3:8};
for n=1:N
    for row=1:length(C{1})
```

```
        if length(char(C{1}(row)))==length(mod{n})  && ...
                all(char(C{1}(row))==mod{n})==1
            match = 1;
            Pmax_eq = inline(vectorize(char(C{2}(row))),...
                'Irr','T','a','b','c','d','e','f');
            Pmax_emp_es(1:nRows(n),:,n) = ...
                Pmax_eq(G_esti(1:nRows(n),:,n),Tmod(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_empT_es(1:nRows(n),:,n) = ...
                Pmax_eq(G_esti(1:nRows(n),:,n),...
                Tmod_emp_es(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_emp_py(1:nRows(n),:,n) = ...
                Pmax_eq(G_pyr(1:nRows(n),:,n),Tmod(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_empT_py(1:nRows(n),:,n) = ...
                Pmax_eq(G_pyr(1:nRows(n),:,n),...
                Tmod_emp_py(1:nRows(n),:,n),...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_STC_emp(n) = Pmax_eq(1000,25,...
                a(row),b(row),c(row),d(row),e(row),f(row));
            Pmax_STC_lapss(n) = C{10}(row);
            Pmax_STC_out(n) = C{11}(row);
            Pmax_STC_label(n) = C{12}(row);
            mod_area(n) = C{9}(n);
        end
    end
end
if  match==0
    disp(['The empirical data for this module has not been stored',...
        'in the file empirical_data.txt.'])
    return
end
```

```
fclose(fid);



%% FILTERING
% Calling the function data_filter to filter the data.
% The filtered points are marked in an array 'flag', and the
% value of flag(i,j) tells the reason why the point has been filtered
% according to the following:
% flag(j,i) = 1, if Pmax was zero in the original data
% flag(j,i) = 2, if Pmax<0 in the original data
% flag(j,i) = 3, if Pmax_emp or Pmax_empT <0 or complex in the original
% data
% flag(j,i) = 4, if abs(Pmax-Pmax_emp)/max(Pmax,Pmax_emp) > 0.5
% flag(j,i) = 5, if abs(G_esti-G_pyr)/max(G_esti,G_pyr) > 0.3
% flag(j,i) = 6, if Pmax>Pmax_STC_max (the maximum STC Pmax) in the
% original data

%Calculating the maximum STC Pmax
Pmax_STC_max = zeros(N,1);
for n=1:N
    Pmax_STC_max(n) = max([Pmax_STC_emp(n) Pmax_STC_lapss(n) ...
        Pmax_STC_out(n) Pmax_STC_label(n)]);
end

[Pmax, Pmax_emp_es, Pmax_empT_es, Pmax_emp_py, Pmax_empT_py,...
    G_esti, G_pyr, flag, nPoints_deleted] ...
    = data_filter_enr(Pmax, Pmax_emp_es, Pmax_empT_es, ...
    Pmax_emp_py, Pmax_empT_py, G_esti, G_pyr, Pmax_STC_max, N, nRows);


%% CREATING A BASE ARRAY
% Using the function 'base_matrix' to create a 'base_array', in which
% every possible measurement point for the concerned time period is
% included, and the data form each module recides at their correct points.
% 'base_matrix' calculates the mode measurement interval for the time
% period, and uses this to create a common time stamp for all the modules
% discussed. 'nMeas_day' tells the (fixed) number of measurement
% points within a day.

% Arranging the  original and filtered data into an array
Data = [Pmax Pmax_emp_es Pmax_empT_es Pmax_emp_py Pmax_empT_py ...
    G_esti G_pyr Tmod flag nPoints_deleted];

[base_array, nMeas_day, dt_in_minutes] = ...
    base_matrix(nDays, Dstart, Dend, time, Data, N, l, nRows);


%% ENERGY CALCULATIONS
% Using the function energycalc to calculate the daily and monthly
% energies from each different Pmax, and daily and monthly irradiation.
% The daily sums are stored into 4-D arrays with the postfix _day,
% consisting of 12x31 matrices. The 3rd dimension is the no of years of
% measurement and the 4th is the no of modules analyzed.
% The monthly sums are stored into 3-D arrays consisting 12x(no of years)
% of matrices, 3rd dimension being the number of modules analyzed.

[E_day, E_month, E_fill_day, E_fill_month, E_emp_es_day, ...
    E_emp_es_month, E_empT_es_day, E_empT_es_month, ...
E_emp_py_day, E_emp_py_month,  E_empT_py_day, E_empT_py_month,...
 G_esti_day, G_esti_month, G_pyr_day, G_pyr_month, Tmod_av_day, ...
 flag_array, date_array]...
    = energycalc (base_array, nDays, nYears, Dstart, Dend,...
    N, nMeas_day, dt_in_minutes);
```

```
%% EFFICIENCY CALCULATIONS

% Calculating the efficiencies using measured outputs and ESTI irradiance
[Eff_es, Eff_es_day, Eff_es_month] = ...
    effcalc (Pmax, E_day, E_month, G_esti, G_esti_day, G_esti_month, ...
    mod_area, nYears, N);

% Calculating the efficiencies using measured outputs and pyranometer
%  irradiance
[Eff_py, Eff_py_day, Eff_py_month] = ...
    effcalc (Pmax, E_day, E_month, G_pyr, G_pyr_day, G_pyr_month, ...
    mod_area, nYears, N);

% Calculating the efficiencies using empirical outputs and
% ESTI-irradiance
[Eff_emp_es, Eff_emp_es_day, Eff_emp_es_month] = ...
    effcalc (Pmax_emp_es, E_emp_es_day, E_emp_es_month,...
    G_esti, G_esti_day, G_esti_month, mod_area, nYears, N);
[Eff_empT_es, Eff_empT_es_day, Eff_empT_es_month] = ...
    effcalc (Pmax_empT_es, E_empT_es_day, E_empT_es_month, ...
    G_esti, G_esti_day, G_esti_month, mod_area, nYears, N);

% Calculating the efficiencies using measured outputs and pyranometer
% irradiance
[Eff_emp_py, Eff_emp_py_day, Eff_emp_py_month] = ...
    effcalc (Pmax_emp_py, E_emp_py_day, E_emp_py_month, ...
    G_pyr, G_pyr_day, G_pyr_month, mod_area, nYears, N);
[Eff_empT_py, Eff_empT_py_day, Eff_empT_py_month] = ...
    effcalc (Pmax_empT_py, E_empT_py_day, E_empT_py_month, ...
    G_pyr, G_pyr_day, G_pyr_month, mod_area, nYears, N);


%% PLOTTING THE RESULTS
```

```
Dstart_vec = datevec(Dstart);
year_vec = zeros(1,nYears);
for k=1:nYears
    year_vec(k)=Dstart_vec(1)+(k-1);
end

%% Using function 'plotter' to plot the monthly and daily measured and
%% predicted energy productions and efficiencies.

% Using the results based on ESTI irradiance
choice = input(['\n Do you want to make graphs where measured ',...
'\n values are compared with predicted ones,',...
'\n using values calculated with ESTI irradiance',...
'\n(for each module separately) (y/n)? '],'s');
if choice=='y'
    irrad_str='ESTI';
    irrad_fn='ESTI';
    plotter(E_month, E_emp_es_month, E_empT_es_month,...
        Eff_es_month, Eff_emp_es_month, Eff_empT_es_month, E_day, ...
        E_emp_es_day, E_empT_es_day, Eff_es_day, Eff_emp_es_day,...
        Eff_empT_es_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end

% Using the results based on pyranometer irradiance
choice = input(['\n Do you want to make graphs where measured ',...
'\n values are compared with predicted ones,',...
'\n using values calculated with pyranometer irradiance',...
'\n(for each module separately) (y/n)? '],'s');
if choice=='y'
    irrad_str='pyranometer';
    irrad_fn='pyran';
    plotter(E_month, E_emp_py_month, E_empT_py_month,...
        Eff_py_month, Eff_emp_py_month, Eff_empT_py_month, E_day, ...
        E_emp_py_day, E_empT_py_day, Eff_py_day, Eff_emp_py_day,...
        Eff_empT_py_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end

%% Using function 'plotter_meas_only' to plot just daily and monthly energy
%% productions and irradiances, making no comparison with the predicted
%% values

% Plotting the daily and monthly energy productions, using the energy
% calculated with Pmax_fill having gaps missing because of filtering
% filled with average ones.
choice = input(['\n Do you want to make graphs with',...
    ' measured (filled) energy production only (y/n)? '],'s');
if choice=='y'
    tit = 'energy production';
    ylbl = 'Energy [Wh]';
    filename = 'meas_E';
    plotter_meas_only(E_fill_day, E_fill_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end

% Plotting the daily and monthly ESTI irradiations
choice = input(['\n Do you want to make graphs with',...
    '\n ESTI irradiations only (y/n)? '],'s');
if choice=='y'
    tit = 'ESTI irradiance';
    ylbl = 'Irradiance [Wh/m^2]';
    filename = 'ESTI';
    plotter_meas_only(G_esti_day, G_esti_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end
```

```
% Plotting the daily and monthly pyranometer irradiations
choice = input(['\n Do you want to make graphs with',...
    '\n pyranometer irradiations only (y/n)? '],'s');
if choice=='y'
    tit = 'pyranometer irradiance';
    ylbl = 'Irradiance [Wh/m^2]';
    filename = 'pyr';
    plotter_meas_only(G_pyr_day, G_pyr_month, nYears, N, mod, tit,...
        ylbl, year_vec, filename)
end


%% Using the function 'mod_comparison' to make module comparison plots.

choice = input(['\n Do you want to make ',...
    'module comparison plots (y/n)? '],'s');
if choice=='y'
    irrad_str={'ESTI', 'pyranometer'};
    irrad_fn={'ESTI','pyr'};
    mod_comparison(Pmax_STC_lapss, Pmax_STC_out,...
        Pmax_STC_emp, Pmax_STC_label, E_day, G_esti_day, G_pyr_day, ...
        Tmod_av_day, mod, irrad_str, irrad_fn, N, year_vec, nYears);
end


%% SAVING THE DATA

choice = input(['\n Do you want to write the results',...
    ' into text-files (y/n)? '],'s');
if choice=='y'
    data_writer (E_day, E_month, E_emp_es_day, E_emp_es_month,...
        E_empT_es_day, E_empT_es_month, E_emp_py_day, E_emp_py_month,...
        E_empT_py_day, E_empT_py_month, ...
        Eff_es_day, Eff_es_month, Eff_py_day, Eff_py_month,...
        Eff_emp_es_day, Eff_emp_es_month, Eff_emp_py_day, ...
        Eff_emp_py_month, Eff_empT_es_day, Eff_empT_es_month, ...
        Eff_empT_py_day, Eff_empT_py_month, G_esti_day, G_esti_month,...
        G_pyr_day, G_pyr_month, Tmod_av_day, flag_array, mod,...
        nYears, N, date_array);
end


%%
pqclear(res);
pqfinish(myconn);
```

C.2 The function getdata_db_ene

```
function [fN, fnYears, nRows, fnDays, Dstart_ser, Dend_ser, fmod, fl, ...
    ftime, fG_esti, fG_pyr, fPmax, fTmod, fTamb]...
    = getdata_db_enr(myconn)

%------------------------------------------------------------------------%
% This function retireves the required .enr -data from the database for
% user-defined time period and set of modules
% and arranges the data into appropriate arrays.
%------------------------------------------------------------------------%

% Defining the time period to be analyzed
text_Dstart = 'The starting date (yyyy-mm-dd) of the data analysis: ';
text_Dend = 'The last date (yyyy-mm-dd) to be considered in the analysis: ';
Dstart = input(text_Dstart,'s');
Dend = input(text_Dend,'s');

% Converting the starting and ending dates into serial number format
```

```
Dstart_ser = datenum(Dstart,'yyyy-mm-dd');
Dend_ser = datenum(Dend,'yyyy-mm-dd');
fnDays = Dend_ser-Dstart_ser+1; % No of days in the dataset

% Converting the starting and ending dates into vector number format
Dstart_vec = datevec(Dstart_ser); Dend_vec=datevec(Dend_ser);
fnYears = Dend_vec(1)-Dstart_vec(1)+1; % No of years of measurement

% Defining the modules to be analyzed
fN = input('How many different modules are compared? ');

fmod = cell(1,fN);
res = zeros(1,fN,'uint32');
nFields = zeros(1,fN);
nRows = zeros(1,fN);

% Defining the data to be retrieved
for i=1:fN
   name = ['Name of the module ' num2str(i) '? '];
   fmod{i} = input(name,'s'); % Cell array containing names of the modules
   text = ['SELECT date,time,pmax,irradiance_esti,irradiance_pyran,',...
       't_mod,t_amb FROM %s WHERE date≥''%s'' AND date≤''%s''',...
       'ORDER BY date,time'];
   exeStr = sprintf(text,fmod{i},Dstart,Dend);
   res(i) = pqexec(myconn, exeStr);
   nFields(i) = pqnfields(res(i));
   nRows(i) = pqntuples(res(i));
end

%Displaying the size of the dataset
disp('Number of columns in the dataset of each module: ')
disp(nFields)
disp('Number of rows in the dataset of each module: ')
disp(nRows)

fl = max(nRows);
ftime = zeros(fl,6,fN);
fPmax = zeros(fl,1,fN);
fG_esti = zeros(fl,1,fN);
fG_pyr = zeros(fl,1,fN);
fTmod = zeros(fl,1,fN);
fTamb = zeros(fl,1,fN);

% Retrieving the data using the variable 'res' recieved above and
% creating the time and data arrays
for i=1:fN
    for j=1:nRows(i)
        timeStr = [pqgetvalue(res(i),j-1,0) ' ' pqgetvalue(res(i),j-1,1)];
        % Dates and times of each measurement:
        ftime(j,:,i) = datevec(timeStr,'yyyy-mm-dd HH:MM:SS');
        fPmax(j,:,i) = pqgetvalue(res(i),j-1,2);    % Pmax (W)
        fG_esti(j,:,i) = pqgetvalue(res(i),j-1,3);  % ESTI irrad., (W/m2)
        fG_pyr(j,:,i) = pqgetvalue(res(i),j-1,4);   % Pyran. irrad., (W/m2)
        fTmod(j,:,i) = pqgetvalue(res(i),j-1,5);    % (C)
        fTamb(j,:,i) = pqgetvalue(res(i),j-1,6);    % (C)
    end
end
```

C.3 The function data_filter_enr

```
function [fPmax, fPmax_emp_E, fPmax_empT_E, fPmax_emp_P, fPmax_empT_P,...
    fG_esti, fG_pyr, fFlag, fnPoints_deleted] ...
           = data_filter_enr(fPmax, fPmax_emp_E, fPmax_empT_E, ...
           fPmax_emp_P, fPmax_empT_P, fG_esti, fG_pyr, fPmax_STC_max,...
           fN, fnRows)
```

```
%---------------------------------------------------------------------------%
% This function filters out bad data points, i.e. sets empirical and
% measured Pmaxes and irradiance to zero if there is something strange
% in the data. The filtered points are marked in an array 'flag', and the
% value of flag(i,j) tells the reason why the point has been filtered
% according to the following:
%
% flag(j,i) = 1, if Pmax was zero in the original data
% flag(j,i) = 2, if Pmax<0 in the original data
% flag(j,i) = 3, if Pmax_emp or Pmax_empT <0 or complex in the
% original  data (this may occur at low temperatures if the empirical
% equation is 'bad')
% flag(j,i) = 4, if abs(Pmax-Pmax_emp)/max(Pmax,Pmax_emp) > 0.5
% flag(j,i) = 5, if abs(G_esti-G_pyr)/max(G_esti,G_pyr) > 0.3
% flag(j,i) = 6, if Pmax>Pmax_STC_max (the maximum STC Pmax) in the
% original data
%---------------------------------------------------------------------------%

fFlag=zeros(size(fPmax));
fnPoints_deleted = zeros(size(fPmax));

for i=1:fN
    for j=1:fnRows(i)
        if fPmax(j,:,i) == 0
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 1;
        end
```

```
if fPmax(j,:,i) < 0
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0; fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0; fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0; fG_pyr(j,:,i) = 0;
    fFlag(j,:,i) = 2;
end
if fPmax_emp_E(j,:,i)<0 || isreal(fPmax_emp_E(j,:,i))==0 || ...
        fPmax_empT_E(j,:,i)<0 || isreal(fPmax_empT_E(j,:,i))==0 ...
        || fPmax_emp_P(j,:,i)<0 || isreal(fPmax_emp_P(j,:,i))==0 ...
        || fPmax_empT_P(j,:,i)<0 || isreal(fPmax_empT_P(j,:,i))==0
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0;
    fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0;
    fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0;
    fG_pyr(j,:,i) = 0;
    fFlag(j,:,i) = 3;
end
if abs(fPmax(j,:,i)-fPmax_emp_E(j,:,i))/...
        max([fPmax(j,:,i) fPmax_emp_E(j,:,i)]) > 0.5 || ...
        abs(fPmax(j,:,i)-fPmax_emp_P(j,:,i))/...
        max([fPmax(j,:,i) fPmax_emp_P(j,:,i)]) > 0.5
    fPmax(j,:,i) = 0;
    fPmax_emp_E(j,:,i) = 0;
    fPmax_empT_E(j,:,i) = 0;
    fPmax_emp_P(j,:,i) = 0;
    fPmax_empT_P(j,:,i) = 0;
    fG_esti(j,:,i) = 0;
    fG_pyr(j,:,i) = 0;
    fFlag(j,:,i) = 4;
end
```

```
        if abs(fG_esti(j,:,i)-fG_pyr(j,:,i))/...
                max([fG_esti(j,:,i) fG_pyr(j,:,i)]) > 0.3
            fPmax(j,:,i) = 0;
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 5;
        end
        if fPmax(j,:,i) > fPmax_STC_max(i)
            fPmax(j,:,i) = 0;
            fPmax_emp_E(j,:,i) = 0;
            fPmax_empT_E(j,:,i) = 0;
            fPmax_emp_P(j,:,i) = 0;
            fPmax_empT_P(j,:,i) = 0;
            fG_esti(j,:,i) = 0;
            fG_pyr(j,:,i) = 0;
            fFlag(j,:,i) = 6;
        end
        % Calculating total the number of points deleted by filtering
        % (necessary to do separately, as there might be some overlapping
        % in the flags)
        fnPoints_deleted(j,:,i) = nnz(fPmax(j,:,i)==0);
    end
end
```

D Functions utilized by both main scripts

D.1 The function noct_estim

```
function fTmod_emp = noct_estim(fTmod, fTamb, fG, fN, fmod, ...
    fnRows, fG_str)

%--------------------------------------------------------------------------%
% This function calculates the values of the Nominal Operating Cell
% Temperature (NOCT) and thereafter the empirical module temperatures
% (Tmod_emp) using the following empirical equation:
% 'Tmod-Tamb=((NOCT-20)/800)*G'.
%--------------------------------------------------------------------------%


Tplot = fTmod-fTamb;
fTmod_emp = zeros(size(fTmod));

for i=1:fN
    figure;
    % Plotting the experimental data (ESTI irrad. vs. Tmod-Tamb)
    plot(fG(fG(1:fnRows(i),:,i)≠0,:,i),...
        Tplot(fG(1:fnRows(i),:,i)≠0,:,i),'.')
    hold on;
    % Least-squares fitting to the experimental data. 'Polyfit' returns
    % the coefficients of the polynomial in descending powers
    p = polyfit(fG(fG(1:fnRows(i),:,i)≠0,:,i),...
        Tplot(fG(1:fnRows(i),:,i)≠0,:,i),1);
    NOCT = p(1)*800+20;
    y0 = p(2);
    Tplot_fit = p(1)*fG(fG(1:fnRows(i),:,i)≠0,:,i)+y0;
    % Plotting the linear fit to the same graph with the experimental
    % data
    plot(fG(fG(1:fnRows(i),:,i)≠0,:,i),Tplot_fit,'r-')
    xlbl = sprintf('%s irradiance [W/m^2]',fG_str);
    xlabel(xlbl);
    ylabel('Tmod-Tamb [°C]');
    titleStr = ...
        sprintf('NOCT estimation using %s irradiance (%s)',...
        fG_str,fmod{i});
    title(titleStr);
    % Calculating empirical Tmod using the fitting coefficients
    fTmod_emp(1:fnRows(i),:,i) = ...
        ((NOCT-20)/800).*fG(1:fnRows(i),:,i)+fTamb(1:fnRows(i),:,i)+y0;
end
```

D.2 The function base_matrix

```
function [fbase_arr, M_d, dt_min] = base_matrix(fnDays, fDstart_ser,...
    fDend_ser, ftime, fData, fN, fl, fnRows)

%----------------------------------------------------------------------%
% This function organizes the data into an array base_array, in which
% every possible measurement point for the time period concerned is
% included, and the data for each module recides at their
% correct points.
%----------------------------------------------------------------------%

% Converting the time array (containing the dates and times of the
% measurement points of all the modules) into serial number format
time_ser = zeros(fl,fN);
```

```
for n=1:fN
    time_ser(:,n) = datenum(ftime(:,:,n));
end



%% Defining the measurement interval to use

% Building a vector of the days in the time period concerned
day = linspace(fDstart_ser, fDend_ser, fnDays); % In serial nr format
day_vec = datevec(day); % In vector format

% Calculating the mode measurement interval for each day and module
% separately into an array dt_ser_all.
% The loop collects the measurement points within each day by finding
% out the first and last row of the day (row1 and row2), and calculates
% time interval for the day as the mode time difference between the
% the measurement points. If there is only one measurement point within
% the day, the time interval is set to 4 minutes.
dt_ser_all = zeros(fnDays,fN);
for n=1:fN
    row1=1; row2=1;
    for d=1:fnDays
        while floor(time_ser(row2,n))==day(d)
            row2=row2+1;
            if floor(time_ser(row2,n))≠day(d)
                if (row2−row1)>1
                    dt_ser_all(d,n) = ...
                        mode(diff(time_ser(row1:(row2−1),n)));
                else
                    dt_ser_all(d,n) = datenum(0,0,0,0,4,0);
                end
                row1=row2;
                break
            end
            if row2==length(time_ser)
                break
            end
        end
    end
end

% The measurement interval for each day is taken as the minimum
% interval from all the modules' intervals (disregarding zeros)
dt_ser_day=zeros(fnDays,1);
for d=1:fnDays
    if any(dt_ser_all(d,:))==1
        dt_ser_day(d)=min(dt_ser_all(d,dt_ser_all(d,:)>0));
    end
end

% The daily meas. intervals as time vector:
dt_vec_day = datevec(dt_ser_day);
% The daily meas. interval in minutes:
dt_min_day = round(dt_vec_day(:,5)+dt_vec_day(:,6)/60);

% Calculating the mode time interval for the whole period
dt_min = mode(dt_min_day); % In minutes
dt_ser = mode(dt_ser_day); % As a serial number


% Communicating with the user on which time interval should be used

text = ['The mode time interval between the measurements in',...
    '\n the period concerned was %d minutes',...
    '\n and the days with that interval were (year month day)'];
sprintf(text,dt_min)
```

```
disp(day_vec(dt_min_day==dt_min,1:3))

text = ['The days (year month day) when mode time interval',...
    '\n was something else but %d minutes were',...
    '\n (the last column tells the mode time interval for the day)'];
sprintf(text,dt_min)
disp([day_vec(dt_min_day≠dt_min,1:3) dt_min_day(dt_min_day≠dt_min)])

text = 'Perform the calculations with the mode interval (y/n)? ';
mode_yn = input(text,'s');
if mode_yn == 'n'
    disp(['Re-perform the analysis in shorter time periods,'...
        'in which the mode time interval stays approximately constant.'])
    return
end


%% Creating the time stamp
% Creating a time stamp vector time_stamp, in which every possible
% measurement point from the concerned time period is stored, in order
% to have a common time stamp for all the modules.

% No of measurements within an hour.
M_h = 60/dt_min;
 % No of measurement points within a day, assuming a maximum measurement
 % period of 6-20 (14 hours).
M_d = M_h*14;

hour = zeros(M_d,1);
time_stamp = zeros(fnDays*M_d,1);

% Time for the earliest possible measurement.
Tstart=datenum(0,0,0,6,0,0);
```

```
for k=1:M_d
    hour(k) = Tstart+(k-1)*dt_ser;
end
for d=1:fnDays
    time_stamp((d-1)*M_d+1:d*M_d,1) = fDstart_ser+(d-1)+hour;
end

%% Creating the base array
% Creating a 3D array, in which all the possible measurement points and
% and data from all the modules discussed reside.
% The first column is the time stamp vector, same for all the modules.
% In the last columns of the array, the measurement points are marked
% with 1, while other elements in the columns equal to 0.

S=size(fData);
fbase_arr=[repmat(time_stamp,[1 1 fN]) zeros(fnDays*M_d,(S(2)+1),fN)];

for n=1:fN
    pointer=1;
    for i=1:fnRows(n)
        for j=pointer:fnDays*M_d
            if abs(time_stamp(j)-time_ser(i,n)) < dt_ser/2
                fbase_arr(j,2:(end-1),n) = fData(i,:,n);
                fbase_arr(j,end,n) = 1; % Marks the measurement points
                pointer = j+1;
                break
            end
        end
    end
end
```

D.3 The function energycalc

```
function [E_d, E_m, E_fill_d, E_fill_m, E_emp_es_d, E_emp_es_m,...
           E_empT_es_d, E_empT_es_m, E_emp_py_d, E_emp_py_m,...
           E_empT_py_d, E_empT_py_m, G_es_d, G_es_m, G_py_d, G_py_m,...
           Tmod_av_d, Tmod_av_m, flag_arr, date_all]...
          = energycalc (fbase_arr, fnDays, fnYears,...
          fDstart_ser, fDend_ser, fN, M_d, dt_min)

%-----------------------------------------------------------------------%
% This function calculates the daily and monthly values: measured
% and predicted energy productions, irradiances, average module
% temperatures and number of all the flagged points as well as
% number of measurement points.
%-----------------------------------------------------------------------%

% Distracting different variables from the base array, and arranging
% them into 3D arrays where each day has its own column.
Pmax = reshape(fbase_arr(:,2,:),M_d,fnDays,fN);
Pmax_emp_es = reshape(fbase_arr(:,3,:),M_d,fnDays,fN);
Pmax_empT_es = reshape(fbase_arr(:,4,:),M_d,fnDays,fN);
Pmax_emp_py = reshape(fbase_arr(:,5,:),M_d,fnDays,fN);
Pmax_empT_py = reshape(fbase_arr(:,6,:),M_d,fnDays,fN);
G_es = reshape(fbase_arr(:,7,:),M_d,fnDays,fN);
G_py = reshape(fbase_arr(:,8,:),M_d,fnDays,fN);
Tmod = reshape(fbase_arr(:,9,:),M_d,fnDays,fN);
flag = reshape(fbase_arr(:,10,:),M_d,fnDays,fN);
points_del = reshape(fbase_arr(:,11,:),M_d,fnDays,fN);
meas_points = reshape(fbase_arr(:,end,:),M_d,fnDays,fN);
```

```
%% Filling the gaps to the measured Pmax
% Calculating another Pmax, 'Pmax_fill', where gaps of 1−2 measurement
% points have been filled with average values of the surrounding
% measurement points.

Pmax_fill = Pmax;
fill_count = zeros(fnDays,fN);

for n=1:fN
    for d=1:fnDays
        for i=1:M_d
            if Pmax_fill(i,d,n)==0 && points_del(i,d,n)≠0
                if i>1 && i<M_d && Pmax_fill(i−1,d,n)≠0 &&...
                        Pmax_fill(i+1,d,n)≠0
                    Pmax_fill(i,d,n) = ...
                        mean([Pmax_fill(i−1,d,n) Pmax_fill(i+1,d,n)]);
                    fill_count(d,n) = fill_count(d,n)+1;
                elseif i>2 && i<M_d && Pmax_fill(i−2,d,n)≠0 &&...
                        Pmax_fill(i+1,d,n)≠0
                    Pmax_fill(i,d,n) = ...
                        mean([Pmax_fill(i−2,d,n) Pmax_fill(i+1,d,n)]);
                    fill_count(d,n) = fill_count(d,n)+1;
                elseif i>1 && i<(M_d−1) && Pmax_fill(i−1,d,n)≠0 &&...
                        Pmax_fill(i+2,d,n)≠0
                    Pmax_fill(i,d,n) = ...
                        mean([Pmax_fill(i−1,d,n) Pmax_fill(i+2,d,n)]);
                    fill_count(d,n) = fill_count(d,n)+1;
                end
            end
        end
    end
end
```

```
%% Calculating the daily energies, irradiances and measurement points.

E = reshape((dt_min/60)*sum(Pmax),fnDays,fN); % In Wh
E_fill = reshape((dt_min/60)*sum(Pmax_fill),fnDays,fN); % In Wh
E_emp_es = reshape((dt_min/60)*sum(Pmax_emp_es),fnDays,fN); % In Wh
E_empT_es = reshape((dt_min/60)*sum(Pmax_empT_es),fnDays,fN); % In Wh
E_emp_py = reshape((dt_min/60)*sum(Pmax_emp_py),fnDays,fN); % In Wh
E_empT_py = reshape((dt_min/60)*sum(Pmax_empT_py),fnDays,fN); % In Wh
G_es = reshape((dt_min/60)*sum(G_es),fnDays,fN); % In Wh/m2
G_py = reshape((dt_min/60)*sum(G_py),fnDays,fN); % In Wh/m2
meas_points = reshape(sum(meas_points),fnDays,fN);
points_del = reshape(sum(points_del),fnDays,fN);


%% Calculating the daily average module temperatures
% Measurement points where Tmod=0 are disregarded

Tmod_dAv = zeros(fnDays,fN);
for n=1:fN
    for d=1:fnDays
        Tmod_dAv(d,n) = mean(Tmod(Tmod(:,d,n)≠0,d,n)); % In C
    end
    Tmod_dAv(isnan(Tmod_dAv(:,n))==1,n)=0;
end


%% Calculating the number of flagged points for each day

flag1 = zeros(fnDays,fN);
flag2 = zeros(fnDays,fN);
flag3 = zeros(fnDays,fN);
```

```
flag4 = zeros(fnDays,fN);
flag5 = zeros(fnDays,fN);
flag6 = zeros(fnDays,fN);

for n=1:fN
    for d=1:fnDays
        % The number of elements that equal to 1 in one column of
        % flag array (i.e. one day):
        flag1(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-1);
        % No of elements that equal to 2 per column:
        flag2(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-2);
        % No of elements that equal to 3 per column:
        flag3(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-3);
        % etc...
        flag4(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-4);
        flag5(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-5);
        flag6(d,n) = numel(flag(:,d,n))-nnz(flag(:,d,n)-6);
    end
end


%% Locating the calculated values into appropriate arrays

flag1_d_vec=zeros(12*31*fnYears,fN);
flag2_d_vec=zeros(12*31*fnYears,fN);
flag3_d_vec=zeros(12*31*fnYears,fN);
flag4_d_vec=zeros(12*31*fnYears,fN);
flag5_d_vec=zeros(12*31*fnYears,fN);
flag6_d_vec=zeros(12*31*fnYears,fN);
meas_points_d_vec=zeros(12*31*fnYears,fN);
points_del_d_vec=zeros(12*31*fnYears,fN);
E_d_vec = zeros(12*31*fnYears,fN);
E_fill_d_vec = zeros(12*31*fnYears,fN);
```

```
E_emp_es_d_vec = zeros(12*31*fnYears,fN);
E_empT_es_d_vec = zeros(12*31*fnYears,fN);
E_emp_py_d_vec = zeros(12*31*fnYears,fN);
E_empT_py_d_vec = zeros(12*31*fnYears,fN);
G_es_d_vec = zeros(12*31*fnYears,fN);
G_py_d_vec = zeros(12*31*fnYears,fN);
Tmod_dAv_vec = zeros(12*31*fnYears,fN);

% Creating date arrays for all possible dates in the years to be
% considered, and for all the dates in the measurement points.
start_date = datevec(fDstart_ser);
y_start = start_date(1);
days_in_12months = repmat((1:31)',12*fnYears,1);
months_in_1year = ...
    reshape((repmat((1:12)',fnYears,31))',12*31*fnYears,1);
year_array = zeros(fnYears*12*31,1);
for y=1:fnYears
    year_array(1+(y-1)*31*12:y*31*12) = ...
        ones(12*31,1)*(y_start+y-1);
end
date_all =[year_array months_in_1year days_in_12months];

date_meas = datevec(fDstart_ser:fDend_ser);

% Locating the calculated daily values into correct dates in  2D
% (12*31*nYears)xN -arrays
pointer=1;
for i=1:fnDays
    for j=pointer:12*31*fnYears
        if all(date_meas(i,1:3)==date_all(j,:))==1
            E_d_vec(j,:) = E(i,:);
            E_fill_d_vec(j,:) = E_fill(i,:);
            E_emp_es_d_vec(j,:) = E_emp_es(i,:);
            E_empT_es_d_vec(j,:) = E_empT_es(i,:);
            E_emp_py_d_vec(j,:) = E_emp_py(i,:);
            E_empT_py_d_vec(j,:) = E_empT_py(i,:);
            G_es_d_vec(j,:) = G_es(i,:);
            G_py_d_vec(j,:) = G_py(i,:);
            Tmod_dAv_vec(j,:) = Tmod_dAv(i,:);
            flag1_d_vec(j,:) = flag1(i,:);
            flag2_d_vec(j,:) = flag2(i,:);
            flag3_d_vec(j,:) = flag3(i,:);
            flag4_d_vec(j,:) = flag4(i,:);
            flag5_d_vec(j,:) = flag5(i,:);
            flag6_d_vec(j,:) = flag6(i,:);
            meas_points_d_vec(j,:) = meas_points(i,:);
            points_del_d_vec(j,:) = points_del(i,:);
            pointer=j+1;
            break
        end
    end
end

% Reshaping the 2D (12*31*nYears)xN -arrays into 4D 31x12x(nYears)xN
% ones...
E_d_arr = reshape(E_d_vec,31,12,fnYears,fN);
E_fill_d_arr = reshape(E_fill_d_vec,31,12,fnYears,fN);
E_emp_es_d_arr = reshape(E_emp_es_d_vec,31,12,fnYears,fN);
E_empT_es_d_arr = reshape(E_empT_es_d_vec,31,12,fnYears,fN);
E_emp_py_d_arr = reshape(E_emp_py_d_vec,31,12,fnYears,fN);
E_empT_py_d_arr = reshape(E_empT_py_d_vec,31,12,fnYears,fN);
G_es_d_arr = reshape(G_es_d_vec,31,12,fnYears,fN);
G_py_d_arr = reshape(G_py_d_vec,31,12,fnYears,fN);
Tmod_dAv_arr = reshape(Tmod_dAv_vec,31,12,fnYears,fN);

E_d = zeros(12,31,fnYears,fN);
```

```
E_fill_d = zeros(12,31,fnYears,fN);
E_emp_es_d = zeros(12,31,fnYears,fN);
E_empT_es_d = zeros(12,31,fnYears,fN);
E_emp_py_d = zeros(12,31,fnYears,fN);
E_empT_py_d = zeros(12,31,fnYears,fN);
G_es_d = zeros(12,31,fnYears,fN);
G_py_d = zeros(12,31,fnYears,fN);
Tmod_av_d = zeros(12,31,fnYears,fN);
flag_arr = zeros(12*31*fnYears,8,fN);

%...and into 12x31x(nYears)xN ones.
for n=1:fN
    for y=1:fnYears
        E_d(:,:,y,n) = E_d_arr(:,:,y,n)';
        E_fill_d(:,:,y,n) = E_fill_d_arr(:,:,y,n)';
        E_emp_es_d(:,:,y,n) = E_emp_es_d_arr(:,:,y,n)';
        E_empT_es_d(:,:,y,n) = E_empT_es_d_arr(:,:,y,n)';
        E_emp_py_d(:,:,y,n) = E_emp_py_d_arr(:,:,y,n)';
        E_empT_py_d(:,:,y,n) = E_empT_py_d_arr(:,:,y,n)';
        G_es_d(:,:,y,n) = G_es_d_arr(:,:,y,n)';
        G_py_d(:,:,y,n) = G_py_d_arr(:,:,y,n)';
        Tmod_av_d(:,:,y,n) = Tmod_dAv_arr(:,:,y,n)';
    end
    % The flagged points, as well as the number of measurement points
    % and points deleted are arranged into one array
    flag_arr(:,:,n) = [flag1_d_vec(:,n) flag2_d_vec(:,n) ...
        flag3_d_vec(:,n) flag4_d_vec(:,n) flag5_d_vec(:,n) ...
        flag6_d_vec(:,n) points_del_d_vec(:,n) meas_points_d_vec(:,n)];
end


%% Calculating the monthly values
```

```
E_m = zeros(12,fnYears,fN);
E_fill_m = zeros(12,fnYears,fN);
E_emp_es_m = zeros(12,fnYears,fN);
E_empT_es_m = zeros(12,fnYears,fN);
E_emp_py_m = zeros(12,fnYears,fN);
E_empT_py_m = zeros(12,fnYears,fN);
G_es_m = zeros(12,fnYears,fN);
G_py_m = zeros(12,fnYears,fN);
Tmod_av_m = zeros(12,fnYears,fN);

for n=1:fN
    for y =1:fnYears
        for m=1:12
            E_m(m,y,n) = sum(E_d(m,:,y,n)); % In Wh
            E_fill_m(m,y,n) = sum(E_fill_d(m,:,y,n)); % In Wh
            E_emp_es_m(m,y,n) = sum(E_emp_es_d(m,:,y,n)); % In Wh
            E_empT_es_m(m,y,n) = sum(E_empT_es_d(m,:,y,n)); % In Wh
            G_es_m(m,y,n) = sum(G_es_d(m,:,y,n)); % In Wh/m2
            E_emp_py_m(m,y,n) = sum(E_emp_py_d(m,:,y,n)); % In Wh
            E_empT_py_m(m,y,n) = sum(E_empT_py_d(m,:,y,n)); % In Wh
            G_py_m(m,y,n) = sum(G_py_d(m,:,y,n)); % In Wh/m2
            Tmod_av_m(m,y,n) = ...
                mean(Tmod_av_d(m,Tmod_av_d(m,:,y,n)≠0,y,n)); % In C
        end
        Tmod_av_m(isnan(Tmod_av_m(:,y,n))==1,y,n)=0;
    end
end
```

D.4 The function plotter

```
function plotter(Var1_m, Var2_m, Var3_m, Eff1_m, Eff2_m, Eff3_m,...
              Var1_d, Var2_d, Var3_d, Eff1_d, Eff2_d, Eff3_d,...
              fmod, G_str, G_fn, fN, year, fnYears)

%----------------------------------------------------------------------%
% This function makes the different plots where measured results are
% compared with the predicted ones. To make the plots, the function
% uses a nested function 'bar_plotter'.
%----------------------------------------------------------------------%

close all

disp(['The plots will be saved in a ''plots'', ',...
    'created in the current directory if necessary.'])
mkdir('plots');



%% Monthly energy output and efficiency comparison between measured
%% and predicted values
% Plotted for each module and year separately.

ylbl_en = 'Energy [Wh]';
ylbl_eff = 'Efficiency (%)';
xlbl_m = 'Month';

x1_month = 1:12;
x2_month = 1.25:12.25;
x3_month = 1.5:12.5;

disp(['The monthly energy production plots ',...
    'will be saved as ''_monthly_outputs_'', '])
disp(['with module name as prefix and irradiance type ',...
    '(ESTI, pyr, ESTIcorr, pyr_corr) and year as postfixes. '])
disp(['The monthly efficiency plots will be saved as ',...
    '''_monthly_eff_'', with the same affixes.'])

for n=1:fN
    for k=1:fnYears
        % Writing the title for each plot
        tit_en = sprintf(['Monthly measured and predicted ',...
            'energy production \n (%s, %d, predicted with %s',...
            ' irradiation)'],fmod{n},year(k),G_str);
        tit_eff = sprintf(['Monthly measured and predicted ',...
            'efficiencies \n(%s, %d, predicted with %s irradiation)'],...
            fmod{n},year(k),G_str);
        % Wtiting the path and filename for each plot
        fn_eff = sprintf('plots\\%s_monthly_eff_%s_%d',...
            fmod{n},G_fn,year(k));
        fn_en = sprintf('plots\\%s_monthly_outputs_%s_%d',...
            fmod{n},G_fn,year(k));
        % Making the plots using 'bar_plotter'
        bar_plotter (Var1_m(:,k,n), Var2_m(:,k,n), Var3_m(:,k,n),...
            x1_month, x2_month, x3_month, tit_en, fn_en,...
            ylbl_en, xlbl_m);
        bar_plotter(Eff1_m(:,k,n), Eff2_m(:,k,n), Eff3_m(:,k,n), ...
            x1_month, x2_month, x3_month, tit_eff, fn_eff, ...
            ylbl_eff, xlbl_m);
    end
end


%% Daily average and monthly 'equalized' output comparison between
%% measured and predicted values
% The daily average means just the daily average energy production
% for each month.
% In the 'monthly equalized' comparison, the daily average energy
% production of each month is multiplied with the no of days in the
% month in order to get an approximation for the monthly energy
% production if there were measurements on every day of the month.

disp(['The monthly equalized output plots will be saved as ',...
    '''_monthly_equalized_'', with the same affixes.'])
disp(['The daily average energy production plots will be saved ',...
    'as ''_daily_av_'', with the same affixes.'])

day_count=zeros(12,fnYears,fN);
Var1_Dave=zeros(12,fnYears,fN);
Var2_Dave=zeros(12,fnYears,fN);
Var3_Dave=zeros(12,fnYears,fN);
Var1_eq=zeros(12,fnYears,fN);
Var2_eq=zeros(12,fnYears,fN);
Var3_eq=zeros(12,fnYears,fN);

days_month=[31 28 31 30 31 30 31 31 30 31 30 31];

for n=1:fN
    for k=1:fnYears
        for i=1:12
            % Calculating the number of measurement days within each
            % month (it's enough to check the nonzero days in measured
            % outputs in 'Var1_d')
            day_count(i,k,n) = nnz(Var1_d(i,:,k,n));
            if day_count(i,k,n)≠0
                % Calculating the daily average outputs
                Var1_Dave(i,k,n) = Var1_m(i,k,n)./day_count(i,k,n);
                Var2_Dave(i,k,n) = Var2_m(i,k,n)./day_count(i,k,n);
                Var3_Dave(i,k,n) = Var3_m(i,k,n)./day_count(i,k,n);
                % Calculating the monthly equalized outputs
                Var1_eq(i,k,n) = Var1_Dave(i,k,n).*days_month(i);
                Var2_eq(i,k,n) = Var2_Dave(i,k,n).*days_month(i);
                Var3_eq(i,k,n) = Var3_Dave(i,k,n).*days_month(i);
            end
        end
        % Writing the title for each plot
        tit_Dav = sprintf(['Daily average measured and predicted ',...
            'energy production \n (%s, %d, predicted with ',...
            '%s irradiation)'],fmod{n},year(k),G_str);
        tit_Meq = sprintf(['Monthly equalized measured and ',...
            'predicted energy outputs \n',...
            '(%s, %d, predicted with %s irradiation)'],...
            fmod{n},year(k),G_str);
        % Wtiting the path and filename for each plot
        fn_Meq = sprintf('plots\\%s_monthly_equalized_%s_%d',...
            fmod{n},G_fn,year(k));
        fn_Dav = sprintf('plots\\%s_daily_av_%s_%d',...
            fmod{n},G_fn,year(k));
        % Making the plots using 'bar_plotter'
        bar_plotter (Var1_Dave(:,k,n), Var2_Dave(:,k,n), ...
            Var3_Dave(:,k,n), x1_month, x2_month, x3_month,...
            tit_Dav, fn_Dav, ylbl_en, xlbl_m);
        bar_plotter (Var1_eq(:,k,n),Var2_eq(:,k,n),Var3_eq(:,k,n),...
            x1_month, x2_month, x3_month, tit_Meq, fn_Meq,...
            ylbl_en, xlbl_m);
    end
end


%% Daily outputs and efficiencies
% One graph per month, for each module and year separately.

disp(['The daily energy production plots will be saved as ',...
    '''_daily_outputs_'' with the same affixes, ',...
    'accompained with the name of the month.'])
```

```
disp(['The daily efficiency plots will be saved as ',...
    '''_daily_eff_'' with the same affixes.'])

x1_day=1:31;
x2_day=1.25:31.25;
x3_day=1.5:31.5;
xlbl_d='Day';

for n=1:fN
    for k=1:fnYears
        for i=1:12
            plot_yes = any(Var1_d(i,:,k,n));
            if plot_yes==1
                j = int2str(i);
                month = month_teller(j);
                % Writing the title for each plot
                tit_en = ...
                    sprintf(['Daily measured and predicted ',...
                    'energy production of %s in %s %d  \n',...
                    '(predicted with %s irradiation)'],...
                    fmod{n},month,year(k),G_str);
                tit_eff = ...
                    sprintf(['Daily measured and predicted ',...
                    'efficiencies of %s in %s %d  \n',...
                    '(predicted with %s irradiation)'],...
                    fmod{n},month,year(k),G_str);
                % Wtiting the path and filename for each plot
                fn_en = sprintf('plots\\%s_daily_outputs_%s_%s_%d',...
                    fmod{n},G_fn,month,year(k));
                fn_eff = sprintf('plots\\%s_daily_eff_%s_%s_%d',...
                    fmod{n},G_fn,month,year(k));
                % Making the plots using 'bar_plotter'
                bar_plotter(Var1_d(i,:,k,n), Var2_d(i,:,k,n), ...
                    Var3_d(i,:,k,n),x1_day, x2_day, x3_day,...
                    tit_en, fn_en, ylbl_en, xlbl_d);
                bar_plotter(Eff1_d(i,:,k,n),...
                    Eff2_d(i,:,k,n), Eff3_d(i,:,k,n),...
                    x1_day, x2_day, x3_day, tit_eff, fn_eff,...
                    ylbl_eff, xlbl_d);
            end
        end
    end
end


%% Bar plotter

    function bar_plotter(Var1, Var2, Var3, x1, x2, x3, tit, fn,...
            ylbl, xlbl)

        % This nested function does the bar plots, where measured and
        % empirical energy outputs and efficiencies are compared.

        figure;
        Max = max([max(Var1) max(Var2) max(Var3)]);
        ymax = Max + Max/10;
        xmax = x3(end)+0.25;
        bar(x1,Var1,0.25,'b');
        hold on;
        bar(x2,Var2,0.25,'r');
        hold on;
        bar(x3,Var3,0.25,'y');
        title(tit,'fontsize',14);
        legend('Measured','Predicted, using T_{mod}',...
            'Predicted, using T_{amb}','Location','South');
        xlabel(xlbl,'fontsize',14);
        ylabel(ylbl,'fontsize',14);
        axis([0.75 xmax 0 ymax]);
        set(gca,'FontSize',14);
        grid on;
        print(gcf,'-dpng',fn)

    end

end
```

D.5 The function plotter_meas_only

```
function plotter_meas_only(Var_d, Var_m, fnYears, fN, fmod,...
    tit_str, ylbl, year, fn)

close all

disp(['The module comparison plots will be saved in a folder ',...
    '''plots'', created in the current directory if necessary.'])
mkdir('plots');

monthly = sprintf(['\n The monthly plots will be saved as ',...
    '''_monthly_%s_'', \n with the module name as prefix ',...
    'and year as postfix'],fn);
disp(monthly)

daily = sprintf(['\n The daily plots will be saved as ',...
    '''_daily_%s_'', \n with the same affixes'],fn);
disp(daily)

% Monthly plots
```

```
x_month = 1:12;
for n=1:fN
    for y=1:fnYears
        figure
        ymax = max(Var_m(:,y,n)) + max(Var_m(:,y,n))/10;
        bar(x_month,Var_m(:,y,n),'b');
        colormap cool
        tit = sprintf('Monthly %s for %s, in %d',...
            tit_str,fmod{n},year(y));
        title(tit,'fontsize',14);
        xlabel('Month','fontsize',14);
        ylabel(ylbl,'fontsize',14);
        axis([0.5 12.5 0 ymax]);
        set(gca,'FontSize',14);
        grid on;
        path = sprintf('plots\\%s_monthly_%s_%d',...
            fmod{n},fn,year(y));
        print(gcf,'-dpng',path)
    end
end


% Daily plots
x_day = 1:31;
for n=1:fN
    for y=1:fnYears
        for m=1:12
            plot_yes = any(Var_d(m,:,y,n));
            if plot_yes==1
                figure
                j = int2str(m);
                month = month_teller(j);
                ymax = max(Var_d(m,:,y,n)) + max(Var_d(m,:,y,n))/10;
                bar(x_day,Var_d(m,:,y,n),'b');
                colormap cool
                tit = sprintf('Daily %s for %s, in %s %d',...
                    tit_str,fmod{n},month,year(y));
                title(tit,'fontsize',14);
                xlabel('Month','fontsize',14);
                ylabel(ylbl,'fontsize',14);
                axis([0.5 31.5 0 ymax]);
                set(gca,'FontSize',14);
                grid on;
                path = sprintf('plots\\%s_daily_%s_%s_%d',...
                    fmod{n},fn,month,year(y));
                print(gcf,'-dpng',path)
            end
        end
    end
end
```

D.6 The function mod_comparison

```
function mod_comparison (Pmax_STClps, Pmax_STCo, Pmax_STCe, ...
    Pmax_STClbl, E_d, Irr1_d, Irr2_d, Tmod_av_d,...
    fmod, irr_str, irr_fn, fN, year, fnYears)

%-----------------------------------------------------------------------%
% This function makes the plots where energy outputs, average module
% temperatures and irradiances of the different modules are compared.
% Energy outputs are compared as relative to three different STC
% power values: LAPSS, outdoor and performance surface STC power value.
% The plots are made using a nested function 'bar_plotter'.
%-----------------------------------------------------------------------%
```

```
close all

disp(['The module comparison plots will be saved in a folder ',...
    '''plots'', created in the current directory if necessary.'])
mkdir('plots');

disp(['The plots where relative outputs of the modules are ',...
    'compared, are saved as ''mod_comp_'', '])
disp('with the used STC power, number of modules and year as postfixes.')

disp(['The plots where irradiances are compared, ',...
    'are saved as ''irr_comp_'', with the used irradiance,'])
disp(' number of modules and year as postfixes.')

disp(['The plots where module temperatures are compared, ',...
    'are saved as ''tmod_comp_'', '])
disp('with the number of modules and year as postfixes.')


%% Disregarding the days, when there were no measurements from all the
%% modules

% In order to be able to compare the modules properly, the value of
% considered variable (output energy/irradiance/module temperature) is
% set to zero for all the modules, if it is zero for some of the modules.

days_meas=zeros(12,fnYears);

for i=1:fnYears
    for j=1:12
        for k=1:31
            if all(E_d(j,k,i,:))==0
                Irr1_d(j,k,i,:) = 0;
                Irr2_d(j,k,i,:) = 0;
                E_d(j,k,i,:) = 0;
                Tmod_av_d(j,k,i,:) = 0;
            else
                days_meas(j,i) = days_meas(j,i)+1;
            end
        end
    end
end

disp(['Number of days in each month, when there were measurement ',...
    'data from all the modules: '])
disp(days_meas)


%% Calculating the daily average values

Irr1_Dav=zeros(12,fnYears,fN); Irr2_Dav=zeros(12,fnYears,fN);
E_normLPS_Dav=zeros(12,fnYears,fN);
E_normLBL_Dav=zeros(12,fnYears,fN);
E_normO_Dav=zeros(12,fnYears,fN);
E_normE_Dav=zeros(12,fnYears,fN);
Tmod_Dav=zeros(12,fnYears,fN);

legend_LPS = cell(1,fN);
legend_LBL = cell(1,fN);
legend_O = cell(1,fN);
legend_E = cell(1,fN);

for n=1:fN
    for i=1:fnYears
        % Calculating the daily average values for months where there
        % were measurements.
        loc_nnz = find(days_meas(:,i)≠0);
        Irr1_Dav(loc_nnz,i,n) = ...
            sum(Irr1_d(loc_nnz,:,i,n),2)./days_meas(loc_nnz,i);
        Irr2_Dav(loc_nnz,i,n) = ...
            sum(Irr2_d(loc_nnz,:,i,n),2)./days_meas(loc_nnz,i);
        E_normLPS_Dav(loc_nnz,i,n) = ...
            sum(E_d(loc_nnz,:,i,n),2)./...
        (days_meas(loc_nnz,i)*Pmax_STClps(n));
        E_normLBL_Dav(loc_nnz,i,n) = ...
            sum(E_d(loc_nnz,:,i,n),2)./(days_meas(loc_nnz,i)...
            *Pmax_STClbl(n));
        E_normO_Dav(loc_nnz,i,n) = ...
            sum(E_d(loc_nnz,:,i,n),2)./...
            (days_meas(loc_nnz,i)*Pmax_STCo(n));
        E_normE_Dav(loc_nnz,i,n) = ...
            sum(E_d(loc_nnz,:,i,n),2)./...
            (days_meas(loc_nnz,i)*Pmax_STCe(n));
        Tmod_Dav(loc_nnz,i,n) = ...
            sum(Tmod_av_d(loc_nnz,:,i,n),2)./days_meas(loc_nnz,i);
    end
    % Writing legends for the graphs
    legend_LPS{n} = sprintf('%s, STC Pmax=%3.1fW',...
        fmod{n},Pmax_STClps(n));
    legend_LBL{n} = sprintf('%s, STC Pmax=%3.1fW',...
        fmod{n},Pmax_STClbl(n));
    legend_O{n} = sprintf('%s, STC Pmax=%3.1fW',...
        fmod{n},Pmax_STCo(n));
    legend_E{n} = sprintf('%s, STC Pmax=%3.1fW',...
        fmod{n},Pmax_STCe(n));
end

% Writing titles for the graphs
tit_LPS = sprintf(['Daily average energy production ',...
    'relative to indoor measured STC power']);
tit_LBL = sprintf(['Daily average energy production ',...
    'relative to STC power \n rated by the manufacturer (label)']);
tit_O = sprintf(['Daily average energy production ',...
    'relative to outdoor STC power']);
tit_E = ['Daily average energy production relative to ',...
    'performance surface STC-power'];
tit_irr1 = sprintf('Daily average %s irradiations',irr_str{1});
tit_irr2 = sprintf('Daily average %s irradiations',irr_str{2});
tit_Tmod = 'Monthly average module temperatures';

% Writing filenames for the graphs
fn_LPS='mod_comp_STC_LAPSS_';
fn_LBL='mod_comp_STC_label_';
fn_O='mod_comp_STC_out_';
fn_E='mod_comp_STC_emp_';
fn_irr1=sprintf('irr_comp_%s_',irr_fn{1});
fn_irr2=sprintf('irr_comp_%s_',irr_fn{2});
fn_Tmod='tmod_comp_';

% Writing y-axis labels for the graphs
ylbl_E='[Wh/W_{peak}]';
ylbl_I='Irradiation [Wh/m^2]';
ylbl_Tmod='Temperature [^oC]';

% Making the graphs using 'bar_plotter'
bar_plotter(E_normLPS_Dav, tit_LPS, fn_LPS, ylbl_E, legend_LPS);
bar_plotter(E_normLBL_Dav,tit_LBL,fn_LBL,ylbl_E, legend_LBL);
bar_plotter(E_normO_Dav,tit_O,fn_O,ylbl_E, legend_O);
bar_plotter(E_normE_Dav, tit_E, fn_E,ylbl_E, legend_E);
bar_plotter(Irr1_Dav, tit_irr1, fn_irr1, ylbl_I, fmod);
bar_plotter(Irr2_Dav, tit_irr2, fn_irr2, ylbl_I, fmod);
bar_plotter(Tmod_Dav, tit_Tmod, fn_Tmod, ylbl_Tmod, fmod);
```

```
%% Module comparison bar plots

    function bar_plotter(Var, tit_str, fn, ylbl, leg)

        % This nested function does the module comparison bar plots.

        bar_color={'b' 'r' 'g' 'y'};
        s=1/(fN+1);
        x=zeros(12,fN);

        for ii=1:fnYears
            hold off
            figure
            for nn=1:fN
                hold on
                x(:,nn)=(1+(nn-1)*s):(12+(nn-1)*s);
                bar(x(:,nn),Var(:,ii,nn),s,bar_color{nn});
            end
            xmax=x(end,end)+s;
            Max = max(max(Var(:,ii,:)));
            ymax = Max + Max/10;
            axis([0.75 xmax 0 ymax]);
            tit=sprintf('%s (%d)',tit_str,year(ii));
            title(tit,'fontsize',14);
            legend(leg,'Location','South');
            xlabel('Month','fontsize',14);
            ylabel(ylbl,'fontsize',14);
            set(gca,'FontSize',14);
            box on;
            grid on;
            path=sprintf('plots\\%s%dmod_%d',fn,fN,year(ii));
            print(gcf,'-dpng',path)
        end

    end

end
```

D.7 The function data_writer

```
function data_writer (E_d, E_m, E_emp_es_d, E_emp_es_m,...
    E_empT_es_d, E_empT_es_m, E_emp_py_d, E_emp_py_m,...
    E_empT_py_d, E_empT_py_m, ...
    Eff_es_d, Eff_es_m, Eff_py_d, Eff_py_m,...
    Eff_emp_es_d, Eff_emp_es_m, Eff_emp_py_d, Eff_emp_py_m,...
    Eff_empT_es_d, Eff_empT_es_m, Eff_empT_py_d, Eff_empT_py_m,...
    G_es_d, G_es_m, G_py_d, G_py_m,...
    Tmod_d, Tmod_m, flag_arr, fmod, fnYears, fN, date_arr)

%--------------------------------------------------------------------%
% This function writes the daily and monthly data into text files.
%--------------------------------------------------------------------%

disp(['The plots will be saved in a folder ''results'', ',...
    'created in the current directory if necessary.'])
mkdir('results');


%% Writing the daily values

flag1_Darray = reshape(flag_arr(:,1,:),12*31*fnYears,fN);
flag2_Darray = reshape(flag_arr(:,2,:),12*31*fnYears,fN);
flag3_Darray = reshape(flag_arr(:,3,:),12*31*fnYears,fN);
```

```
flag4_Darray = reshape(flag_arr(:,4,:),12*31*fnYears,fN);
flag5_Darray = reshape(flag_arr(:,5,:),12*31*fnYears,fN);
flag6_Darray = reshape(flag_arr(:,6,:),12*31*fnYears,fN);
points_del_Darray = reshape(flag_arr(:,7,:),12*31*fnYears,fN);
meas_points_Darray = reshape(flag_arr(:,8,:),12*31*fnYears,fN);

G_es_Darray=zeros(12*31*fnYears,fN);
G_py_Darray=zeros(12*31*fnYears,fN);
E_Darray=zeros(12*31*fnYears,fN);
E_emp_es_Darray=zeros(12*31*fnYears,fN);
E_empT_es_Darray=zeros(12*31*fnYears,fN);
E_emp_py_Darray=zeros(12*31*fnYears,fN);
E_empT_py_Darray=zeros(12*31*fnYears,fN);
Eff_es_Darray=zeros(12*31*fnYears,fN);
Eff_py_Darray=zeros(12*31*fnYears,fN);
Eff_emp_es_Darray=zeros(12*31*fnYears,fN);
Eff_empT_es_Darray=zeros(12*31*fnYears,fN);
Eff_emp_py_Darray=zeros(12*31*fnYears,fN);
Eff_empT_py_Darray=zeros(12*31*fnYears,fN);
T_Darray=zeros(12*31*fnYears,fN);

Array_day=zeros(12*31*fnYears,22,fN);

for n=1:fN
    for y=1:fnYears
        G_es_Darray(1+(y-1)*12*31:y*12*31,n) ...
            = reshape(G_es_d(:,:,y,n)',12*31,1);
        G_py_Darray(1+(y-1)*12*31:y*12*31,n) ...
            = reshape(G_py_d(:,:,y,n)',12*31,1);
        E_Darray(1+(y-1)*12*31:y*12*31,n) ...
            = reshape(E_d(:,:,y,n)',12*31,1);
        E_emp_es_Darray(1+(y-1)*12*31:y*12*31,n) ...
            = reshape(E_emp_es_d(:,:,y,n)',12*31,1);
        E_empT_es_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(E_empT_es_d(:,:,y,n)',12*31,1);
        E_emp_py_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(E_emp_py_d(:,:,y,n)',12*31,1);
        E_empT_py_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(E_empT_py_d(:,:,y,n)',12*31,1);
        Eff_es_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_es_d(:,:,y,n)',12*31,1);
        Eff_py_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_py_d(:,:,y,n)',12*31,1);
        Eff_emp_es_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_emp_es_d(:,:,y,n)',12*31,1);
        Eff_empT_es_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_empT_es_d(:,:,y,n)',12*31,1);
        Eff_emp_py_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_emp_py_d(:,:,y,n)',12*31,1);
        Eff_empT_py_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Eff_empT_py_d(:,:,y,n)',12*31,1);
        T_Darray(1+(y−1)*12*31:y*12*31,n) ...
            = reshape(Tmod_d(:,:,y,n)',12*31,1);
    end

    Array_day(:,:,n)=[G_es_Darray(:,n) G_py_Darray(:,n) ...
        E_Darray(:,n) E_emp_es_Darray(:,n) E_empT_es_Darray(:,n)...
        E_emp_py_Darray(:,n) E_empT_py_Darray(:,n)...
        Eff_es_Darray(:,n) Eff_emp_es_Darray(:,n) ...
        Eff_empT_es_Darray(:,n) Eff_py_Darray(:,n) ...
        Eff_emp_py_Darray(:,n) Eff_empT_py_Darray(:,n) T_Darray(:,n)...
        flag1_Darray(:,n) flag2_Darray(:,n) flag3_Darray(:,n)...
        flag4_Darray(:,n) flag5_Darray(:,n) flag6_Darray(:,n)...
        points_del_Darray(:,n) meas_points_Darray(:,n)];
end
```

```
fn = input(['\n Filename in which you want to store the daily ',...
    'results \n (module name will be added as a postfix)? '],'s');
for n=1:fN
    filename=sprintf('results\\%s_%s.txt',fn,fmod{n});
    fid = fopen(filename,'wt');
    % Writing the header to the file 'filename'
    fprintf(fid,['%% Date \t G_esti[Wh/m2] \t G_pyr[Wh/m2] ',...
        '\t E[Wh] \t E_emp_esti[Wh] \t E_empT_esti[Wh] \t',...
        'E_emp_pyr[Wh] \t E_empT_pyr[Wh] \t Eff_esti(%%) \t',...
        'Eff_emp_esti(%%) \t Eff_empT_esti(%%) \t Eff_pyr(%%) \t',...
        'Eff_emp_pyr(%%) \t Eff_empT_pyr(%%) \t Aver.Tmod[C] \t',...
        'F1 \t F2 \t F3\t F4 \t F5 \t F6 \t nPoints_deleted \t',...
        'nMeas_points \n']);
    % Writing the data to the file 'filename'
    for i=1:12*31*fnYears
        fprintf(fid,['%d/%02d/%02d \t %8.2f \t %8.2f \t %6.2f ',...
            '\t %6.2f \t %6.2f \t %6.2f \t %6.2f \t %5.2f \t ',...
            '%5.2f \t %5.2f \t %5.2f \t %5.2f \t %5.2f \t %5.2f ',...
            '\t %d \t %d \t %d \t %d \t %d \t %d  \t %d  \t %d \n'],...
            date_arr(i,:),Array_day(i,:,n));
    end
    % Adding explanations to the text file
    fprintf(fid,...
        '\n%% G_esti = Daily total irradiation from ESTI sensor \n');
    fprintf(fid,...
        '%% G_pyr = Daily total irradiation from pyranometer \n');
    fprintf(fid,...
        '%% E = Daily total measured energy production \n');
    fprintf(fid,...
        ['%% E_emp_esti = Daily total predicted energy production, ',...
        'calculated using ESTI irradiation and measured ',...
        'module temperatures. \n']);
    fprintf(fid,...
        ['%% E_empT_esti = Daily total predicted energy production, ',...
        'calculated using ESTI irradiation and empirical ',...
        'module temperatures. \n']);
    fprintf(fid,...
        ['%% E_emp_pyr = Daily total predicted energy production, ',...
        'calculated using pyranometer irradiation and ',...
        'measured module temperatures. \n']);
    fprintf(fid,...
        ['%% E_empT_pyr = Daily total predicted energy production, ',...
        'calculated using pyranometer irradiation and ',...
        'empirical module temperatures. \n']);
    fprintf(fid,...
        ['%% Eff_esti = Daily efficiency, calculated using ',...
        'measured energy production and ESTI irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_emp_esti = Daily efficiency, calculated using ',...
        'predicted energy production (calculated with measured ',...
        'module temperatures) and ESTI irradiance. \n']);
    fprintf(fid,['%% Eff_empT_esti = Daily efficiency, calculated ',...
        'using predicted energy production (calculated with ',...
        'empirical module temperatures) and ESTI irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_pyr = Daily efficiency, calculated using ',...
        'measured energy production and pyranometer irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_emp_pyr = Daily efficiency, calculated using ',...
        'predicted energy production (calculated with measured ',...
        'module temperatures) and pyranometer irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_empT_pyr = Daily efficiency, calculated using '...
        'predicted energy production (calculated with empirical ',...
        'module temperatures) and pyranometer irradiance. \n']);
    fprintf(fid,...
        '%% Aver.Tmod = Daily average module temperature. \n');
    fprintf(fid,...
        '\n%% F1 = No of Pmax==0 -elements in the original data \n');
    fprintf(fid,...
        '%% F2 = No of Pmax<0 -elements in the original data \n');
    fprintf(fid,...
        ['%% F3 = No of Pmax_emp or Pmax_empT <0 or ',...
        'complex -elements in the original data \n']);
    fprintf(fid,['%% F4 = No of ',...
        'abs(Pmax-Pmax_emp)/max(Pmax,Pmax_emp)>0.5 -elements ',...
        'in the original data \n']);
    fprintf(fid,...
        ['%% F5 = No of irr_change>(given treshold value) ',...
        '-elements in the original data (ene analysis) / ',...
        'No of abs(G_esti-G_pyr)/max(G_esti,G_pyr)>0.3  ',...
        '-elements in the original data (enr analysis) \n']);
    fprintf(fid,...
        '%% F6 = No of Pmax>Pmax_STC -elements in the original data \n');
    fprintf(fid,...
        '%% nPoints_deleted = the no of points deleted in filtering \n');
    fprintf(fid,...
        '%% nMeas_points = the original number of measurement ',....
        'points in the unfiltered data \n');
    fclose(fid);
end

%% Writing the monthly values

G_es_Marray=zeros(12*fnYears,fN);
G_py_Marray=zeros(12*fnYears,fN);
E_Marray=zeros(12*fnYears,fN);
E_emp_es_Marray=zeros(12*fnYears,fN);
E_empT_es_Marray=zeros(12*fnYears,fN);
```

```
E_emp_py_Marray=zeros(12*fnYears,fN);
E_empT_py_Marray=zeros(12*fnYears,fN);
Eff_es_Marray=zeros(12*fnYears,fN);
Eff_py_Marray=zeros(12*fnYears,fN);
Eff_emp_es_Marray=zeros(12*fnYears,fN);
Eff_empT_es_Marray=zeros(12*fnYears,fN);
Eff_emp_py_Marray=zeros(12*fnYears,fN);
Eff_empT_py_Marray=zeros(12*fnYears,fN);
T_Marray=zeros(12*fnYears,fN);
flag1_Marray=zeros(12*fnYears,fN);
flag2_Marray=zeros(12*fnYears,fN);
flag3_Marray=zeros(12*fnYears,fN);
flag4_Marray=zeros(12*fnYears,fN);
flag5_Marray=zeros(12*fnYears,fN);
flag6_Marray=zeros(12*fnYears,fN);
points_del_Marray=zeros(12*fnYears,fN);
meas_points_Marray=zeros(12*fnYears,fN);

Array_month=zeros(fnYears*12,24);
year_array=zeros(12*fnYears,1);
month_array = repmat((1:12)',fnYears,1);


for n=1:fN
    for m=1:12*fnYears
        flag1_Marray(m,n) = sum(flag1_Darray((m-1)*31+1:m*31,n));
        flag2_Marray(m,n) = sum(flag2_Darray((m-1)*31+1:m*31,n));
        flag3_Marray(m,n) = sum(flag3_Darray((m-1)*31+1:m*31,n));
        flag4_Marray(m,n) = sum(flag4_Darray((m-1)*31+1:m*31,n));
        flag5_Marray(m,n) = sum(flag5_Darray((m-1)*31+1:m*31,n));
        flag6_Marray(m,n) = sum(flag6_Darray((m-1)*31+1:m*31,n));
        points_del_Marray(m,n) ...
            = sum(points_del_Darray((m-1)*31+1:m*31,n));
        meas_points_Marray(m,n) ...
            = sum(meas_points_Darray((m−1)*31+1:m*31,n));
    end
end

for y=1:fnYears
    year_array(1+(y−1)*12:y*12)=ones(12,1)*(date_arr(1,1)+y−1);
end

for n=1:fN
    G_es_Marray(:,n) = reshape(G_es_m(:,:,n),12*fnYears,1);
    G_py_Marray(:,n) = reshape(G_py_m(:,:,n),12*fnYears,1);
    E_Marray(:,n) = reshape(E_m(:,:,n),12*fnYears,1);
    E_emp_es_Marray(:,n) = reshape(E_emp_es_m(:,:,n),12*fnYears,1);
    E_empT_es_Marray(:,n) = reshape(E_empT_es_m(:,:,n),12*fnYears,1);
    E_emp_py_Marray(:,n) = reshape(E_emp_py_m(:,:,n),12*fnYears,1);
    E_empT_py_Marray(:,n) = reshape(E_empT_py_m(:,:,n),12*fnYears,1);
    Eff_es_Marray(:,n) = reshape(Eff_es_m(:,:,n),12*fnYears,1);
    Eff_emp_es_Marray(:,n) = ...
        reshape(Eff_emp_es_m(:,:,n),12*fnYears,1);
    Eff_empT_es_Marray(:,n) = ...
        reshape(Eff_empT_es_m(:,:,n),12*fnYears,1);
    Eff_py_Marray(:,n) = reshape(Eff_py_m(:,:,n),12*fnYears,1);
    Eff_emp_py_Marray(:,n) = ...
        reshape(Eff_emp_py_m(:,:,n),12*fnYears,1);
    Eff_empT_py_Marray(:,n) = ...
        reshape(Eff_empT_py_m(:,:,n),12*fnYears,1);
    T_Marray(:,n) = reshape(Tmod_m(:,:,n),12*fnYears,1);

    Array_month(:,:,n) = [month_array year_array G_es_Marray(:,n) ...
        G_py_Marray(:,n) E_Marray(:,n) E_emp_es_Marray(:,n)  ...
        E_empT_es_Marray(:,n) E_emp_py_Marray(:,n)...
        E_empT_py_Marray(:,n) Eff_es_Marray(:,n) ...
        Eff_emp_es_Marray(:,n) Eff_empT_es_Marray(:,n) ...
        Eff_py_Marray(:,n) Eff_emp_py_Marray(:,n) ...
        Eff_empT_py_Marray(:,n) T_Marray(:,n) ...
        flag1_Marray(:,n) flag2_Marray(:,n) flag3_Marray(:,n) ...
        flag4_Marray(:,n) flag5_Marray(:,n) flag6_Marray(:,n)...
        points_del_Marray(:,n) meas_points_Marray(:,n)];
end

fn = input(['\n Filename in which you want to store the monthly ',...
    'results \n (module name will be added as a postfix)? '],'s');
for n=1:fN
    filename = sprintf('results\\%s_%s.txt',fn,fmod{n});
    fid = fopen(filename,'wt');
    % Writing the header to the file 'filename'
    fprintf(fid,['%% Month \t G_esti[Wh/m2] \t G_pyr[Wh/m2] \t',...
        'E[Wh] \t E_emp_esti[Wh] \t E_empT_esti[Wh] \t',...
        'E_emp_pyr[Wh] \t E_empT_pyr[Wh] \t Eff_esti(%%) \t',...
        'Eff_emp_esti(%%) \t Eff_empT_esti(%%) \t Eff_pyr(%%)',...
        '\t Eff_emp_pyr(%%) \t Eff_empT_pyr(%%) \t Aver.Tmod[C]',...
        '\t F1 \t F2 \t F3 \t F4 \t F5  \t F6 \t ',...
        'nPoints_deleted \t nMeas_points \n']);
    % Writing the data to the file 'filename'
    for i=1:12*fnYears
        fprintf(fid,['%02d/%d \t %8.2f \t %8.2f \t %7.2f \t',...
            '%7.2f \t %7.2f \t %7.2f \t %7.2f \t %5.2f \t',...
            '%5.2f \t %5.2f \t %5.2f \t %5.2f \t %5.2f \t',...
            '%5.2f \t %d \t %d \t %d \t %d \t %d \t %d  \t',...
            '%d \t %d\n'],Array_month(i,:,n));
    end
    % Adding explanations to the text file
    fprintf(fid,...
        '\n%% G_esti = Monthly total irradiation from ESTI sensor \n');
    fprintf(fid,...
        '%% G_pyr = Monthly total irradiation from pyranometer \n');
    fprintf(fid,...
        '%% E = Monthly total measured energy production \n');
    fprintf(fid,...
        ['%% E_emp_esti = Monthly total predicted energy ',...
        'production, calculated using ESTI irradiation and ',...
        'measured module temperatures. \n']);
    fprintf(fid,...
        ['%% E_empT_esti = Monthly total predicted energy ',...
        'production, calculated using ESTI irradiation and ',...
        'empirical module temperatures. \n']);
    fprintf(fid,...
        ['%% E_emp_pyr = Monthly total predicted energy ',...
        'production, calculated using pyranometer irradiation ',...
        'and measured module temperatures. \n']);
    fprintf(fid,...
        ['%% E_empT_pyr = Monthly total predicted energy ',...
        'production, calculated using pyranometer irradiation ',...
        'and empirical module temperatures. \n']);
    fprintf(fid,...
        ['%% Eff_esti = Monthly efficiency, calculated using ',...
        'measured energy production and ESTI irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_emp_esti = Monthly efficiency, calculated using ',...
        'predicted energy production (calculated with measured ',...
        'module temperatures) and ESTI irradiance. \n']);
    fprintf(fid,['%% Eff_empT_esti = Monthly efficiency, ',...
        'calculated using predicted energy production ',...
        '(calculated with empirical module temperatures) ',...
        'and ESTI irradiance. \n']);
    fprintf(fid,...
        ['%% Eff_pyr = Monthly efficiency, calculated using ',...
        'measured energy production and pyranometer irradiance. \n']);
```

```
fprintf(fid,...
    ['%% Eff_emp_pyr = Monthly efficiency, calculated using ',...
    'predicted energy production (calculated with measured ',...
    'module temperatures) and pyranometer irradiance. \n']);
fprintf(fid,...
    ['%% Eff_empT_pyr = Monthly efficiency, calculated using '...
    'predicted energy production (calculated with empirical ',...
    'module temperatures) and pyranometer irradiance. \n']);
fprintf(fid,...
    '%% Aver.Tmod = Monthly average module temperature. \n');
fprintf(fid,...
    '\n%% F1 = No of Pmax==0 -elements in the original data \n');
fprintf(fid,...
    '%% F2 = No of Pmax<0 -elements in the original data \n');
fprintf(fid,...
    ['%% F3 = No of Pmax_emp or Pmax_empT <0 or ',...
    'complex -elements in the original data \n']);
fprintf(fid,['%% F4 = No of ',...
    'abs(Pmax-Pmax_emp)/max(Pmax,Pmax_emp)>0.5 -elements ',...
    'in the original data \n']);
fprintf(fid,...
    ['%% F5 = No of irr_change>(given treshold value) ',...
    '-elements in the original data (ene analysis) / ',...
    'No of abs(G_esti-G_pyr)/max(G_esti,G_pyr)>0.3  ',...
    '-elements in the original data (enr analysis) \n']);
fprintf(fid,...
    '%% F6 = No of Pmax>Pmax_STC -elements in the original data \n');
fprintf(fid,...
    '%% nPoints_deleted = the no of points deleted in filtering \n');
fprintf(fid,...
    '%% nMeas_points = the original number of measurement ',....
    'points in the unfiltered data \n');
 fclose(fid);
```

```
end
```

D.8 The function month_teller

```
function [fcurrentmonth] = month_teller(jj)

%----------------------------------------------------------%
%This function gives the name of the current month
%----------------------------------------------------------%
fcurrentmonth='0';
switch jj
case '1',
   fcurrentmonth='JANUARY';
case '2',
   fcurrentmonth='FEBRUARY';
case '3',
   fcurrentmonth='MARCH';
case '4',
   fcurrentmonth='APRIL';
case '5',
   fcurrentmonth='MAY';
case '6',
   fcurrentmonth='JUNE';
case '7',
   fcurrentmonth='JULY';
case '8',
   fcurrentmonth='AUGUST';
case '9',
   fcurrentmonth='SEPTEMBER';
case '10',
   fcurrentmonth='OCTOBER';
case '11',
   fcurrentmonth='NOVEMBER';
case '12',
   fcurrentmonth='DECEMBER';
otherwise,
   disp('THE MONTH NUMBER IS NOT VALID,IT SHOULD BE BETWEEN 1 AND 12');
return
end
```

E Configuration file for empirical data

Below is the text file from which the program retrieves the empirical equation for maximum power as a function of irradiance and module temperature ($P_{max}(G, T_{mod})$) and its coefficients (a-f). In addition different STC P_{max} values used in module comparison and the module surface area used in calcuating efficiencies are retrieved from this file. Here the columns have been rearranged in order to fit the data to the page; in the original file, the columns of each row are on the same line.

```
% This file contains the indoor performance surface equation
% and parameters for calculating the empirical Pmax,
% as well as the module area, and LAPSS and outdoor STC powers
% for calculating relative outputs to perform module comparison.
% !NOTE!: In the empirical equation, all the parameters
% (a,b,c,d,e,f) must excist, allthough part of them were zeros.
% The different STC powers (outdoor, LAPSS, label) are used to
% perform module comparison in terms of relative (Wh/Wp) outputs.
% If some of the values are not known, assign them to unity.

%       Empirical equation
by71    a+b*Irr+c*log(T)+d+e+f
by72    a+b*Irr+c*T.*log(T)+d+e+f
hr706   a+b*Irr+c*log(T)+d+e+f
ju711   a+b*Irr+c*T+d+e+f
ai01    a+b*Irr+c*log(T)+d*Irr.^2+e*(log(T)).^2+f*Irr.*log(T)
dn09    a+b*Irr+c*log(T)
```

```
a                b                c

20.48933         0.069211135      -6.1469094

2.316871         0.0654907        -0.021921053

1.7579569        0.0075692241     -0.5038663

5.3191675        0.075535083      -0.17738281

-21.393925       0.09083602       11.132581

18.737608        0.062444719      -5.4242185


d                e                f                Module area[m2]

0                0                0                0.7308

0                0                0                0.7308

0                0                0                0.09

0                0                0                0.70625

-9.7387338E-7    -1.5077081       -0.01048595      0.491056

0                0                0                0.717002


Pmax_STC_LAPSS[W]        Pmax_STC_out[W] Pmax_STC_label[W]

79.1                     83.314          75

76.1                     83.199          75

8.9                      9.1             1

82.8                     81.80           80

1                        1               1

1                        1               1
```