UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

NUMERICAL SIMULATION OF RADIATION AND CONDUCTION HEAT TRANSFER IN A PARTICLE LADEN AREA

Autor:

Muñoz Mínguez, Marta

Responsable de Intercambio en la Uva:

Parra Santos, Mº Teresa

Universidad de destino:

San Diego State University

Valladolid, enero de 2018.

TFM REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO:	Numerical Simulation of Radiation and Conduction Heat Transfer in a Particle Laden Area
ALUMNO:	Marta Muñoz Mínguez
FECHA:	5 de diciembre de 2017
CENTRO:	Universidad Estatal de San Diego, Departamento de Ingeniería Aeroespacial
TUTOR:	Gustaaf B. Jacobs

RESUMEN

Este TFM estudia el proceso transitorio del calentamiento de partículas en el interior de un Receptor de Partículas en Caída, una tecnología empleada para transformar la energía solar en electricidad.

La razón de ser de este TFM es el tamaño significativamente superior de estas partículas en comparación con las utilizadas en otros tipos de receptores solares, ya que puede afectar fuertemente al tiempo necesario para que las partículas alcancen las altas temperaturas deseadas.

El Método de los Elementos Finitos y el Método de Monte Carlo para el trazado de rayos se han utilizado para simular este proceso en un modelo bidimensional escrito en el programa MATLAB, donde la transferencia de calor mediante radiación y conducción se ha modelado minuciosamente.

Los resultados derivados de este estudio demuestran que, efectivamente, el tiempo necesario para que las partículas alcancen una alta temperatura no es trivial y merece una atención especial.

PALABRAS CLAVE

Receptor de Partículas en Caída, Radiación, Conducción, Método de Monte Carlo, Método de los Elementos Finitos

ORIGINAL ABSTRACT

This Thesis is a study of the transient heating process of the particles inside a Falling Particle Receiver, a technology used to transform the solar energy in electricity. The particles used in the Falling Particle Receivers are significantly bigger than the ones used in other solar receivers more common and studied in more depth, like the Small Particle Receivers. This important difference in particle size is the raison d'être of this Thesis, because it might strongly affect the time needed for the particles to reach the high temperatures desired.

The Finite Element Method and the Monte Carlo Ray Tracing Method have been used to simulate this process in a two-dimensional model written in the program MATLAB, where radiation and conduction heat transfer have been minutely modeled.

Throughout this Thesis, the governing equations of the process are explained, the model is firsly tested in simple problems with single particles, and secondly run in a set of particles located inside of the receiver.

The obtained results prove how, effectively, the time needed for the particles to reach high temperatures is not trivial, and they also show that it is very likely that recirculation of the particles is necessary in order to reach the desired temperatures.

KEY WORDS

Falling Particle Receiver, Radiation, Conduction, Monte Carlo Ray Tracing Method,

Finite Element Method

NUMERICAL SIMULATION OF RADIATION AND CONDUCTION HEAT TRANSFER IN A PARTICLE LADEN AREA

Author: Marta Muñoz Mínguez

Advisor: Dr. Gustaaf B. Jacobs

Máster en Ingeniería Industrial

Universidad de Valladolid

San Diego, December of 2017

ABSTRACT

NUMERICAL SIMULATION OF
RADIATION AND CONDUCTION HEAT TRANSFER
IN A PARTICLE LADEN AREA

Marta Muñoz Mínguez
Universidad de Valladolid
December of 2017

This Thesis is a study of the transient heating process of the particles inside a Falling Particle Receiver, a technology used to transform the solar energy in electricity. The particles used in the Falling Particle Receivers are significantly bigger than the ones used in other solar receivers more common and studied in more depth, like the Small Particle Receivers. This important difference in particle size is the raison d'ˆetre of this Thesis, because it might strongly affect the time needed for the particles to reach the high temperatures desired.

The Finite Element Method and the Monte Carlo Ray Tracing Method have been used to simulate this process in a two dimensional model written in the program MATLAB, where radiation and conduction heat transfer have been minutely modeled.

Throughout this Thesis, the governing equations of the process are explained, the model is firsly tested in simple problems with single particles, and secondly run in a set of particles located inside of the receiver.

The obtained results prove how, effectively, the time needed for the particles to reach high temperatures is not trivial. They also show that it is very likely that recirculation of the particles is necessary in order to reach the desired temperatures.

ACKNOWLEDGEMENTS

To my parents, for all their help, support and effort, that has allowed me to live a very unique and espcial experience in the city of San Diego.

I would also like to express my gratitude towards my advisor in San Diego State University Dr. Gustaaf B. Jacobs, without whose constant help and advice I would not have been able to complete this work, and towards Dr. Fletcher Miller for his help and input regarding the radiation phenomena.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

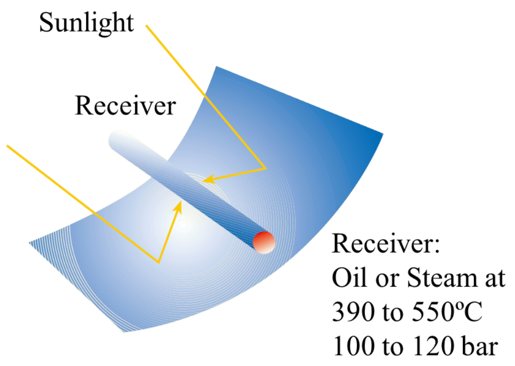
CHAPTER 1

Introduction

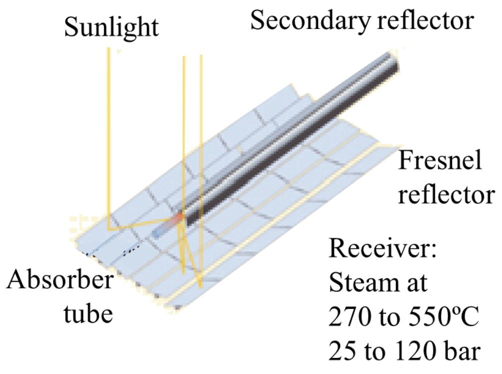
In the last decades, the concern of the society for the planet's sustainability and energy resources has increased noticeably. The environmental considerations together with the growing dependency on energy have led to an urgent global need for clean and renewable energy sources. The main challenge in the development of the renewable energy technologies is its cost, which currently is not competitive with the cost of fossil fuels. Cooperation between political, economic and scientific institutions is fundamental to guarantee the healthy sustainability of mankind.

Sunlight is a major source of energy and it holds the greatest potential to meet the energy demand of the world. The total annual solar energy input into the Earth is more than 15000 times greater than the current yearly use of fossil and nuclear fuels [5]. If only a 0.1% of this solar energy was converted at an efficiency of a 10%, the electricity obtained would be four times the world's electricity generating capacity in 2013 (about 5000GW) [6].

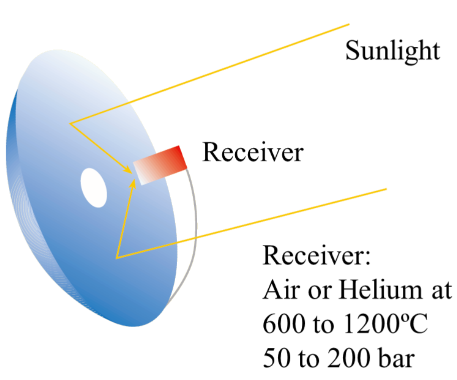
Solar energy can currently be harnessed with two different technologies: *solar photovoltaic* and *solar thermal* technologies. *Solar photovoltaic* technologies directly transform sunlight into electricity in photovoltaic cells and are only effective during daytime, while *solar thermal* technologies concentrate sunlight to obtain heat. If said heat is directly used, the technologies are known as *active* or *passive* solar heating, and if it is indirectly used to generate electric power, the technology is known as *concentrating solar power (CSP)*. This last technology uses solar energy collectors to reach temperatures high enough to drive steam turbines that generate the mechanical work necessary to run a generator and produce electricity [5]. There are three different CSP technologies, primarily separated by their scale and operating temperatures, including *line-concentrating systems* (parabolic troughs and linear Fresnel systems), *dish concentrating systems* and *central tower systems*, also known as *solar power towers (SPT)* [7,8]. A scheme of these different technologies is shown in Figure 1.1.



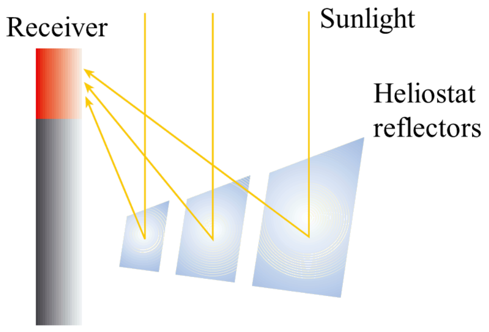
(a) Parabolic Troughs squeme



(b) Linear Fresnel scheme



(c) Dish Concentrating system scheme



(d) Power Tower scheme

Figure 1.1: Squemes of different CSP technologies [1]

SPT systems use a field of distributed heliostats that individually track the sun and focus the sunlight onto a central receiver, located in the centre of the heliostat field at the top of a large solar tower. The concentrated solar energy is absorbed in the central receiver by a working medium, that then is sent to storage or to a powercycle [9, 10]. The working medium needed to transport the stored energy can be a gas, a liquid, or be made up of solid particles [11]. There is a great variety of receiver architectures, that can be divided in three main groups based on how the radiation is absorbed and transferred to the working medium [8]: *tube absorption receivers*, *volumetric receivers* and *particle receivers*. Tube absorption receivers use dark tubes to absorb solar energy and transfer it via convection to a working fluid inside, usually working with liquids such as molten salts; volumetric receivers have porous materials into which radiation penetrates and is absorbed, being air its most common working fluid; and particle receivers use dark solid particles to directly absorb sunlight.

There exist different types of particle receivers, such as Centrifugal Receivers or Falling Particle Receivers (FPR), and there are several different configurations for a FPR, being one of them the Free-Falling Particle Receiver. A simplified scheme of this configuration is shown in Figure 1.2.

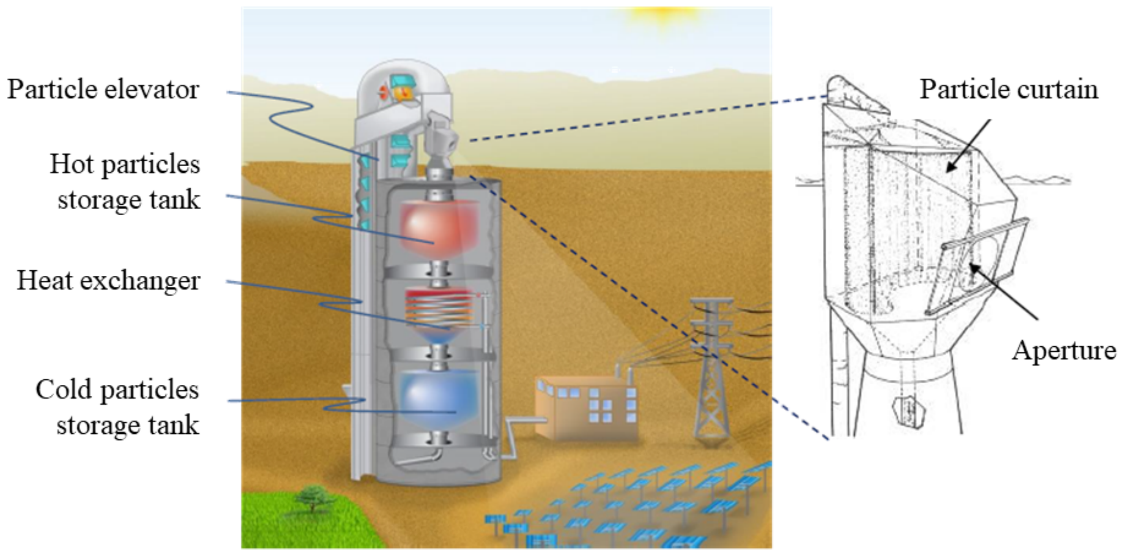


Figure 1.2: Squeme of a Free-Falling Particle Receiver [2]

The process works as follows:
(1) Cold sand-size refractory particles fall down freely, forming a curtain after being released through a slot at the top of the receiver. There, the concentrated sunlight that comes in through a window directly irradiates them. The particles absorb part of this radiation, having a much a higher temperature by the time they exit the receiver.
(2) These heated particles are stored in an insulated storage tank, called the “hot storage tank”.
(3) Then they run through a heat exchanger to heat the working fluid for the power cycle [11, 12], serving as both the heat transfer fluid and thermal storage media.
(4) The cooled particles fall into the “cold storage tank”.
(5) Finally, the particles are brought back up by an elevator to the top receiver, and the process repeats.
The main advantages of the FPR compared to other CSP technologies are the reach of operational temperatures over of $1000°C$, a higher receiver efficiency, a storage media that is chemically benign and low-cost and a reduction on the thermal stress on the plant components. However, the overall efficiency is reduced by convection heat transfer through the open aperture and by heat loss from the particles to the air [13].

Modeling CSP receivers is not an easy task. The developed models are usually specific to a concrete architecture and application, since the radiation fluxes an related heat transfer are different in every case. As the geometry of the receiver becomes more complicated, modeling the radiation between surfaces gets more complicated as well.

Simulations and experiments regarding the FPR were already done in the 1980s, mostly by Sandia National Laboratories. One of the reasons why the free-falling PR was proposed as an alternative to harness solar energy was the study made by Falcone et al. in 1985 [14], where he studied the costs of the Solid Particle Receiver (SPR) in comparison with other air receiver concepts for high-temperature applications. In 1986, Stahl and Griffin focused on identifying appropriate particle materials (Stahl and Griffin, 1986 [15]). Investigations carried out by Hruby [16] were focused on the receiver design and the particle materials, while computational models where created to simulate the receiver operation and the temperature of the particles [17, 18]. In 1988, Hruby et al. [19] developed further work in the study of flow characteristics and convective heat transfer in a falling particle curtain. In 1999, Meier [13] performed 2D CFD simulations of a chemical FPR, to study the performance of a FPR in a pilot plant, using the Monte Carlo method to model the heating of the particles by radiation. His model contemplated radiation from the gas to and from the particles but did not consider the change in the radiation field as a response to the absorption by or emission from the particles. Meier's results were extended to 3D by Chen et al. in 2007 [20], to help in the design of a prototype receiver by Sandia National Laboratories. Also in 2007, Chen et al. [21] developed a CFD model in FLUENT where a solar ray-tracing algorithm was used to predict the solar illumination energy on the walls of different receiver designs. The heating of the particles considered re-radiation from the walls, but not the direct irradiation from the heliostats.

In the following years, several FPR prototypes were built in order to gather information about distribution of particles velocity, curtain thickness, curtain opacity... and other properties, and to experimentally validate new simulation codes. In 2008, a prototype SPR was designed and tested on-sun at Sandia Nationa Laboratories [22], and Ho et al. [23] developed a CFD model to try on it. They simulated the solar irradiation and heat transfer with a discrete-ordinates radiation model in FLUENT, without performing any ray tracing. Their model solved the radiative transfer equation (RTE) over a domain of discrete solid angles, transforming it into as many transport equations as there are solid angles. In 2009, Kim et al. [24] validated the experimental results from a prototype with the computational results of a 2D MFIX simulation model. Their work was mainly focused on particle velocity, curtain thickness and transparency, but neglected the heat transfer to the particles. In 2010, Khalsa [25] developed a program in Microsoft Excel VBA that also used the discrete-ordinates method for the radiation model. His program characterized multiple beams emanating from a surface called the solar patch, defined at the center of the receiver aperture and divided into subpatches, each of which represented irradiation from a unique section of the heliostat field. In 2011, Khalsa and Ho [26] developed a model that improved the treatment of the radiation given in [27] and [25], since it considered both the directional and spatial variability of the incoming radiance distribution. This method allowed to transform the incoming radiation from the heliostat field into a radiance boundary condition on the solid particle receiver aperture. In 2015, Wu et al. [28] developed a three-dimensional, steady-state numerical model of a centrifugal particle receiver. The cavity body was discretized with the Finite Element Method (FEM), and the discrete particles where not modeled. Instead,they considered that the receiver wall was fully covered by a single-layered, optically dense, and homogeneously moving particle film that was represented by a 1D fluid flow element.

These models focus on different aspects of a FPR: particle velocity, curtain opacity, solar irradiation... but they often do not simulate the particles themselves, but consider that the rays bounce randomly in the particle-laden area. Many models neglect the time it takes for heat transfer to happen inside the particles, considering that because of their small size, the heating process can be assumed to be instantaneous. That is only a realistic consideration when the size of the particles is similar or smaller than the wavelength of sunlight - from 0.5 μm to 2.5 μm approximately - which occurs, for example, in the Small Particle Receivers (SPR) that work with nanoparticles. Since the average diameter of the particles in the FPR is close to 500 μm, the time it takes for a particle to reach the desired temperatures can be an important factor and should not be disregarded. Figure 1.3 shows a schematic comparison of the sizes of a SPR and a FPR.

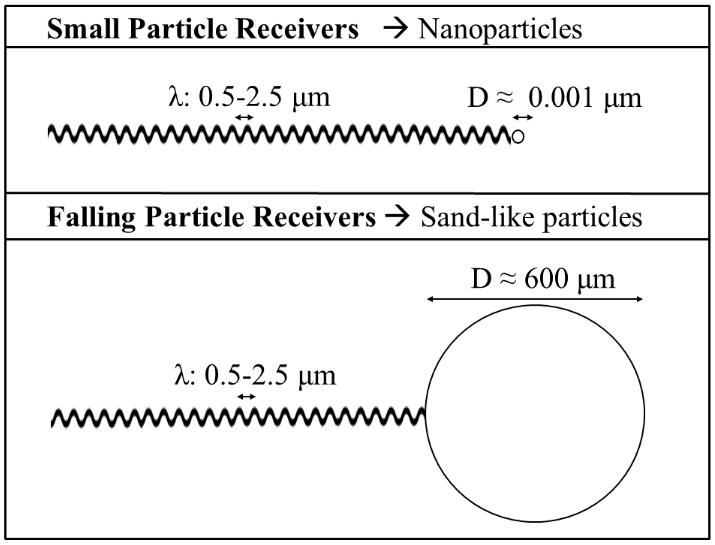

Small Particle Receivers → Nanoparticles
λ: 0.5-2.5 μm
D ≈ 0.001 μm
Falling Particle Receivers → Sand-like particles
D ≈ 600 μm
λ: 0.5-2.5 μm

Figure 1.3: Comparison of particle sizes in SPR and FPR

In this Thesis, we simulate the interior of a FPR, using the Monte Carlo Ray Tracing Method to simulate the radiative energy and the Finite Element Method (FEM) to model the transient heat transfer in two-dimensional circular particles. Solar irradiation is modeled by a set of rays that come through the receiver window in random directions and circular two-dimensional bodies of different sizes are generated in random positions inside the receiver, in order to simulate a small sample of falling particles. The rays are partially absorbed and reflected by the particles located in their path or by the receiver walls, bouncing in random directions until their energy is neglictible or until they go out through the window. Absorption, reflection and emission from the particles is minutely simulated. Every time a ray reaches a particle a Neumann boundary condition is implemented in it, that depends on the energy of the ray in that instant and the properties of the particle material. The radiative energy emitted by the particles is also simulated by a set of rays, that act in the same way as the solar rays. However, the gas-particle interaction is neglected and the gas and particle flows are not considered. The program has been implemented in MATLAB and tested in a single particle initially, and run in multiple particles secondly to imitate the functioning of a set of particles.

Chapter 2 of this Thesis introduces the governing equations of radiation and conduction of the process. Chapter 3 presents the numerical methods used, which are a Monte Carlo ray tracing method for the solar irradiation and emission from the particles, and the Finite Element Method (FEM) for the heat transfer through the particles. Chapter 4 introduces the numerical model developed, applied to simple cases of study and to the complete model. Chapter 5 presents the most important results and the corresponding conclusions. Finally in Chapter 6, propositions of future work in this model are presented. A detailed explanation of the computational model can be found on Appendix A, and the developed MATLAB code has been attached in Appendix B.

CHAPTER 2

Governing Equations

2.1 Radiation Heat Transfer

Radiation Heat Transfer consists on the transfer of thermal energy by an electromagnetic wave. The electromagnetic radiation spectrum ranges from cosmic rays with wavelengths smaller than $10^{-8}\mu m$ to radio waves with wavelengths up to $10^{10}\mu m$, where thermal radiation occupies the portion between $10^{-1}\mu m$ and $10^{3}\mu m$ that is detected as heat or light. Radiation Heat Transfer does not require a physical medium to take place, which is why sunlight can travel all the way from the Sun to the Earth. However, the Earth's atmosphere acts as a filter and only a fraction of the spectrum reaches the surface. As stated in Chapter 1, the part of the thermal radiation that does has a wavelength approximately between 0.5 and $2.5\mu m$, as can be seen in Figure 2.1.

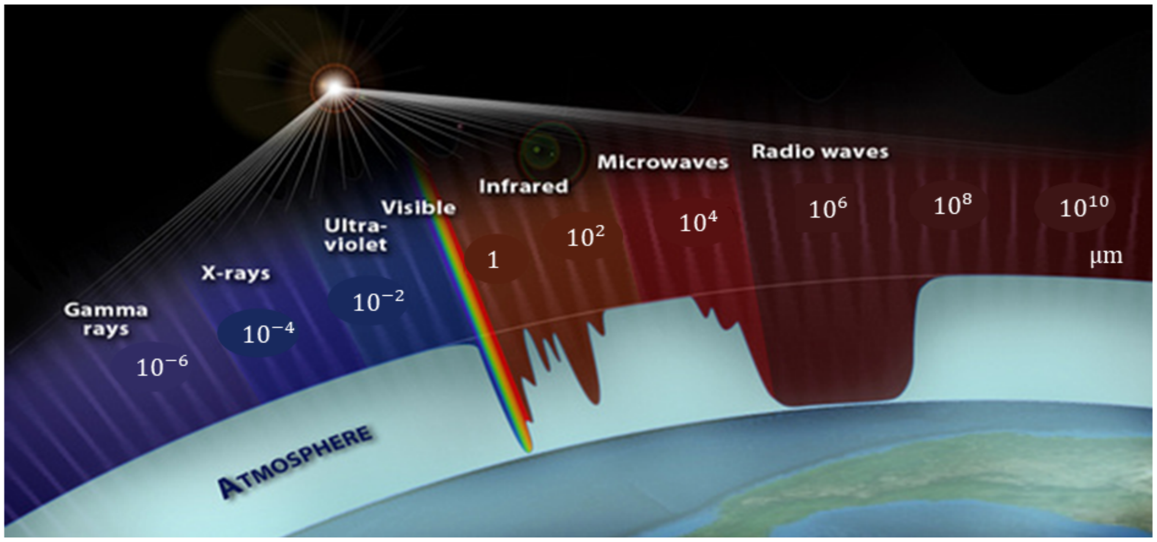


Figure 2.1: Part of the electromagnetic spectrum that reaches the surface of the Earth [3]

When radiation incides on the surface of a body, different phenomena can occur: *reflection*, if all or part of the energy is reflected by the surface, *absorption*, if all or part of the energy penetrates the body and is absorbed by it, or *transmission*, if all or part of the energy penetrates the body, is transmitted an emerges from it. If the medium is *opaque* (as are the particles in a FPR), all the radiation that penetrates into the medium is absorbed, and therefore transmission does not occur. The fraction of the incident energy that is reflected, absorbed or transmitted by a surface is defined by its reflectivity ρ , absorptivity α and transmissivity τ, which are properties of the material and have a value between 0 and 1.

The reflectivity of a surface depends on the direction of the incident rays and on the direction of the reflected rays. In real surfaces, the reflected rays usually form an irregular shape, but modeling reflection this way is very complicated. Therefore, different assumptions are often made for simplification purposes. Surfaces can be classified as smooth or rough depending their smoothness, which is defined relatively to the wavelength of the incident radiation. If the height of the surface roughness is bigger than said wavelength, the surface is rough, and otherwise it is smooth. In rough surfaces reflection is assumed to be perfectly *diffuse*, being the radiation equally reflected in all directions, and in smooth surfaces it is considered to be *specular*, where the angles of incidence and reflection are equal. A schematic representation of these different types of reflection is shown in Figure 2.2.

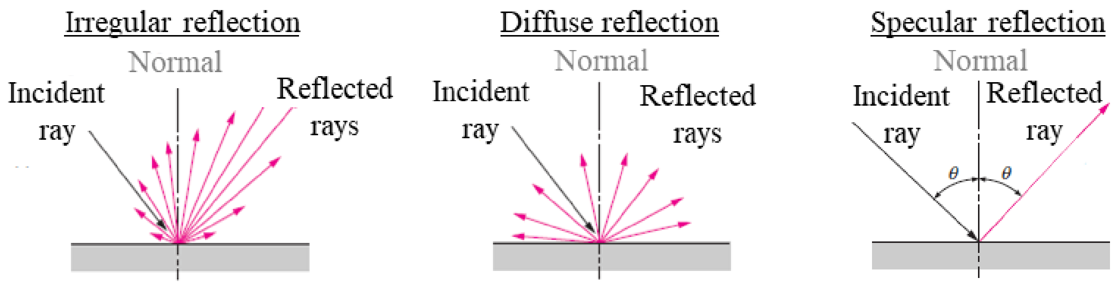


Figure 2.2: Different types of reflection [4]

Besides the three phenomena explained before, any body with a temperature higher than the absolute zero emits radiation, so the phenomenon of *emission* must also be taken into account when studying Radiative Heat Transfer. The wavelength of the emitted radiation is in the range of infrared or visible light depending on the body's absolute temperature.

In order to properly understand the study of Radiation Heat Transfer it is necessary to introduce the concept of a *black body*, which is a theoretical physical body that absorbs all the energy that it receives (perfect absorber), in every wavelength and from every direction, and emitts more energy than any other body. The *Stefan-Bolztmann law* expresses the *total emissive power* (e_b) of a *black body* as

$$e_b = \int_0^\infty e_{b\lambda} d\lambda = \sigma T^4 \;, \tag{2.1}$$

where $e_{b\lambda}$ is the emissive power of the black body at a specific wavelength λ in$\frac{W}{m^2}$, α is Stefann-Bolztmann constant ($\sigma = 5.6704 \cdot 10^8 \frac{W}{m^2K^4}$) and T is the absolute temperature of the body in K. The emissive power of a real surface is expressed as a fraction of the blackbody emissive power as

$$e_\lambda = \epsilon_\lambda e_b \;, \tag{2.2}$$

being ϵ_λ the monochromatic hemispherical emissivity of the surface. The emissivity of a body expresses how well it emits compared to a black body, and it depends on the radiation wavelength, ans the surface material, temperature and roughness.

For simplifications purposes, in most engineering applications it is assumed that emissivities of real surfaces are independent of the radiation wavelength. Bodies with this property are called *gray bodies*, and their emissive power is expressed as

$$e = \epsilon(T)e_b = \epsilon(T)\sigma T^4 \, . \tag{2.3}$$

If their emissivity is independent of temperature, then

$$e = \epsilon e_b = \epsilon \sigma T^4 \, . \tag{2.4}$$

Real bodies can absorb a fraction of the energy that they receive, determined by their *absorptivity* (α). According to *Kirchoffs law*, for a specific wavelength the monochromatic *absorptivity* (α) and *emissivity* (ϵ) of a surface at a given temperature are equal

$$\alpha_\lambda(T) = \epsilon_\lambda(T) \, . \tag{2.5}$$

For a *gray body* with properties independent of temperature, it is expressed as

$$\alpha = \epsilon \, . \tag{2.6}$$

The *reflectivity* (ρ) of a surface is the portion of the incident energy that is reflected back. Considering that for an opaque medium the total energy is either absorbed or reflected, the *reflectivity* (ρ) of a surface with properties independent of temperature can be expressed as

$$\rho = 1 - \alpha \, , \tag{2.7}$$

or, from Equation (2.6)

$$\rho = 1 - \epsilon \, . \tag{2.8}$$

An intermediate solution between the consideration of real or gray bodies is the use of a *multiband model*, where the spectrum is discretized into bands of finite width and it is assumed that radiation quantities are uniform in them. These bands are defined by radiation properties such as wavelength, frequency…

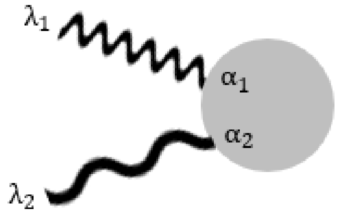


Figure 2.3: Schematic representation of a two band model, with bands defined by their wavelength

2.2 Conduction Heat Transfer

Fourier's law describes the conduction of thermal energy through a solid due to a temperature gradient. For an isotropic medium it takes the form

$$q = -k\frac{\partial T}{\partial n} \, , \tag{2.9}$$

where q is the rate of heat flow per unit area in the n direction, k is thermal conductivity and n is the normal direction, being the temperature gradient negative in the direction of positive heat flow. The law of conservation of energy for an isotropic 2D solid with temperature dependent on the thermal conductivity is

$$-\left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}\right) + Q = \rho C_p \frac{\partial T}{\partial t} \, , \tag{2.10}$$

where Q, ρ, C_p and t are the internal heat generation rate per unit volume, the density, the specific heat and the time, respectively.

The substitution of Equation (2.9) in Equation (2.10) leads to the heat conduction equation, which for an isotropic material with constant thermal properties, is expressed as

$$k\frac{\partial^2 T}{\partial x^2} + k\frac{\partial^2 T}{\partial y^2} + Q = \rho C_p \frac{\partial T}{\partial t} \, . \tag{2.11}$$

Dividing by k and regrouping ρ, C_p and k in this manner $\alpha = \dfrac{k}{\rho \, c_p}$, the thermal diffusivity α of the solid is obtained and, if the thermal properties are constant and there is no internal heat generation, the heat conduction equation is reduced to the diffusion equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \, . \tag{2.12}$$

2.2.1 Initial and Boundary Conditions

In order to solve Equation (2.12), an initial condition and several boundary conditions must be specified. The initial condition establishes the temperature of the solid at an initial time. In a two dimensional solid, it is expressed as

$$T = T_0(x, y) \; . \tag{2.13}$$

There are different kinds of boundary conditions, but the only kind of interest for the problem of study is the Neumann boundary condition, that specifies a heat flow across a boundary. It takes the form

$$-k\frac{\partial T}{\partial n} = q_n \; , \tag{2.14}$$

where q_n is the rate of the surface heat flow per area.

CHAPTER 3

Numerical Methods

3.1 The Monte Carlo Ray Tracing Method

A Monte Carlo Ray Tracing Method (MCRTM) has been developed to simulate the rays inside the receiver of the FPR. Instead of directly solving the Radiative Transfer Equation (RTE), the MCRT methods use a statistical approach to model the phenomena involved in the radiation heat transfer. The MCRT method used in this Thesis traces a statistically significant number of rays that simulate the behaviour of sunlight along their path inside the receiver: from their beginning in the window until their remaining energy is negligible or they are reflected back outside the window, considering their interaction with the particles (absorption and reflection) and the back wall (reflection). Following the path of a single ray does not give any significant information; it is the information obtained after the simulation of a large number of rays what gives a real idea of the way the Radiation Heat Transfer works. Thereby, after a great number of simulations, reliable estimations of the parameters involved inthe process are obtained. One of the downsides of this method is that, due to its statistical nature, in order to obtain accurate results a large number of rays must be simulated, which usually results in large computation time [29].

3.1.1 Modeling of the solar rays

The total energy transmitted to the interior of the receiver can be estimated as the heat flux q that reaches the window times the length l of the window section studied. In two dimensions:

$$Q_{total}(W) = q\left(\frac{W}{m}\right) \cdot l(m)\ . \tag{3.1}$$

This energy is then divided by the number of simulated rays, being the results more realistic when more rays are modeled.

$$Q_{ray}(W) = \frac{Q_{total}(W)}{n_{rays}} \tag{3.2}$$

Figure 3.1 squematizes this process.

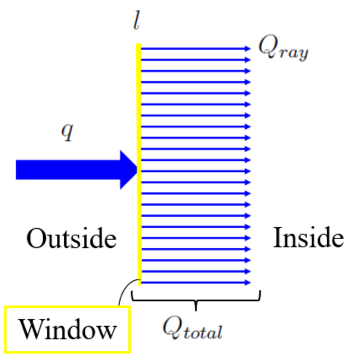


Figure 3.1: Heat flux in the simulated solar rays

The path of the rays is considered to start in the window, from where they follow a random direction (generated by a Random Number Generator (RNG)) inside the receiver, being reflected on the particles in random directions if they are located in their stablished path, or in the back wall otherwise. When a particle is hit by a ray, it absorbs a percentage of its energy, defined by the particle's material absorptivity α

$$Q_{absorbed} = Q_{incident} \cdot \alpha, \tag{3.3}$$

and the remaining energy $Q_{reflected} = Q_{incident} \cdot (1-\alpha)$ is redirected in a random direction and might hit other particles on its way. If the ray hits the back wall of the receiver, it is reflected in a random direction and it is considered that none of its energy is absorbed. The ray will continue being reflected, and its energy being absorbed by the particles until its path leads it to leave the receiver through the window. Along the path of a ray, a counter n stores the number of times that it has been reflected on any particle, as can be seen in Figure 3.2

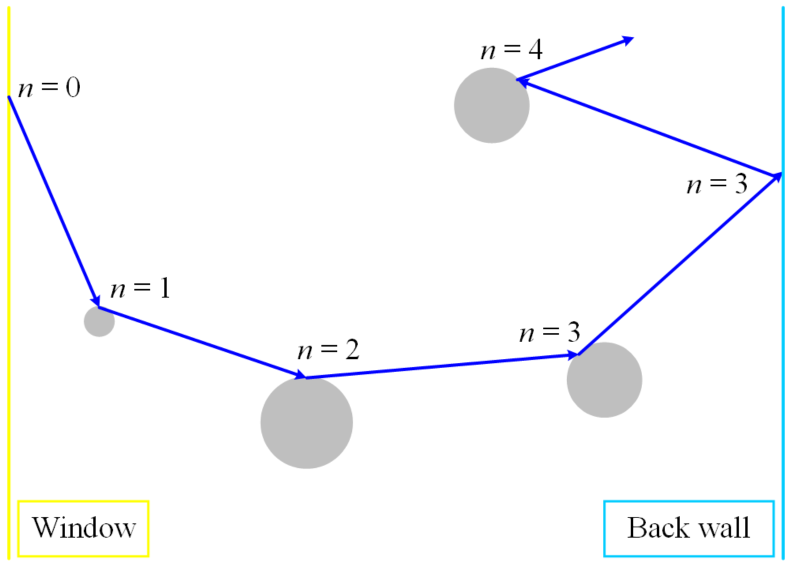


Figure 3.2: Scheme of the reflections of a ray along its path

This value of n allows to know the ray's energy at any moment. It depends on its initial energy, on n, and on the absorptivity of the particles α, and can be calculated as

$$Q_{ray} = Q_{initial} \cdot (1-\alpha)^n \, . \tag{3.4}$$

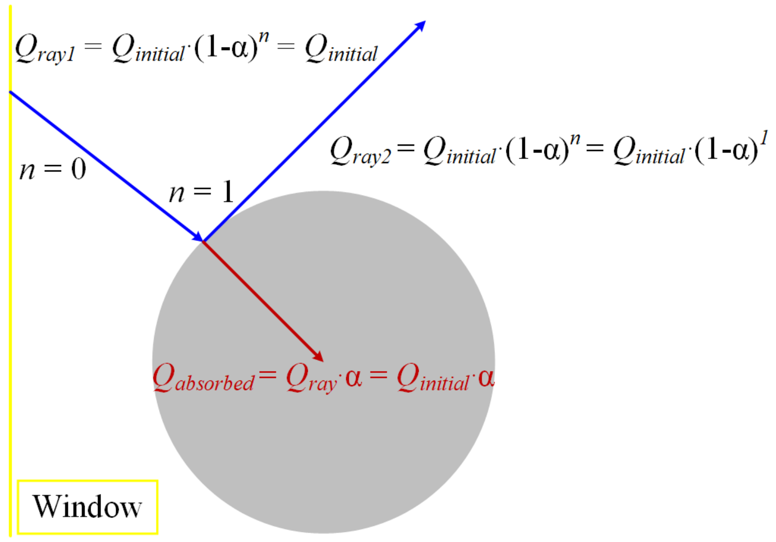

$Q_{ray1} = Q_{initial} \cdot (1\text{-}\alpha)^n = Q_{initial}$
$n = 0$
$n = 1$
$Q_{ray2} = Q_{initial} \cdot (1\text{-}\alpha)^n = Q_{initial} \cdot (1\text{-}\alpha)^1$
$Q_{absorbed} = Q_{ray} \cdot \alpha = Q_{initial} \cdot \alpha$
Window

Figure 3.3: Scheme of the absorption of a ray

3.1.2 Modeling of the emitted rays

The particles emit an amount of energy that depends on their absolute temperature, following the Stefan-Boltzmann law for a gray body (Equation (2.4)). This energy is also emitted in the form of rays and modeled by straight lines that, as the solar rays do, follow a path that changes its direction when they are reflected and partially absorbed by the particles they find on their way or by the walls, until they are almost completely absorbed or go out the window.

The percentage of the energy emitted by a particle that is absorbed by another particle depends on the emissivity ϵ of the particle that absorbs it, being

$$Q_{absorbed} = Q_{incident} \cdot \epsilon, \tag{3.5}$$

as can be seen in the following scheme

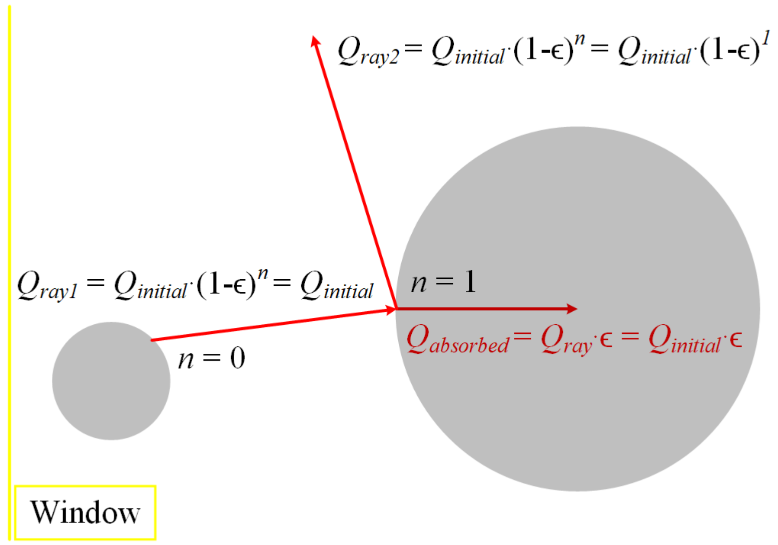

$Q_{ray2} = Q_{initial} \cdot (1-\epsilon)^n = Q_{initial} \cdot (1-\epsilon)^1$
$Q_{ray1} = Q_{initial} \cdot (1-\epsilon)^n = Q_{initial}$
$n = 1$
$n = 0$
$Q_{absorbed} = Q_{ray} \cdot \epsilon = Q_{initial} \cdot \epsilon$
Window

Figure 3.4: Scheme of the path of an emitted ray

The energy of a ray in any instant can therefore be calculated as

$$Q_{ray} = Q_{initial} \cdot (1-\epsilon)^n \, . \tag{3.6}$$

3.2 The Finite Element Method for Heat Conduction

Discretization methods are commonly used to approximate the PDEs with numerical model equations, which are solved with numerical methods. One of them is the FEM, which divides the domain of study into *E* elements of *n* nodes each, being the analytical solution for the temperature approximated by a numerical solution ($\mathbf{T} \approx T$). When applied to a two dimensional transient Heat Conduction problem, the temperature and temperature gradients for each element can be estimated as

$$T^{(e)}(x,y,t) = \sum_{i=1}^{n} N_i(x,y) T_i(t) \, , \tag{3.7}$$

$$\frac{\partial T^{(e)}}{\partial x}(x,y,t) = \sum_{i=1}^{n} \frac{\partial N_i}{\partial x}(x,y) T_i(t) \text{ and} \tag{3.8}$$

$$\frac{\partial T^{(e)}}{\partial y}(x,y,t) = \sum_{i=1}^{n} \frac{\partial N_i}{\partial y}(x,y) T_i(t) \, , \tag{3.9}$$

being $T_i(t)$ and $N_i(x,y)$ the value of the temperature and the interpolation function at each node, respectively. In matrix notation

$$T^{(e)}(x,y,t) = [N(x,y)]\{T(t)\} \, , \tag{3.10}$$

$$\begin{Bmatrix} \dfrac{\partial T^{(e)}}{\partial x}(x,y,t) \\ \\ \dfrac{\partial T^{(e)}}{\partial y}(x,y,t) \end{Bmatrix} = [B(x,y)]\{T(t)\} \, ,$$

being

$$[N(x,y)] = [N_1 N_2 ... N_n] \, , \tag{3.11}$$

$$[B(x,y)] = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \cdots & \dfrac{\partial N_n}{\partial x} \\ \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \cdots & \dfrac{\partial N_n}{\partial y} \end{bmatrix} \, , \tag{3.12}$$

{T(t)} the vector of the elements nodal temperature, [N] the temperature interpolation matrix and [B] the temperature gradients interpolation matrix.

The element equations are derived using Galerkin's Weighted Residual Method, that minimizes the residual with the interpolation function.

Equation (2.12) governs heat conduction in the problem of study,and can be rewritten as

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) - \rho C_p \frac{\partial T}{\partial t} = 0, \tag{3.13}$$

with initial and boundary conditions

$$T = T_0(x,y) \qquad \text{in } \Omega, \text{ t = 0} \tag{3.14}$$

$$k\frac{\partial T}{\partial x}n_x + k\frac{\partial T}{\partial y}n_y = q \ \text{ on } S_1 \, . \tag{3.15}$$

To derive the element equations from Equations (3.13), (3.14) and (3.15), the approximate behaviour of the temperature within each element must be expressed as in Equation (3.7). Now *Galerkin's Weighted Residual Method* is applied in order to minimize the residual with the interpolation function:

$$\iint_{\Omega^{(e)}} N_i \left[\frac{\partial}{\partial x}\left(k\frac{\partial T^{(e)}}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T^{(e)}}{\partial y}\right) - \rho C_p \frac{\partial T^{(e)}}{\partial t}\right] dxdy = 0. \tag{3.16}$$

To reduce the order of the derivatives and introduce the influence of the natural boundary conditions, the previous equation is integrated by parts. Focusing the attention on its first two terms, and according to

$$\int_a^b udv = uv\Big|_a^b \int_a^b vdu \, , \tag{3.17}$$

where $u = N_i$ and $v = k\dfrac{\partial T^{(e)}}{\partial x}\hat{i} + k\dfrac{\partial T^{(e)}}{\partial y}\hat{j}$.

The obtained result is the *symmetric weak form*

$$\begin{gathered} -\iint_{\Omega^{(e)}} \left(k\frac{\partial T^{(e)}}{\partial x}\frac{\partial N_i}{\partial x} + k\frac{\partial T^{(e)}}{\partial y}\frac{\partial N_i}{\partial y} \right) dxdy - \int_{\Omega^{(e)}} N_i \rho C_p \frac{\partial T^{(e)}}{\partial t} dxdy + \\ \int_{S_1^{(e)}} \left(k\frac{\partial T^{(e)}}{\partial x} n_x + k\frac{\partial T^{(e)}}{\partial y} n_y \right) N_i d\Gamma = 0. \end{gathered} \tag{3.18}$$

The surface integral in the previous equation introduces the natural boundary conditions of Equation (3.15) for the elements on the boundary of Ω

$$k\frac{\partial T^{(e)}}{\partial x} n_x + k\frac{\partial T^{(e)}}{\partial y} n_y = q^{(e)}. \tag{3.19}$$

Substituting Equations (3.7), (3.8) and (3.9) , the result takes the form

$$[K]^{(e)}\{T\}^{(e)} + [M]^{(e)} \left\{ \frac{dT}{dt} \right\}^{(e)} = \{q\}^{(e)} , \tag{3.20}$$

being

$K_{ij} = \displaystyle\iint_{\Omega^{(e)}} \left(k\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + k\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} \right) dxdy$,

$M_{ij} = \displaystyle\int_{\Omega^{(e)}} \rho C_p N_i N_j dxdy$ and

$q_i = \displaystyle\int_{S_1^{(e)}} q N_i d\Gamma$.

Globally, the equation becomes

$$[K]\{T\} + [M] \left\{ \frac{dT}{dt} \right\} = \{q\} , \tag{3.21}$$

where $[K_c]$, $[M]$ and $\{q\}$ are the global conductivity matrix, the global mass matrix and the heat flux stated by the Neumann boundary conditions [30].

Equation (3.21) can be solved with different resolution schemes, that differ in the way they approximate the value of the derivatives. The first derivative at a point in a curve $\phi(x)$ is the slope of the tangent line to the function at that specific point, and it can be expressed as

$$\left(\frac{\partial \phi}{\partial x} \right)_{x_i} = \lim_{\Delta x \to 0} \frac{\phi(x_i + \Delta x) - \phi(x_i)}{\Delta x} \, . \tag{3.22}$$

Its value can be approximated by the slope of a line that passes through two nearby points on the curve. Depending on which these two points are, there are different resolution squemes: the explicit or forward difference solution if the two points are x_i and $x_i + \Delta x$, the implicit or backward difference solution when the points are $x_i \Delta x$ and x_i, the central difference solution when the two points lie on opposite sides of x_i... and many other types. Figure 3.5 shows a graphic example of these schemes.

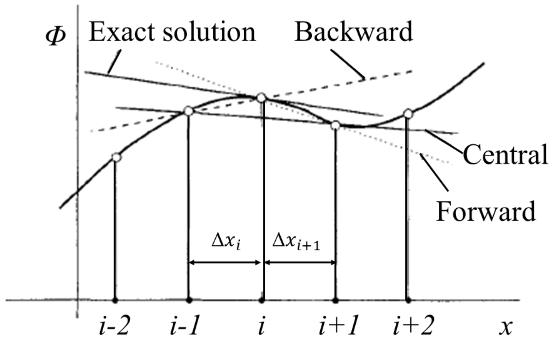


Figure 3.5: Squemes to approximate a derivative.

The approximation improves as the points get closer to each other, which is why grid refinement is an important step in numerical resolutions. Continuous differentiable functions $\phi(x)$ can be expressed as a Taylor series in the proximity of x_i as

$$\phi(x) = \phi(x_i) + (x - x_i)\left(\frac{\partial \phi}{\partial x}\right)_i + \frac{(x-x_i)^2}{2!}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i + \frac{(x-x_i)^3}{3!}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + ... + \frac{(x-x_i)^n}{n!}\left(\frac{\partial^n \phi}{\partial x^n}\right)_i + h.o.t.\,, \tag{3.23}$$

where *h.o.t.* stands for higher-order terms. Approximate expressions for the derivatives at point x_i can be obtained depending on the function values at the points nearby. For example, at x_{i+1}:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1} - \phi_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{2}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(x_{i+1} - x_i)^2}{6}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + h.o.t.\,, \tag{3.24}$$

at x_{i-1}:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_i - \phi_{i-1}}{x_i - x_{i-1}} + \frac{x_i - x_{i-1}}{2}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(x_i - x_{i-1})^2}{6}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + h.o.t.\,, \tag{3.25}$$

and for both x_{i-1} and x_{i+1}:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1} - \phi_{i-1}}{x_{i+1} - x_{i-1}} - \frac{(x_{i+1} - x_i)^2 - (x_i - x_{i-1})^2}{2(x_{i+1} - x_{i-1})}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(x_{i+1} - x_i)^3 + (x_i - x_{i-1})^3}{6(x_{i+1} - x_{i-1})}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + h.o.t.\,. \tag{3.26}$$

If the distance between the grid points is small, the higher-order terms will be small in most cases, and the derivatives can be approximated truncating each of the series after the first terms

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_{i+1} - \phi_i}{x_{i+1} - x_i} \, , \tag{3.27}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_i - \phi_{i-1}}{x_i - x_{i-1}} \, , \tag{3.28}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_{i+1} - \phi_{i-1}}{x_{i+1} - x_{i-1}} \, , \tag{3.29}$$

being these the forward or explicit, backward or implicit, and central-difference schemes. The deleted terms from the right hand sides are the truncation errors, which measure the precision of the approximation and affect how the error is reduced as the spacing between points dicreases. The first truncated term is usually the main source of error.

3.2.1 Grid Refinement

If the grids are fine enough, the truncation error is proportional to the leading term in the Taylor series, and can be expressed as

$$E \approx C(h)^p + h.o.t. \, , \tag{3.30}$$

where *C* is a constant, dependent on the derivatives at the given point, *h* is a measure of grid spacing and *p* is the order of convergence. A second-order solution has a value of $p = 2$.

The initial step in a grid refinement study consists on creating three grids with a different level of refinement and a constant refinement ratio *r* between them. The difference between the solutions on these three grids can be used to estimate the order of convergence *p*. If the grid 1 has a spacing of *4h*, the grid 2 of *2h* and the grid 3 of *h*, since the exact solution may be expressed as

$$\boldsymbol{\phi} = \phi_h + C(h)^p + h.o.t. = \phi_{2h} + C(2h)^p + h.o.t. = \phi_{4h} + C(4h)^p + h.o.t. \, , \tag{3.31}$$

where $\boldsymbol{\phi}$ is exact solution of the partial differential equation, ϕ_h is the solution in a node in grid 3, ϕ_{2h} is the solution in the same node in grid 2 and ϕ_{4h} is the solution in the same node in grid 1, the order of convergence *p* can be estimated as

$$p = \frac{ln\left(\dfrac{\phi_{2h} - \phi_{4h}}{\phi_h - \phi_{2h}}\right)}{ln(r)} \tag{3.32}$$

[31]. Since the error dependence on grid size is usually irregular when the grid is coarse, the grids considered must be sufficiently refined such that the solution is in the *Asymptotic Range of Convergence*, where errors decrease at a rate defined by the order of convergence p. In this range, the grid spacings h and the errors E result in a constant value $C = \frac{E}{h^p}$ [32]. The estimated value of p shows if the studied grids are in said range. For this to be true, p must be approximately the same as order of the scheme.

CHAPTER 4

Numerical Model

4.1 Single particle

As a first step in the creation of the model, the FEM has been used to simulate heat transfer in a single particle. We have considered a simple problem that allows us to create a good model and test it easily.

Since we are working in two dimensions, the particle is modeled by a circle. This circle and its location in space are defined by its centre (determined by its coordinates X and Y), its radius R and the properties of its material. One specific point P in said circle can be defined by its distance from the center r and an angle θ, or simply by its coordinates X and Y.

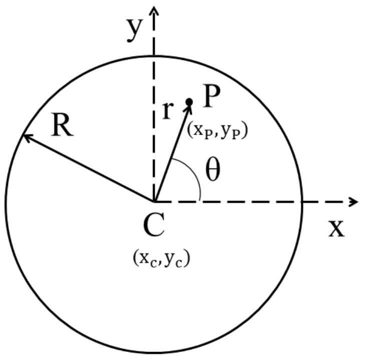


Figure 4.1: Parameters that define the location a particle.

In order to start with a simple model where results can easily be tested, one particle of a generic R = 1 *mm* has been considered. This particle size has been chosen because most particles used in FPR have a size in the range of milimeters. One grid will be created for this particle, that will be properly scaled when working with a different particle size. The material of the particle of study is a kind of sintered bauxite called CARBOHSP, has the following main properties: thermal diffusivity $\alpha = \frac{k}{\rho \, C_p} = 7.41310^{-7} \frac{m^2}{s}$ (where the conductivity is $k = 2\frac{W}{mK}$, the density is $\rho = 3550\frac{kg}{m^3}$ and the specific heat is $C_p = 760\frac{J}{kg°C}$) [33], absorptivity $\alpha = 0.934$ and emissivity $\epsilon = 0.843$ [2].

The time required for the heat to diffuse through the particle (from the contour of the particle to its center) can be calculated from the diffusion equation (2.9) applied to the line that separates said points, squematically represented in Figure 4.2

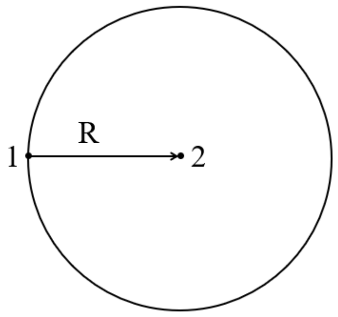


Figure 4.2: Squeme of the distance between the countour and the center of a particle.

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \rightarrow \frac{T_2 - T_1}{R^2} = \frac{1}{\alpha}\frac{T_2 - T_1}{t_s} \rightarrow t_s = \frac{R^2}{\alpha} \rightarrow t_s = \frac{(1 \cdot 10^{-3} m)^2}{7.413 \cdot 10^{-7} \frac{m^2}{s}} = 1.349s$$

4.1.1 Initial and Boundary conditions

The initial condition in this specific problem establishes that the initial temperature in every node is $20°C$, and the boundary conditions, that the particle is being heated by a uniform heat flux *q* of 600000 $\frac{W}{m^2}$ along all its contour.

4.1.2 Analytic solution

The simulation is run during 1.349 *s*, and considering the heat flux that is transmitted to the contour of the particle, the length of said contour and the duration of the transmission, the total heat that the particle receives can be estimated as

$$Q = q \cdot 2\pi r \cdot t = 600000 \frac{W}{m^2} \cdot 2\pi(1 \cdot 10^{-3})m \cdot 1.349s = 5085 \frac{J}{m} \ . \tag{4.1}$$

The specific heat is a property of the material that defines the amount of heat per unit mass necessary to raise the temperature by one degree Celsius. If no change of phase takes place, the relation between heat and temperature change can be expressed as

$$Q = C_p m \Delta T \ , \tag{4.2}$$

being *m* the mass of the body and ΔT the increment of temperature. Reordering the previous equation,

$$\Delta T = \frac{Q}{C_p m} \ . \tag{4.3}$$

Finally, considering that $m = \rho A$ (being A the area), ΔT can be estimated as

$$\Delta T = \frac{Q}{C_p \rho A} = \frac{Q}{C_p \rho \pi r^2} == \frac{5085 \frac{J}{m}}{760 \frac{J}{kg^\circ C} 3550 \frac{kg}{m^3} \pi (1 \cdot 10^{-3} m)^2} = 600^\circ C \; . \tag{4.4}$$

$$T_{final} = T_{initial} + \Delta T = 20^\circ C + 600^\circ C = 620^\circ C \tag{4.5}$$

The temperature reached after applying the heat flux for 1.349 *s* must be approximately $620^\circ C$, so the average temperature obtained with the model must be close to this value.

4.1.3 Grid Refinement study

The particle is divided into *E* elements of *n* nodes each with the grid generator Gmsh. The grid created is made up triangular elements of $n = 3$ nodes in the interior, and linear elements of $n = 2$ nodes on the countour. Grid refinement is carried out to ensure that the results are valid and accurate enough for the purpose of this Thesis and the simulations take as little time as possible.

The procedure explained on section 3.1 of this paper is followed. To start the grid refinement study, three different grids have been generated with Gmsh: grid 1 is the least refined one, and to get grids 2 and 3 a constant refinement ratio $r = 2$ is used between them. The next table sums up the main characteristics of said grids:

	Number of nodes	**Number of triangular elements**
Grid 1	468	868
Grid 2	1801	3472
Grid 3	7073	13888

Table 4.1: Comparison of the characteristics of the three candidate grids.

And figure 4.3 shows a graphic representation of them:

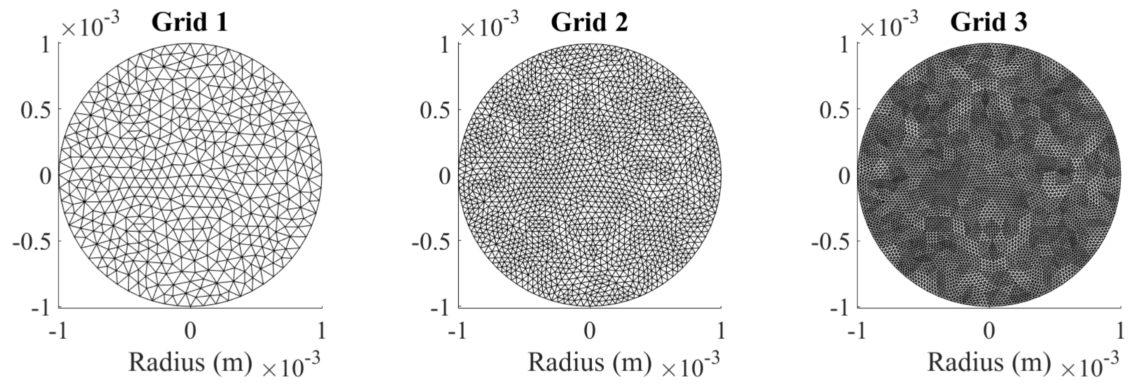


Figure 4.3: Graphic representation of grids 1, 2 and 3.

The program is run in the same conditions for each of the grids, being the value of the temperature in the nodes the parameter of interest to study the grid convergence. Using Equation (3.32) to estimate p in every node shared by the three grids, the obtained values are very close to 2 (the order of the scheme) being its average value for all the nodes 2.0352, which shows that the grids are indeed in the *Asymptotic Range of Convergence*. This can be graphically checked by plotting the results of the temperature in a straight line that crosses the particle going through its center for each of the grids, as shown in Figure 4.4

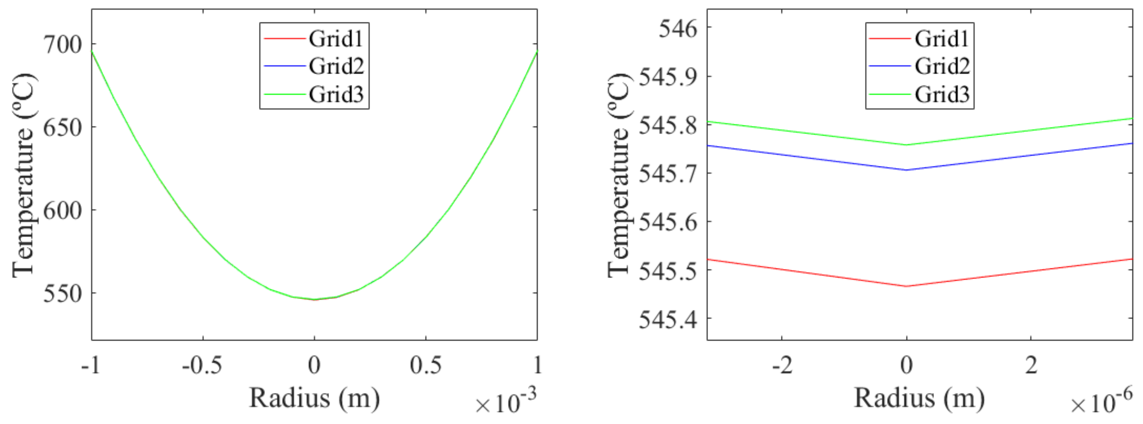


Figure 4.4: Graphic representation of temperature vs. radius through the particle in grids 1, 2 and 3.

It can be easily seen that these solutions do not differ much, especially for the most refined grids: for a temperature increment of approximately $526°C$ in the center of the particle, the solution in grid 1 only varies $0.302°C$ compared to the solution in grid 3 (the most refined one). This is a variation of a 0.000574 %, that can be considered negligible for the purpose of this Thesis. Now that we know approximately what the solution must be, it is important to check if a similar solution can be obtained with a coarser grid, in order to reduce the computational cost. Grid 4 is therefore generated:

	Number of nodes	**Number of triangular elements**
Grid 1	468	868
Grid 2	1801	3472
Grid 3	7073	13888
Grid 4	130	224

Table 4.2: Comparison of the characteristics of the four candidate grids.

In Figure 4.5, the new grid is graphically compared to the previous ones

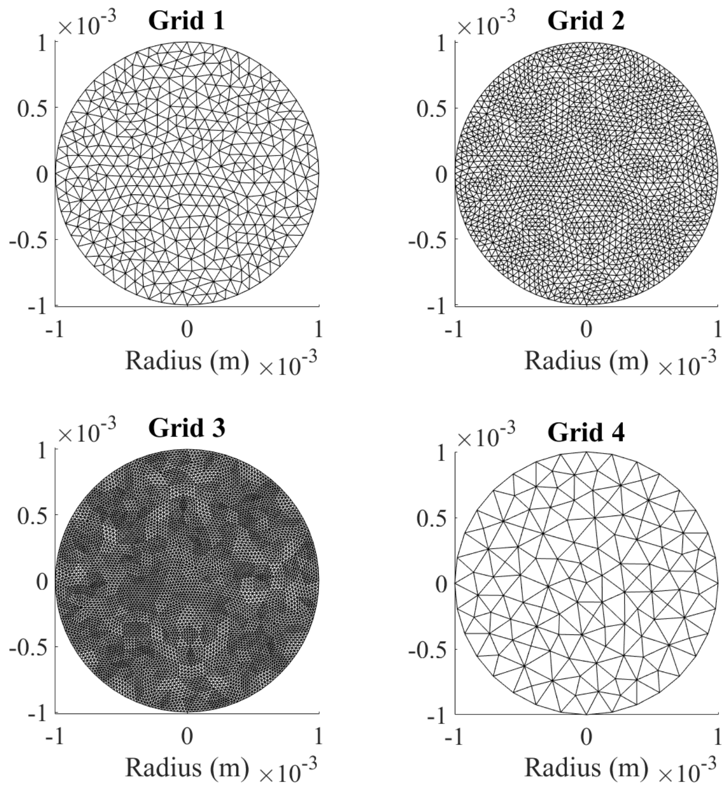


Figure 4.5: Graphic representation of grids 1, 2 and 3.

The results of the temperature in the points of a line that crosses the particle going through its center for this grid and the previous ones are shown in Figure 4.6

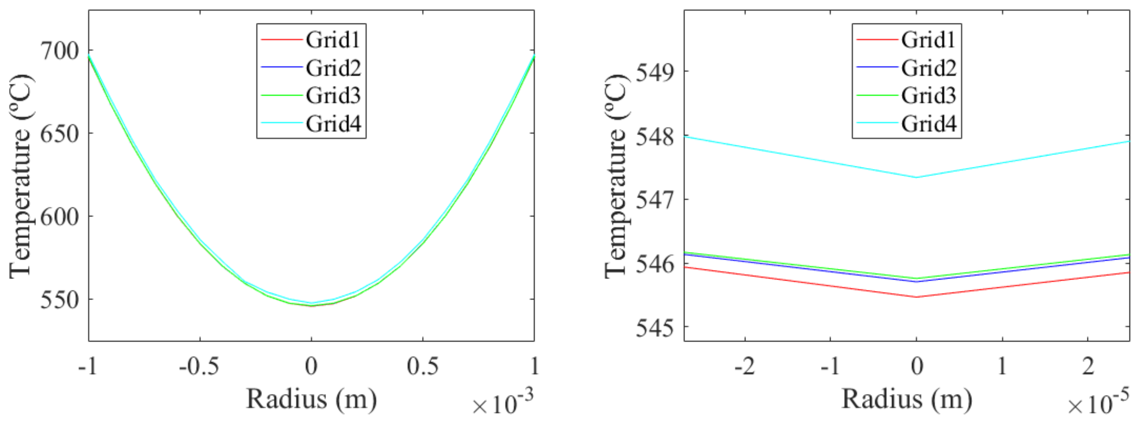


Figure 4.6: Graphic representation of temperature vs. radius through the particlein grids 1, 2, 3 and 4.

This solution differs much more from the solution in grid 3 than grid 1 did, up to $1.572°C$ at some points. That is a difference of a 0.00298 %, a variation 5 times bigger than the obtained when working with grid 1.

Therefore, a new grid is generated to see if it is possible to find greater accuracy with another grid that is still coarser than grid 1. Grid 5 has more nodes that grid 4 but less than grid 1, and its main properties are shown in the following table

	Number of nodes	**Number of triangular elements**
Grid 1	468	868
Grid 2	1801	3472
Grid 3	7073	13888
Grid 4	130	224
Grid 5	314	572

Table 4.3: Comparison of the characteristics of the five candidate grids.

Figure 4.7 shows a graphical comparison of all the grids:

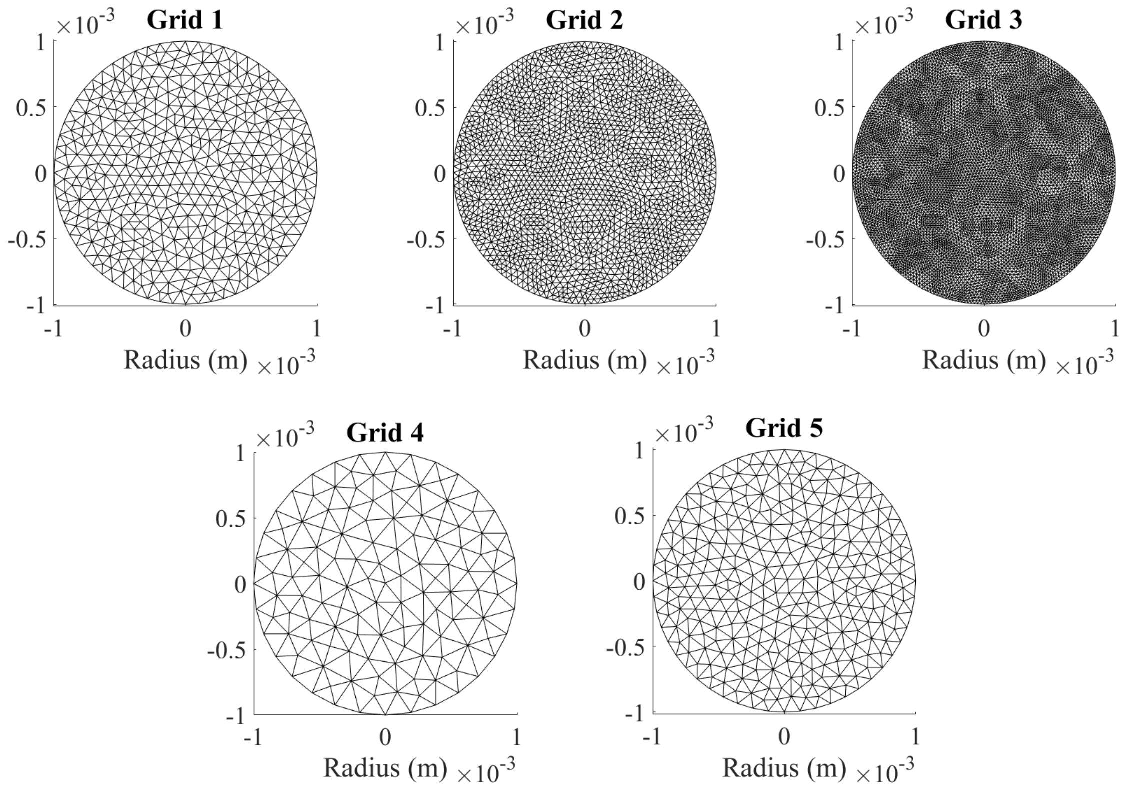


Figure 4.7: Graphic representation of grids 1, 2, 3 , 4 and 5.

The temperature obtained in the points of a line that crosses the particle going through its center for all the studied grids is shown in Figure 4.8

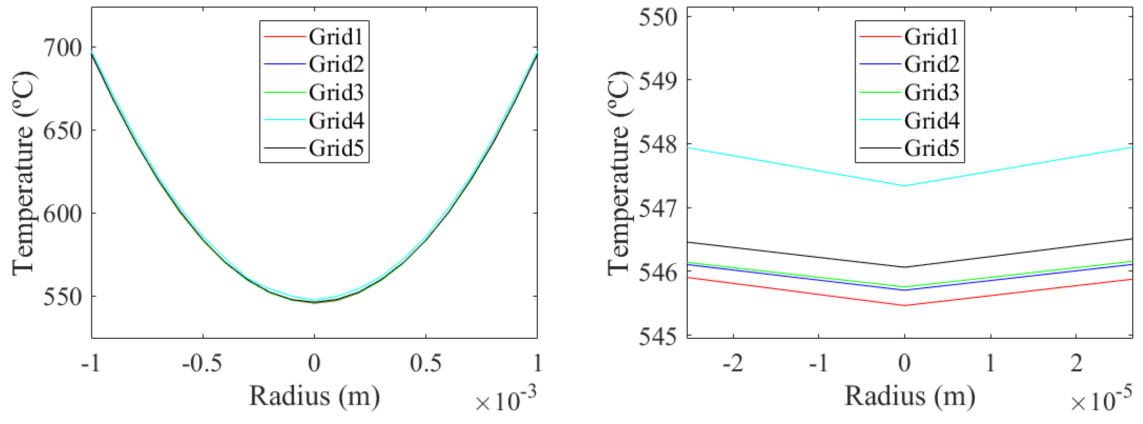


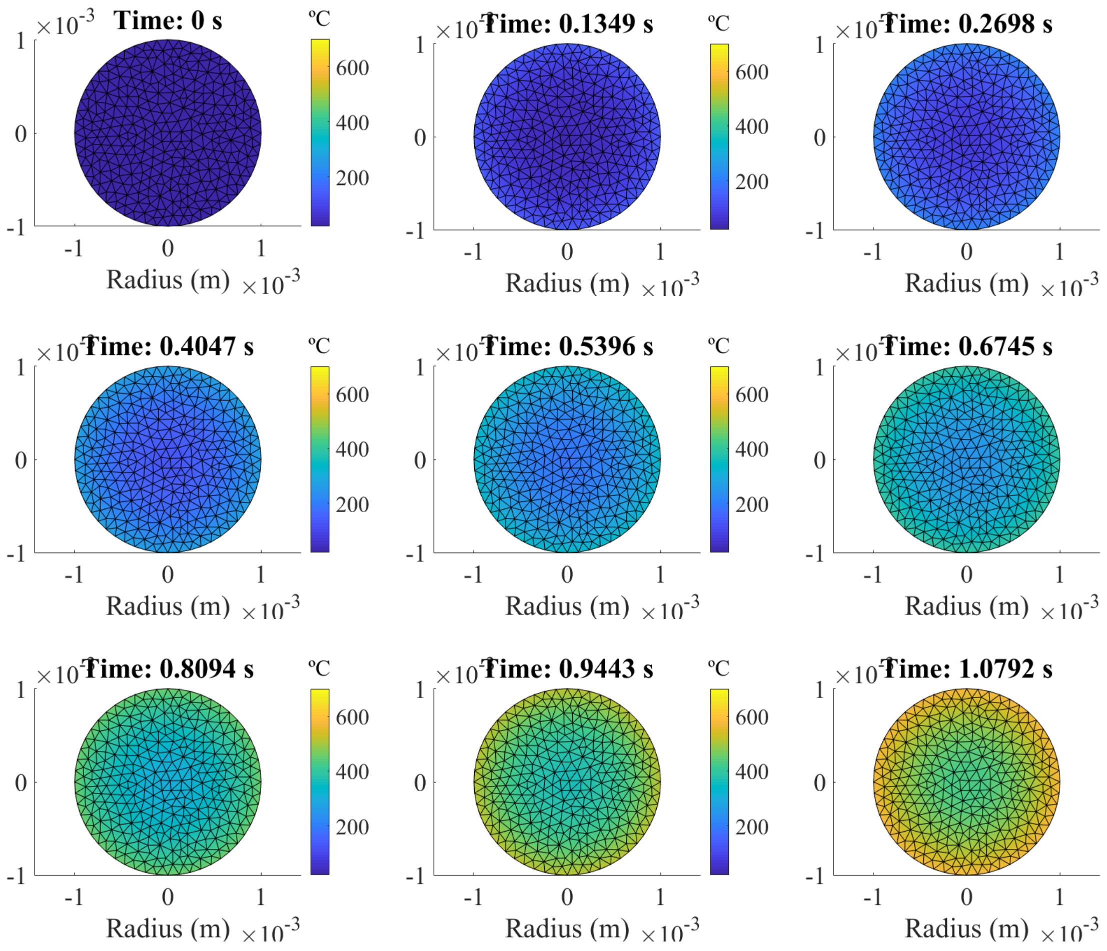
Figure 4.8: Graphic representation of temperature vs. radius through the particlein grids 1, 2, 3, 4 and 5.

The values of the temperature obtained with grid 5 do not differ as much as the previous ones: for a temperature increment of approximately $526°C$ in the center of the particle, the solution in grid 5 only varies $0.297°C$ compared to the solution in the most refined grid. This is a variation of a 0.000564%, that can be considered small enough for the purpose of this Thesis. The computation time has been reduced compared to grid 1: from 1.406672 *s* to 1.339610 *s* in this simulation, which is a reduction of a 5.006%.

4.1.4 Numerical solution

The average temperature of the particle obtained with grid 5 in this experiment is $628.4°C$, which is close enough to the solution calculated analytically.

Figure 4.9 shows the evolution of heat transfer in the chosen grid, calculated with the FEM.



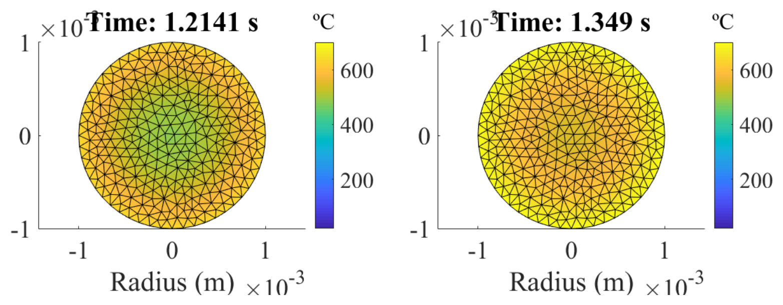


Figure 4.9: Evolution of heat transfer in grid 5.

4.1.5 Emission from the particle. Steady-state

The previous situation was proposed to verify how the FEM model simulates heat transfer, therefore in order to simplifly analytical resolution, emission from the particle was not considered. Taking emission into account would have complicated the analytical calculations of the increment of the energy in the particle, and the scope of the experiment was not to have a realistic result but to find an adequate grid to work with and demonstrate the proper functioning of the FEM on heat transfer. However, to obtain a realistic model of heat transfer it is absolutely necessary to consider the energy emission from the particles, which follows the *Stefan-Boltzmann Law* for a *gray body*, previously expressed on Equation (2.4). As the temperature in the particle increases, its emitted energy increases as well and at some point, a condition of equilibrium is reached. This equilibrium state is called *steady-state*, and it should be properly simulated by the model.
Furthermore, instead of considering a generic particle size of R = 1 *mm*, this example is run with the estimated average size of CARBOHSP R = 0.3495 *mm*.

The temperature evolution is obtained after running a simulation for 8 *s*, in the same conditions of the previous experiment but adding emission from the particle, is shown in Figure 4.10.

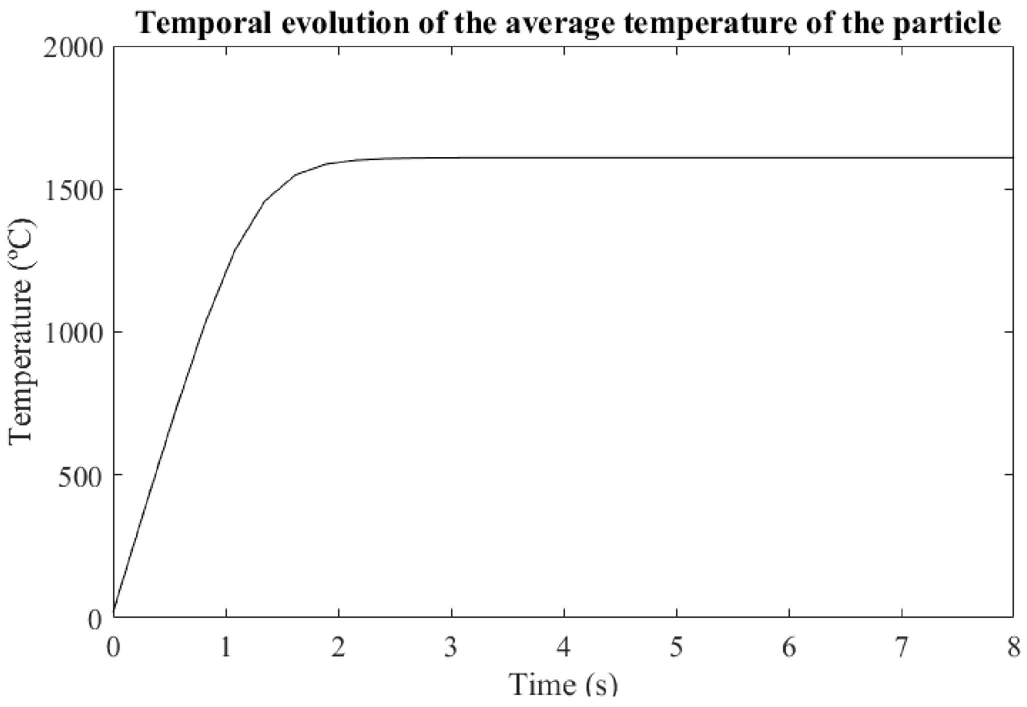


Figure 4.10: Evolution of the temperature in one particle.

This result proves that the program succesfully simulates the reach of the steady-state by the particle.

4.1.6 Solar rays simulation

Now that the working grid has been chosen and it has been demonstrated that the FEM works correctly, the interaction with the modeled rays must be tested. Again, in this first experiment the emission from the particles is not considered, since that would make more difficult to estimate the increment of the temperature in the particle. The simulation is run during 2.69799 s, the total incoming heat flux from the heliostats is considered to be $600000\frac{W}{m}$ (2D), and the section of the window receiver studied has a length of 0.697mm. According to Equation (3.1), the total energy transmitted to the studied domain is

$$Q_{window} = q \cdot l = 600000\frac{W}{m^2} \cdot 0.697 \cdot 10^{-3} m = 418.2\frac{W}{m} = 418.2\frac{J}{m \cdot s} \,. \tag{4.6}$$

Considering the duration of this heat transfer, the total energy that reaches the particle is

$$Q_{particle} = Q_{window} \cdot t = 418.2\frac{J}{m \cdot s} \cdot 0.65534 s = 274.06\frac{J}{m} \,, \tag{4.7}$$

and the increment in the temperature of the particle can be estimated as

$$\Delta T = \frac{Q}{\rho C_p \pi r^2} = \frac{274.06\frac{J}{m}}{3550\frac{kg}{m^3}760\frac{J}{kg^\circ C}\pi(0.3485\cdot 10^{-3}m)^2} = 266.23^\circ C\ . \tag{4.8}$$

$$T_{final} = T_{initial} + \Delta T = 20 + 266.23 = 286.23^\circ C \tag{4.9}$$

Since the initial temperature of the particle is $20^\circ C$, the temperature reached after 0.65534 *s* must be approximately $286.23^\circ C$. The average temperature obtained in the model must be close to this value.

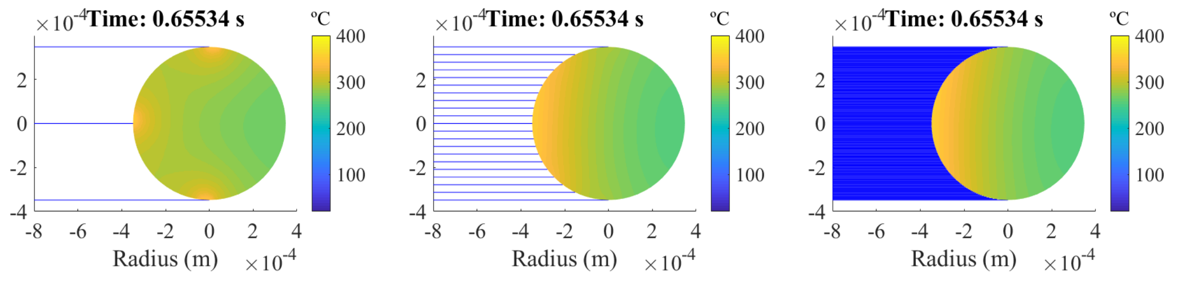
Three different simulations have been run, modeling a different number of rays in each one of them.

Simulation 1 - 3 rays
In the first simulation, one ray has been considered for every 0.3495 *mm* of window. Since the section of the window of interest in this experiment is 0.697 *mm* long, only 3 rays have been simulated. Since the model is two dimensional, these rays must transmit a total of 274.06 *J*, which means that each one of them transmits 91.35 *J*. The final average temperature reached by the particle in this particular case is 287 $^\circ C$. This result can be seen in the subfigure (a) of Figure 4.11.

Simulation 2 - 21 rays
One ray has been simulated for every 0.03495 *mm* of window, which means that in the correspondig section of the window, 21 rays have been simulated, each one of them transmiting 13.05*J*. The final average temperature reached by the particle is 287.12 $^\circ C$. The resut of this simulation is graphically represented in the subfigure (b) of Figure 4.11.

Simulation 3 - 201 rays
One ray has been simulated for every 0.003495 *mm* of window. In the correspondig section of the window, 201 rays have been simulated, each one of them transmiting 1.36*J*. The final average temperature reached is 287.13 $^\circ C$, and the result can be graphically seen in the subfigure(c) of Figure 4.11.

As expected, these three results are close enough to the estimated final temperature for the particle. Figure 4.11 shows a comparison of the heat transfer in these three cases, where the grid lines have been erased for a smoother appearance. It can be seen that, even though the final average temperature is similar in all the cases, the temperature distribution is more realistic when a higher number of rays is simulated.



(a) Simulation 1 (b) Simulation 2 (c) Simulation 3

Figure 4.11: Evolution of the temperature in one particle with a different number rays.

4.2 Multiple particles

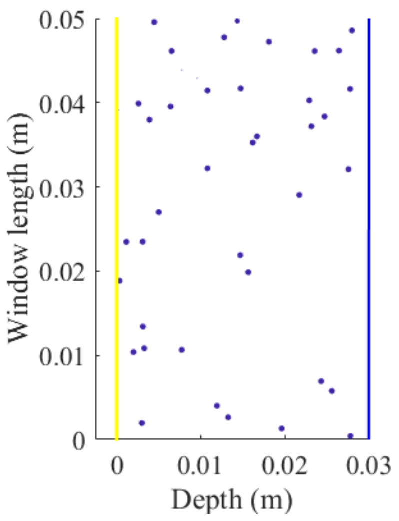
Now that the code has been tried out in simpler situations, it can be developed for the main problem of study in this Thesis: a sample of particles falling inside a FPR that are heated by sunlight. The code created has been divided in six different parts: parts 1 and 2 generate randomly located particles in the computational domain, parts 3 and 4 generate a grid inside each of these particles, part 5 initializes the heat transfer according to the Initial Conditions and part 6 generates a set of rays and uses the FEM to model the heat transfer. A much more detailed explanation of the code can be found on Appendix A.

4.2.1 Generation of the particles

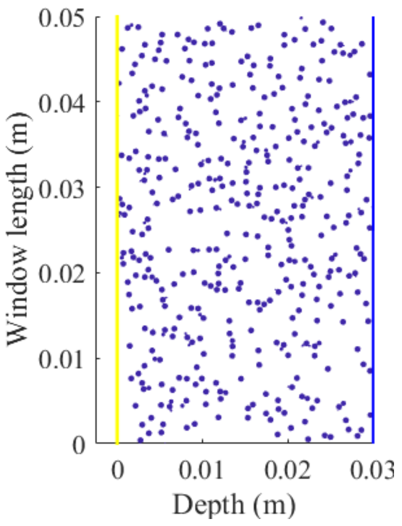
The user of the program must choose the material of the particles of study. According to said material, the values of conductivity k, density ρ specific heat C_p, absorptivity α and emissivity ϵ of the particles must be introduced.
A two band model has been used to model the absorptivity of the particles. The bands are defined by the wavelength of the incident radiation: if it is shorter than 2.5 μm (solar radiation) the absorptivity of the particle is the one defined by the material properties, and if it is longer than 2.5 μm (radiation emitted by other particles) the absorptivity is equal to the particle emissivity.

The user of the program is able to choose the area of the receiver where the particles are created, the void ratio of that area, the distance between the window and the curtain of particles and the location of the back wall of the receiver.

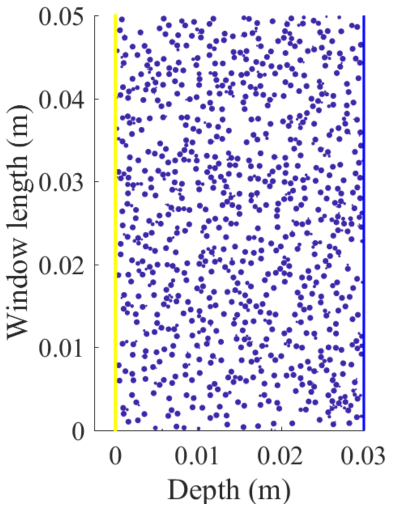
Figure 4.12 shows the particles generated in an area of 15 cm^2 – being the length of the window section simulated 5 *cm* and the depth of the receiver simulated 3 *cm* –, with a distance of 0 *mm* from the window to the curtain of particles, the back wall at a distance of 3 *cm* from the window and a void ratio of a 99 %, 90 % and 80 %, respectively.



(a) Void ratio: 99%



(b) Void ratio: 90%



(c) Void ratio: 80%

Figure 4.12: Generation of particles inside the receiver with a different void ratio.

4.2.2 Generation of the rays

Solar rays are simulated as explained in subsection 3.1.1.

4.2.3 Simulations

Running a simulation for a group of particles as numerous as the ones showed above would take too much time. In order to get some initial results for the developed code in less time, a smaller section of the receiver is modeled.The particles are located in an area of 25 mm^2 – being the length of the window section of said area 5 *mm* and the depth of the receiver 5 *mm* as well –. Three simulations have been run for three groups of particles with a different number of particles, size and spatial distribution.

The group of particles is irradiated with solar rays, that instead of coming only from the section of the window of 5 *mm* in front of which the particles are located, are coming also from from 10 *mm* above and 10 *mm* below. This has been modelesd this way because it is realistic to assume that these particles will receive sunlight coming from different parts of the window.

The total energy coming from these 25 *mm* of window is

$$Q_{window} = q \cdot l = 600000 \frac{W}{m^2} \cdot 2.5 \cdot 10^{-3} m = 1500 \frac{W}{m} = 1500 \frac{J}{m \cdot s} \ . \tag{4.10}$$

The energy emitted by the particles due to their temperature is also simulated in the way explained in subsection 3.1.2.

It is important to highlight that in this models the particles do not move, which might lead to unrealistic results in some particles if they are surrounded by others and do not receive enough sunlight.

4.2.3.1 Simulation 1 - Same sized particles

In this case, four particles have been simulated, all of them with a radius of 0.3495 *mm*, the average value for CARBOHSP particles. The simulation has been run during 8.1 *s*, where 2500 rays have been used to model sunlight (1 ray every 0.01 *mm* of window).

The location of the particles of study can be seen in Figure 4.13.

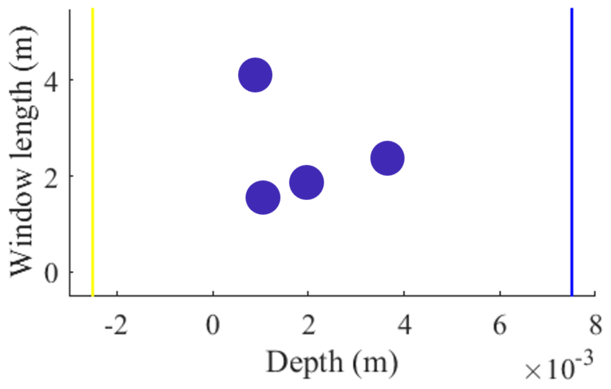


Figure 4.13: Location of the particles of study.

The solar rays are represented by blue lines, as seen in Figure 4.14, which shows a close look of the area of study so that it is possible to appreciate the different lines.

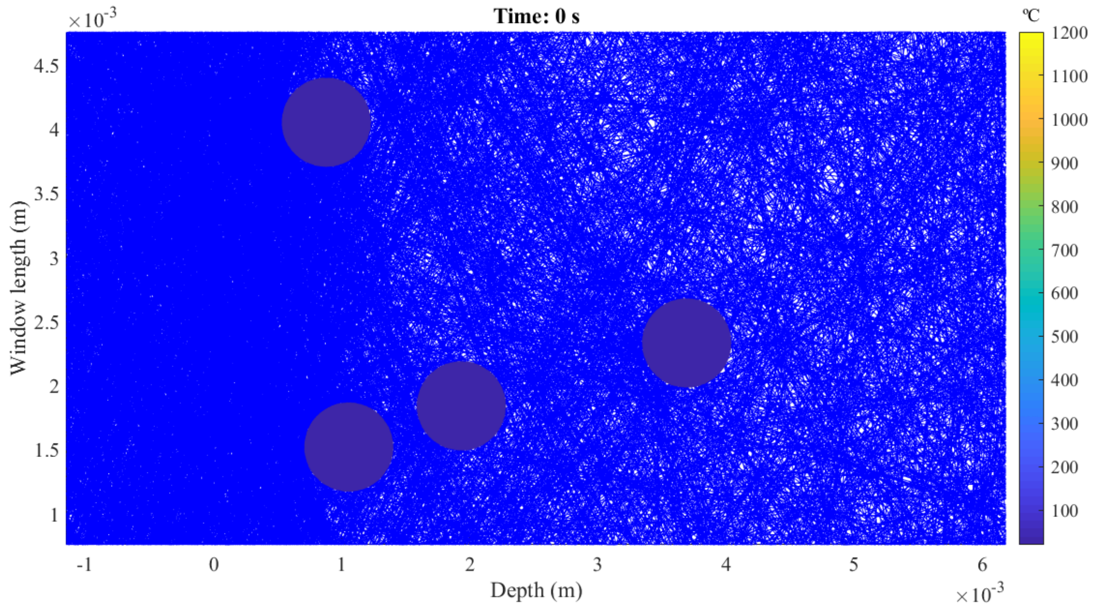


Figure 4.14: Close look of the area of study.

The rays emitted by the particles are represented by red lines, as shown in Figure 4.15.

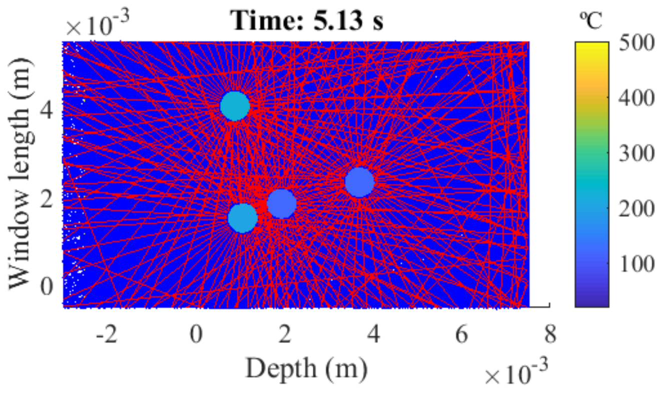


Figure 4.15: Location of the particles in Simulation 1

Both Figures 4.14 and 4.15 show the rays at a specific moment in time. These are not static, but change their paths in every timestep.

Figure 4.16 shows the evolution of the temperature of the particles after the simulation of 8.1 s. The two particles that are closer to the window are heated faster than the other two, which makes sense because they receive more solar rays. However, it looks like the particles begin to reach the steady state at around 240 $°C$, a temperature way below the one that is desired to reach.

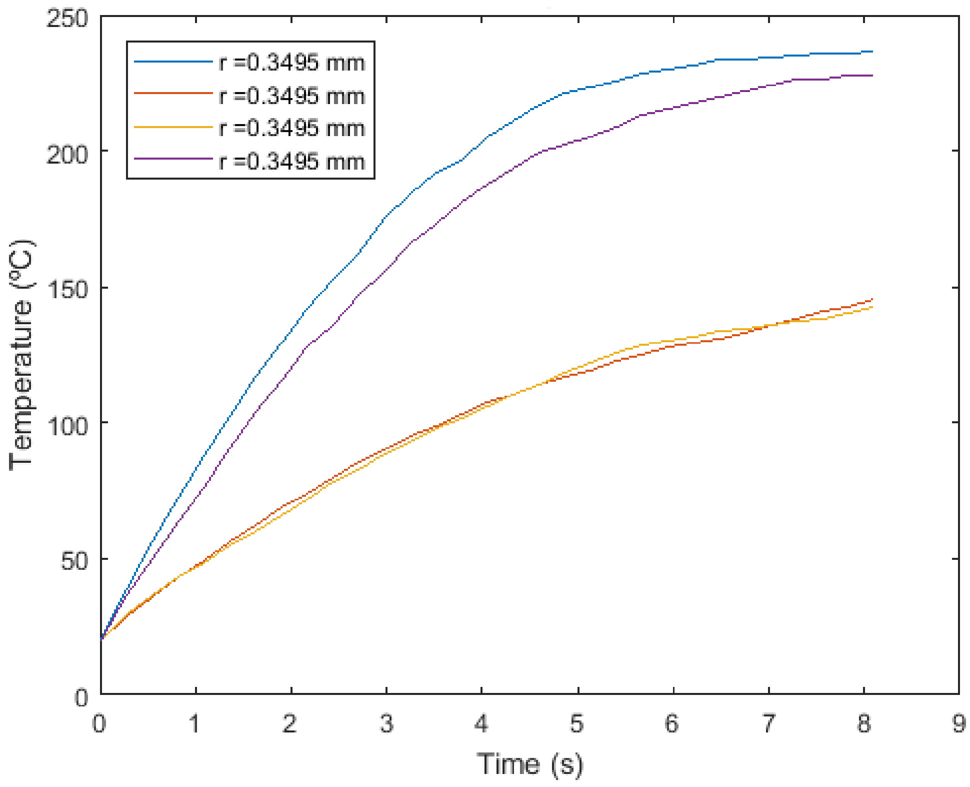


Figure 4.16: Temporal evolution of the average temperature of the particles

4.2.3.2 Simulation 2 - Different sized particles

Seven particles of various sizes have been simulated. The simulation has been run during 13.8 *s* and, as in the previous case, 2500 rays have been used to model sunlight (1 ray every 0.01 *mm* of window).

The location of the particles is shown in Figure 4.17 and the evolution of their temperature, in Figure 4.18.

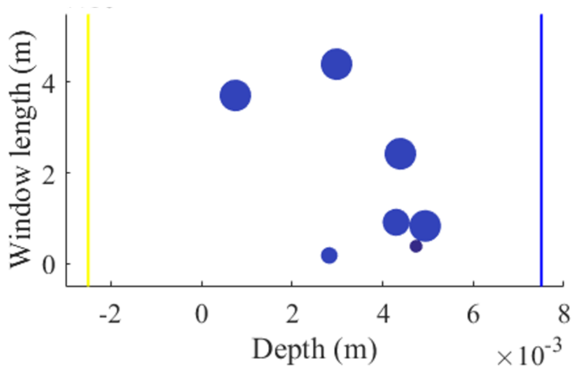


Figure 4.17: Location of the particles in Simulation 2

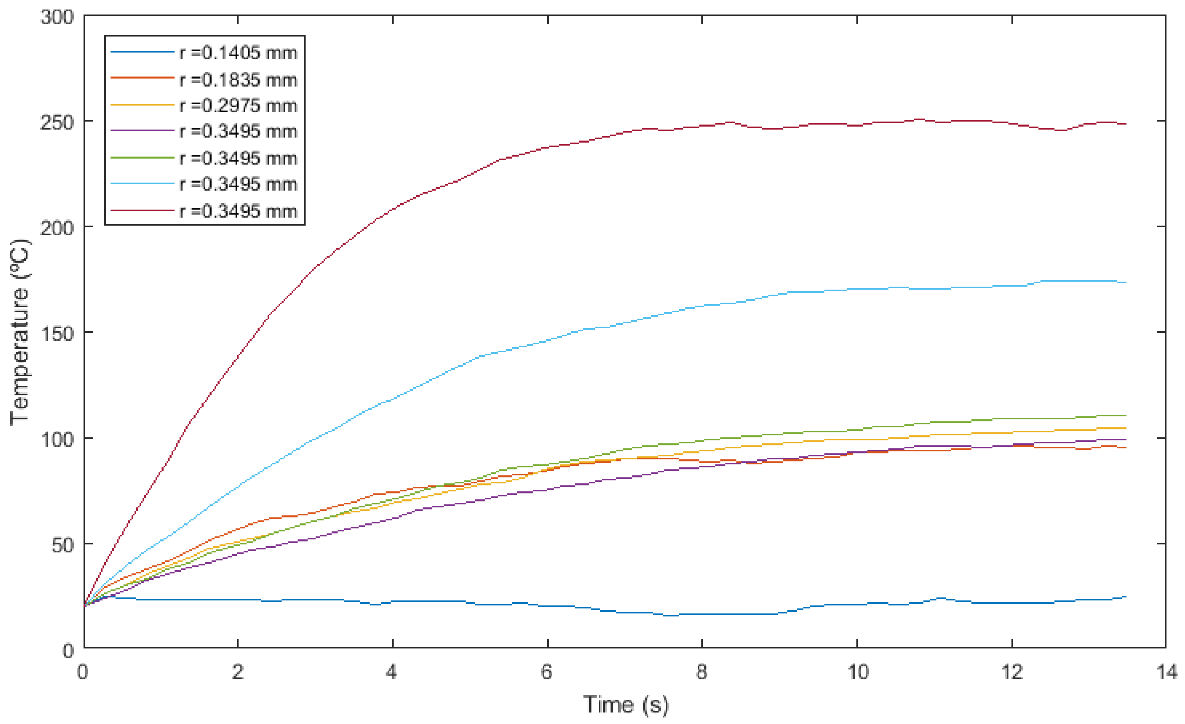


Figure 4.18: Temporal evolution of the average temperature of the particles

The result obtained in the particle of R = 0.1405 *mm* (the dark blue line in Figure 4.18) stands out, since the temperature of the particle does not increase, but remains almost stable. This is due to its location: since it is surrounded by other particles, it does not get enough sunlight to increase its temperature. It emitts radiation due to its temperature and absorbs radiation emitted by other particles and some sunlight, which allows it to keep a temperature somehow stable. This would not occur if particle flow had been simulated, because the particle would not have been surrounded by other particles all the time.

4.2.3.3 Simulation 3 - One particle smaller than the others

Four particles have been created, being one of them slightly smaller than the others. The simulation has been run during 8.1 *s* and in this last case, 5001 rays have been used to model sunlight (1 ray every 0.005 *mm* of window). This is two times more rays than in the previous simulations to model the same amount of energy, which will lead to a more accurate result but at the expense of more computational time.

The location of the particles is shown in Figure 4.19 and the evolution of their temperature, in Figure 4.20.

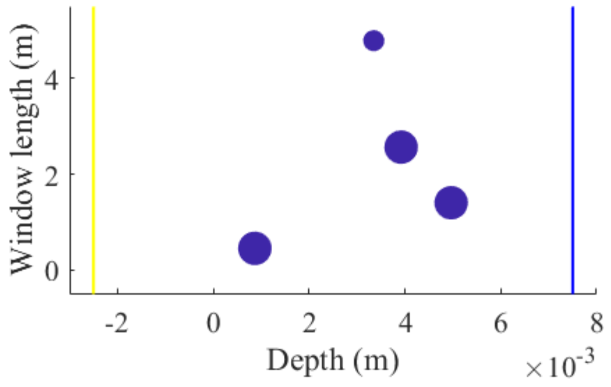


Figure 4.19: Location of the particles in Simulation 3

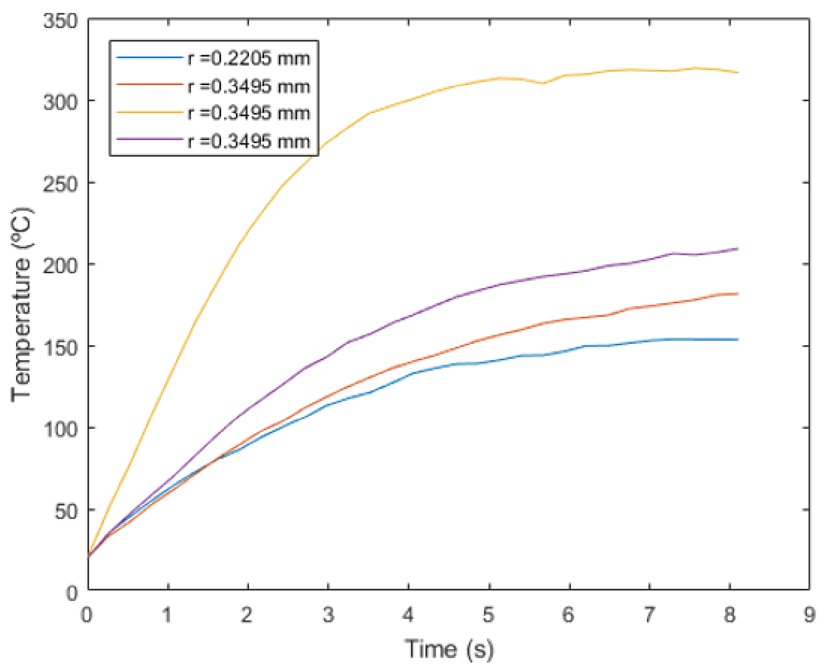


Figure 4.20: Temporal evolution of the average temperature of the particles

These last results show how all the particles are uniformly heated and how they start reaching the steady state at, as was said when discussing simulation 1, temperatures way below what is desired in a FPR. These results present how the smallest particle is heated in a similar manner as the other three particles, which serves to prove that in simulation 2 the results in the smallest particle where not due to its size, but to its location.

CHAPTER 5

Results and Conclusions

The transient heating process of small particles inside a FPR has been simulated, and the results obtained show that the time necessary for the particles to increase their temperature is not negligible.

Also, according to the results the particles will not reach the desired temperatures in the time it takes for them to fall in the curtain, and might need recirculation in the receiver to reach said temperatures. The particles have the capacity to reach them, as was proven in subsection 4.1.5 when modeling an ideal situation where a single particle is uniformly heated in all its contour (Figure 4.10), but do not get enought sunlight when being irradiated only from one side as they fall.

The program created consists on two different parts: the radiation model and the conduction model, and both of them can be individually used to model different problems or technologies.

CHAPTER 6

Future Work

The following improvements to the model are proposed:

– The addition of particle flow, so that the location of the particles changes and the results obtained are more realistic.

– The consideration of heat losses in the back wall, since in this model it has been assumed that the wall was ideal and did not absorb any of the radiation.

– The limitation of the direction of the incoming rays with an appropiate study of the angles of incidence of the concentrated rays from the heliostat field.

– The simulation with different particle materials.

– And finally, the extension of the model to three dimensions.

BIBLIOGRAPHY

[1] German Aerospace Center. Concentrating solar power now. clean energy for sustainable development. The Federal Ministry for the Environment, Nature Conservation and Nuclear Safety (BMU), 2002.

[2] C. Ho, J. Christian, D. Gill, A. Moya, S. Jeter, S. Abdel-Khalik, D. Sadowski, N. Siegel, H. Al-Ansary, L. Amsbeck, B. Gobereit, and R. Buck. Technology Advancements for Next Generation Falling Particle Receivers. *Energy Procedia*, 49:398–407, 2014.

[3] Zina Deretsky Website.

[4] Y. A. Cengel, R. H. Turner, and J. M. Cimbala. *Fundamentals of Thermal-Fluid Sciences*. McGraw-Hill Higher Education, 2001.

[5] M.F. Hordeski. *New Technologies for Energy Efficiency*. Marcel Dekker Inc., 2003.

[6] World energy resources: A summary. Technical report, World Energy Council, 2013.

[7] G. Gereffi and K. Dubay. Concentrating solar power clean energy for the electric grid. Technical report, Center on Globalization, Governance and Competitiveness, 2008.

[8] A. Oles. *Modeling of Falling-Particle Solar Receivers for Hydrogen Production and Thermochemical Energy Storage*. PhD thesis, University of Maryland, 2014.

[9] GE Renewable Energy. *Molten Salt Central Receiver*, 2016.

[10] H. Müller-Steinhagen and F. Trieb. Concentrating solar power - A review of the technology. *Quarterly of the Royal Academy of Engineering*, 18:43–50, 2004.

[11] T. Tan and Y. Chen. Review of study on solid particle solar receivers. *Renewable and Sustainable Energy Reviews*, 14(1):265–276, 2010.

[12] K.Ho. Clifford. A Review of High-Temperature Particle Receivers for Concentrating Solar Power. *Applied Thermal Engineering*, 109:958969, 2016.

[13] A. Meier. A predictive CFD model for a falling particle receiver/reactor exposed to concentrated sunlight. *Chemical Engineering Science*, 54(13-14):2899–2905, 1999.

[14] P.K. Falcone, J.E. Noring, and J.M. Hruby. Assessment of a Solid Particle Receiver for a High Temperature Solar Central Receiver System. Technical Report SAND85-8208, Sandia National Laboratories, 1985.

[15] J.W. Griffin and K.A. Stahl. Optical Properties of Solid Particle Receiver Materials: Angular Scattering and Extinction Characteristics of Norton Masterbeads ®. *Solar Energy Materials*, 14:395–416, 1986.

[16] J.M. Hruby. A Technical Feasibility Study of a Solid Particle Solar Central Receiver for High Temperature Applications. Technical Report SAND86-8211, Sandia National Laboratories, 1986.

[17] G. Evans, W. Houf, R. Grief, and C. Crowe. Numerical Modelling of a Solid Particle Solar Central Receiver. *Sandia National Laboratories*, 1985.

[18] G. Evans, W. Houf, R. Greif, and C. Crowe. Gas-Particle Flow Within a High Temperature Solar Cavity Receiver Including Radiation Heat Transfer. *Journal of Solar Energy Engineering*, 109:134142., 1987.

[19] J. Hruby, R. Steeper, G. Evans, and C. Crowe. An experimental and numerical study of flow and convective heat transfer in a freely falling curtain of particles. *Journal of Fluids Engineering*, 110:172181, 1988.

[20] H. Chen, Y. Chen, H-T. Hsieh, G. Kolb, and N. Siegel. Numerical investigation on optimal design of solid particle solar receiver. *Proceedings of ASME Energy Sustainability*, (ES2007-36134):971–979, 2007.

[21] H. Chen, Y. Chen, H-T. Hsieh, and N. Siegel. Computational fluid dynamics modeling of gas-particle flow within a solid particle solar receiver. *Journal Solar Ener*, 120(2):160170, 2007.

[22] N. Siegel and G. Kolb. Design and On-Sun Testing of a Solid Particle Receiver Prototype. *Proceedings of ASME 2nd International Conference on Energy Sustainability*, (ES2008-54090):329–334, 2008.

[23] C.K. Ho, S.S. Khalsa, and N.P. Siegel. Modeling On-Sun Tests Of A Prototype Solid Particle Receiver for Concentrating Solar Power Processes and Storage. *Proceedings of ASME*, pages ES2009–90035, 2009.

[24] K. Kim, N. Siegel, G. Kolb, V. Rangaswamy, , and S.F Moujaes. A Study of Solid Particle Flow Characterization in Solar Particle Receiver. *Solar Energy*, 83(10):1784–1793, 2009.

[25] S.S. Khalsa and C.K. Ho. Development of a solar patch calculator to evaluate heliostat-field irradiance as a boundary condition in cfd models. *Proceedings of the ASME 2010 4th International Conference on Energy Sustainability*, 2(ES2010-90052):483490, 2010.

[26] C.K. Ho S.S. Khalsa. Radiation Boundary Conditions for Computational Fluid Dynamics Models of High-Temperature Cavity Receivers. *Journal of Solar Energy Engineering*, 133(3):031020, 2011.

[27] N. Siegel, C. Ho, S. Khalsa, and G. Kolb. Development and Evaluation of a Prototype Solid Particle Receiver: On-Sun Testing and Model Validation. *Journal of Solar Energy Engineering*, 132:021008, 2010.

[28] W. Wu, R. Uhlig, R. Buck, and R. Pitz-Paal. Numerical Simulation of a Centrifugal Particle Receiver for High-Temperature Concentrating Solar Applications. *Numerical Heat Transfer Part A*, 68(2):133–149, 2015.

[29] Pablo Fernndez del Campo. Numerical-Stochastic Modeling, Simulation and Design Optimization of Small Particle Solar Receivers for Concentrated Solar Power Plants. Master's thesis, University of Valladolid and San Diego State University, 2013.

[30] K.H. Huebner, D.L. Dewhirst, D.E. Smith, and T.G. Byrom. *The Finite Element Method*

for Engineers. John Wiley & Sons Inc., 3rd edition, 2001.

[31] M. Peric J.H. Ferziger. *Computational Methods for Fluid Dynamics*. Springer, 3rd edition, 2002.

[32] NPARC alliance CFD verification and validation Web Site - Tutorial on CFD Verification and Validation. NASA.

[33] C.ho J. Christian. Alternative designs of a high efficiency, north-facing, solid particle receiver. *Energy Procedia*, 49:314–323, 2014.

APPENDIX

A - Manual of the code

A - Manual of the code

The following code simulates the heat transfer in a sample of particles inside a FPR.
It has been divided in five parts:
Part 1: Generation of randomly located particles with different radii
Part 2: Relocation of the generated particles to the axes of interest
Part 3: Generation of a mesh inside the particles
Part 4: Initial Solution
Part 5: Heat transfer in the particles
Part 5.1: Generation of random rays and their reflections
Part 5.2: Heat Transfer resolution with the FEM

A.1 Explanation of the different parts

Part 1: Generation of randomly located particles with different radii

The original version of this part of the code was developed by Andrea Chiarelli, Andrew Dawson, Alvaro Garcia from The University of Nottingham (The MIT License).
Its purpose is to generate particles randomly located in a two-dimensional area, that do not overlap each other and have a random raius inside a chosen range. The code has been slightly modified, and the parameters have been chosen to meet the requirements of the present Thesis.
The process of generation of particles in the desired area is a little complicated, since there are some limitations derived from the use of an already existing code. I has been decided that the easiest way to go is to, first, generate the particles in an area that does not lead to any problems in the original code and, secondly, relocate said particles in the real area of interest.
Thereby, the particles are initially generated in a area of 3x5 *m*, and have a radius in the range of *cm*. Since this dimensions are obviously way too big for the area and particles that we want to study, everything must be scaled and relocated, which is later done in Part 2.
It is important to highlight that the user can choose the maximum and minum radii that desires for the particles, introducing said values in the parameters max_radius and min_radius in *mm*. In order to generate and study a manageable number of particles, the target planar void ratio used in the example simulations has been between 99% and 90% ($target\_planar\_void\_ratio$ from 99 to 90), a parameter that can easily be changed by the user according to his needs if desired.

Part 2: Relocation of the generated particles to the axes of interest

The first step in this part of the code is the definition of the parameter $scale$ = 0.01. Multiplying the dimensions obtained in Part 1 by this parameter, we obtain the units that we desired: the unit *m* becomes *cm* and the unit *cm* becomes *0.1* mm. This means that the area where particles are generated is now a rectangle of 3x5 *cm*, and the radii of the particles is in the range of 0.1 *mm*. The process carried out in this part is simply a relocation of the centers of the particles in the area of 3x5 *cm*, taking into account geometric considerations.

Part 3: Generation of a mesh inside the particles

This section of the code has been developed in combination with a code created by Emmanuel Lefrançois from the Université de Technologie de Compiegne for the resolution of conduction heat transfer using the FEM.
With the mesh generator Gmsh, a default grid has been created for a generic circle of R = 1 *mm*, as was explained in section 4.1.3 of this report. This same grid is implemented in every particle studied, but the position of the nodes and elements must be properly relocated for each of them, according to their particular radii.
The program reads a file named *circle.msh* created with Gmsh generator that has a specific format, and then the process followed is just a relocation of the grids considering the centers of the particles, and of the position of the nodes considering the radii of the particles (scaling).
If the user wants to work with another grid, he must provide a file named *circle.msh* created with Gmsh generator, and in order for it to have the correct format he must make some modifications to it.
First, the user must check if the number of nodes is written in the 5th row of the file and followed by the number and coordinates that correspond to all the nodes, like this:

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
314
1 0 0 0
2 0.001 0 0
3 0 0.001 0
4  0.001 0 0
5 0  0.001 0
...
```

If there are more lines above, the user must change the file *get_mesh.m* so that it ignores all those lines and the first thing it reads is the number of nodes.
If the file *circle.msh* has the number of nodes in the 5th row as said above, the file *get_mesh.m* ignores the lines above this way:

```
fid = fopen(filename);

% Lecture of the fist 4 uselessl lines
for i = 1:4
    s = fgetl(fid);
end
...
```

So if, for example, the number of nodes was written in the 15th row of the file *circle.msh*, the file *get_mesh.m* should be:

```
fid = fopen(filename);

% Lecture of the fist 14 useless lines
for i = 1:14
    s = fgetl(fid);
end
...
```

Secondly, the user must change the numbers of the file that correspond to the elements, so that they have this format:

```
$Elements
624
1   1 2 11 1 2 6
2   1 2 11 1 6 7
3   1 2 11 1 7 8
4   1 2 11 1 8 9
5   1 2 11 1 9 10
6   1 2 11 1 10 11
7   1 2 11 1 11 12
8   1 2 11 1 12 13
9   1 2 11 1 13 14
10 1 2 11 1 14 15
...
...
...
53 2 2 31 6 235 79 148
54 2 2 31 6 128 235 148
55 2 2 31 6 74 128 148
56 2 2 31 6 94 193 139
57 2 2 31 6 95 194 138
58 2 2 31 6 79 235 150
59 2 2 31 6 108 150 235
60 2 2 31 6 66 112 184
61 2 2 31 6 87 184 112
62 2 2 31 6 66 282 228
63 2 2 31 6 113 254 175
64 2 2 31 6 115 174 255
...
```

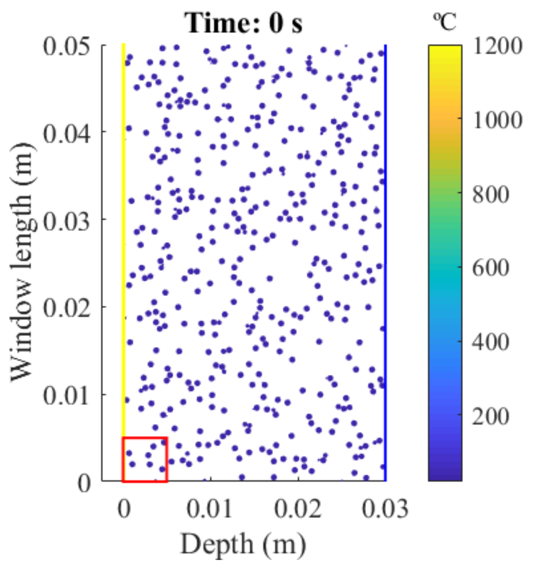
The explanation of this format is, reading the rows of numbers from the left to the right:
The first number is the element number.
The second number is a 1 if the element is a line with $n = 2$ nodes and a 2 if the element is a triangle with $n = 3$ nodes, and will directly come out like this from Gmsh generator.
The third number must be a 2 and will be ignored by the program.
The fourth number must be changed by the user. He must write 11 for the linear elements (being 11 the initial condition that means that no ray has hit the boundary line that surrounds that element), and 31 for the triangular elements (a number later used to obtain the material properties stored in other part of the program). Once the program is run, everytime a ray hits a

linear element the number 11 will be change to a 12.
The next numbers must not be changed, being the last two or three the numbers of the nodes that make up each element.

Part 4: Initial Solution

The initial value of the temperature in every node of every particle is the one introduced by the user as $initial\_temperature$, in $°C$. In all the examples presented in this paper, $initial\_temperature$ = 20. In this part of the program all the particles generated in Parts 1 and 2 will be plotted in their positions – in Figure 1 –, and the smaller area of particles that has been chosen to use for the heat transfer resolution will be plotted too – in Figure 2 –. In both cases, the particles will be shown in the color corresponding to their initial temperature according to the colorbar located on the side of the figures. The vertical yellow line represents the window where rays come from, and the vertical blue line represents the back wall of the receiver cavity, where rays are reflected every time they reach it. The position of the window is established when the user defines the distance between the curtain of falling particles and the window, with the parameter $distance\_window\_curtain$. The curtain of falling particles starts at X=0, so the distance chosen will set the window in a negative position of the coordinate X. The positions of both the window and the back wall are chosen by the user. The position of the back wall in Figures 1 and 2 does not have to be the same. It may be convenient to simulate it closer to the window than it is in reality, since we are working with a reduced model. This position of the back wall in Figure 2 is defined by the user with the parameter $xdest$.

A simulation has been run to exemplify what has been explained above, with the following values for the parameters of interest:
$initial\_temperature$ = 20
$distance\_window\_curtain$ = 0
$target\_planar\_void\_ratio$ = 90
Area where particles are created (not chosen by the user):
$xorigmin\_part$ = 0
$xorigmax\_part$ = 0.030
$yorigmin\_part = xorigmin\_part$
$yorigmax\_part$ = 0.050
Area where the particles will be studied:
$xmin\_area$ = 0
$xmax\_area$ = 0.005
$ymin\_area$ = 0
$ymax\_area$ = 0.005
Area that will be plotted, slightly bigger than the previous one:
$xaxismin\_area$ = -0.0030
$xaxismax\_area$ = 0.008
$yaxismin\_area$ = -0.0005
$yaxismax\_area$ = 0.0055



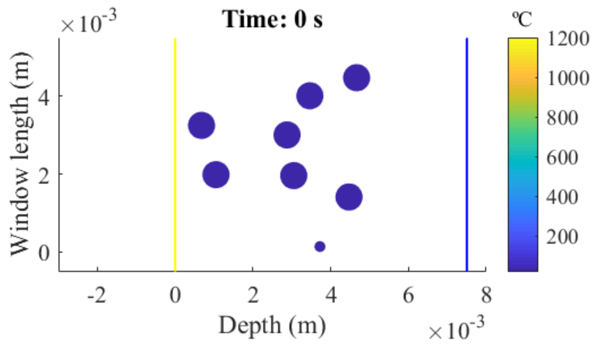


Figure A.1: Example of plotted Figures 1 and 2.

Part 5: Heat transfer in the particles

In this last part of the code, the rays are created and the simulation of radiation and conduction heat transfer takes place. It has been divided in two parts, the first one for the generation of the rays and the second one for the heat transfer modeling.

Part 5.1: Generation of random rays and their reflections

This section is also separated in two parts: one for the solar rays, and one for the rays emitted by the particles because of their temperature.

Solar rays

The solar rays are considered start at the window, and are separated between them a uniform distance dy defined by the user. This means that the origin point of all the solar rays has the same X coordinate, and the Y coordinate of the origin point of every ray varies a distance dy with the previous ray.
The destination point of every ray is a has the same X coordinate as the back wall, and a random Y coordinate inside the range [$ydestmin$,$ydestmax$] defined by the user, that also has a distance of dy between every possible point in said range.
The code has been written in a way that runs ray by ray in a big loop. Inside it, there is another loop that runs every section of the studied ray, being a section the straight line the ray describes between reflection consecutives points. In order to calculate the point where the ray will be reflected, the code equalizes the equation of a line of the studied section of the ray ($y = m \cdot x + n$) with the equation of a circle of every particle ($r = (x - a)^2 + (y - b)^2$). If there are points of the line of the ray that coincide with any points of the circumferences of

the particles, the code chooses the closest point found in this calculation as the next reflection point. This is graphically exemplified in the following Figure A.2, where, after finding four possible intersection points of the ray with the circumferences, the code chooses the closest one and ignores the rest. Then the ray is redirected in a random direction and the process continues.

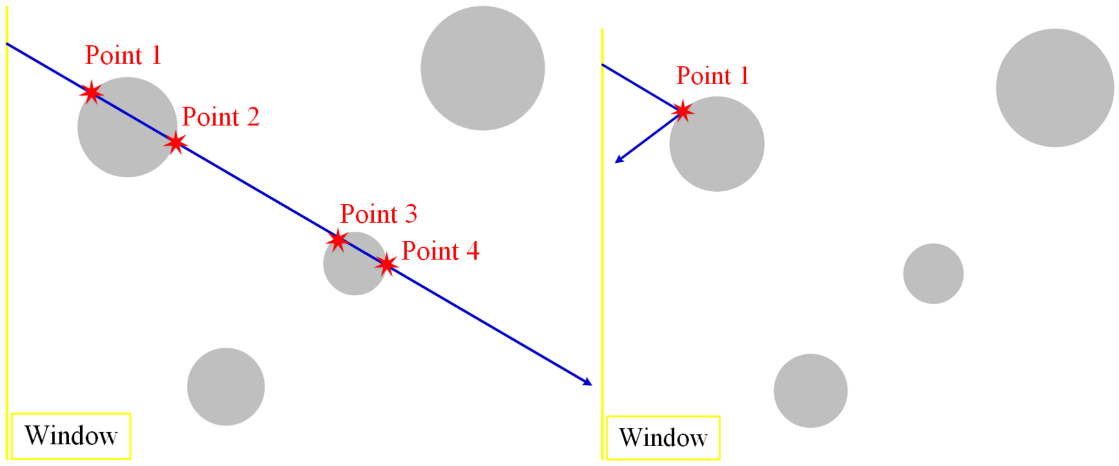


Figure A.2: Choice of the point of reflection of a ray.

If the ray does not find any particle in its way and reaches the back wall, it will be reflected by it. However, if it reaches the window it will not be reflected back into the receiver but will go out of the area of study.The choice of the new direction folowwed by a ray after being reflected by a particle takes various steps. First, an angle α is calculated, in a range from 0° to 360° as shown in Figure A.4.

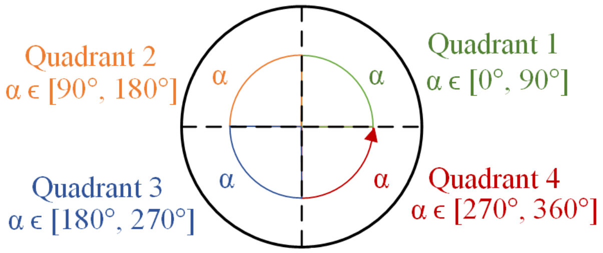


Figure A.3: Angle α from the reflection point.

Secondly, a random angle γ is generated in a range from 0° to 180°.

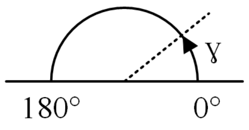


Figure A.4: Random angle γ.

We want to find a destination point for the ray that is realistic. We can combine γ and α in the following manner to obtain a new direction for the ray:
$x_{destinationpoint} = x_{reflectionpoint} + cos(\gamma + \alpha - 90°)$
$y_{destinationpoint} = y_{reflectionpoint} + sin(\gamma + \alpha - 90°)$

However, the destination point obtained this way will be too close to the particle and inside the area of study. This can be graphically seen in the example Figure A.5.

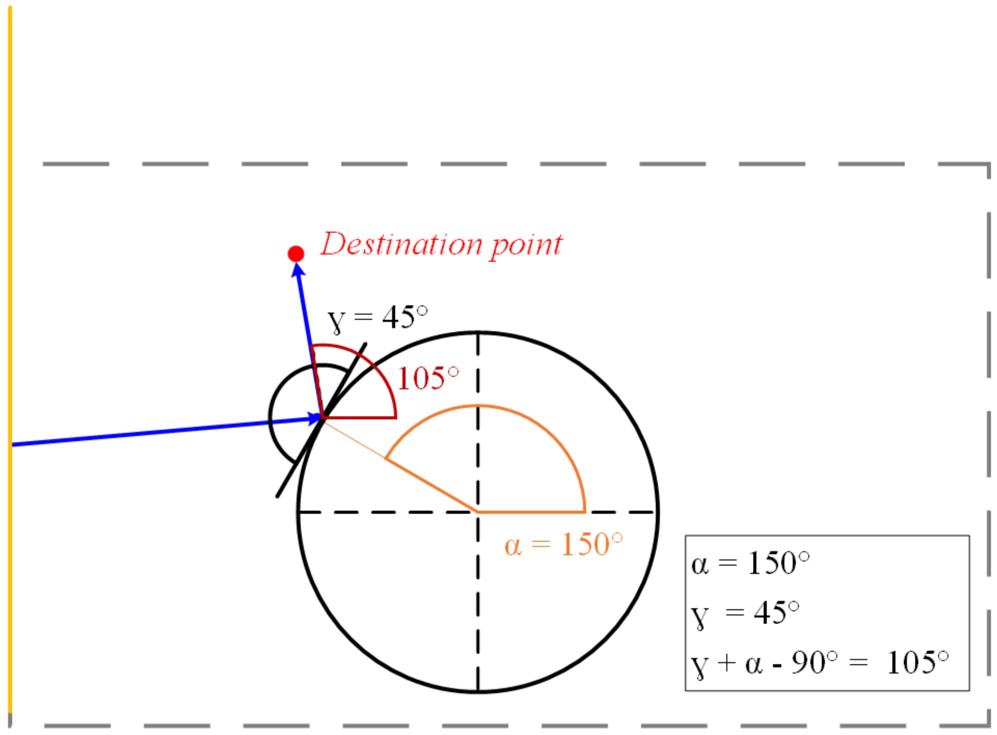


Figure A.5: Example of a destination point after a reflection that would be too close to the particle.

To solve this small problem we multiply by a constant *K* (named k_length in the code), so that the destination point is further from the particle and outside of the area of study:
$x_{destinationpoint} = x_{reflectionpoint} + K \cdot cos(\gamma + \alpha - 90°)$
$y_{destinationpoint} = y_{reflectionpoint} + K \cdot cos(\gamma + \alpha - 90°)$

This final solution is seen in Figure A.6.

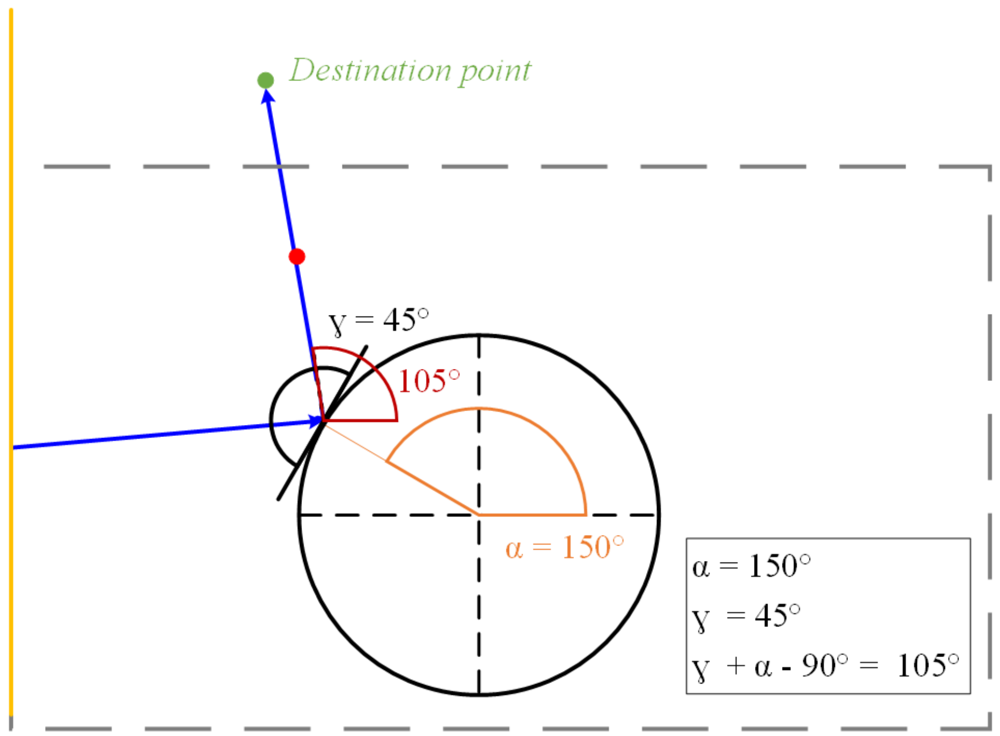


Figure A.6: Destination point after a reflection.

The rays can be reflected as many times as the user desires. Since every time that they are reflected by a particle a big part of their energy is absorbed by it, after a few reflections its energy can be considered negligible. The maximum number of sections allowed for each solar ray is then decided by the user with the parameter $n\_sections\_ray$.
Information about every reflection that has been experimented by a solar ray along its path is stored in the matrix $ray\_intersections$, and information about every reflection of solar rays that has happened in the surface of a particle is stored in the matrix $particle\_intersections$.

Emitted rays

As was explained in section 4.1.3, a grid has been used in every particle to model the conduction heat transfer. The contour of the circular grid is made up of a number of linear elements of $n = 2$ nodes each, while the interior of the particles is made up of triangular elements of $n = 3$ nodes each. In order to model the emission of radiation by the particles, it has been considered that every linear element of the boundary of the particles (that we often refer to as *boundary lines*) emits one ray. Since the chosen grid has 52 linear elements in the contour of the circle (paramenter n_bars in the code), every particle will emit 52 rays due to its temperature.
The energy of these emitted rays is calculated with the Stefan-Boltzann law for gray bodies (Equation (2.4)), using for the value of the temperature the mean between the temperature of the two nodes of each element.
From this point, the emitted rays are modeled just like the solar rays: as straight lines that are reflected and absorbed by other particles, also reflected on the back wall or that go out through

the receiver window. The maximum number of sections allowed for each emitted ray is the same as for the solar rays: $n\_sections\_ray$. Information about every reflection that has been experimented by an emitted ray along its path is stored in the matrix $ray\_intersections\_em$, and information about every reflection of rays emitted by other particles that has happened in the surface of a concrete particle is stored in the matrix $particle\_intersections\_em$.

Part 5.2: Heat Transfer resolution with the FEM

This part of the code models the heat transfer as a result from the interaction between the rays and the particles. Every time that a solar ray hits a particle, information about this fact is stored in $particle\_intersections$ and $ray\_intersections$, and every time that a ray emitted by one particle hits another particle, the information is stored in $particle\_intersections\_em$ and $ray\_intersections\_em$. The information about all the rays emitted by the particles is stored in $emission\_radiation$.
These interactions between rays and particles generate a heat transfer that is modeled with the Finite Element Method. A very important part of this code consists on the treatment on the boundary conditions implemented in the boundary lines of every particle, that change in every time step, due to the impact of rays and the emission of rays. This treatment is explained step by step in the following paragraphs.

The energy absorbed by a particle when it is hit by the first section of one solar ray (which means that it just came from the window and has all the energy) is (representing the initial energy of the ray as E_{solar}):

$$\mathrm{E}_{solar}(W) \cdot \alpha$$

The energy absorbed by a particle when it is hit by the second section of one solar ray is the initial energy that the ray had minus the energy that it has lost when it was absorbed in its first reflection times the absorptivity α:

$$(\mathrm{E}_{solar}(W) - \mathrm{E}_{solar}(W) \cdot \alpha) \cdot \alpha = \mathrm{E}_{solar}(W) \cdot (1-\alpha) \cdot \alpha$$

The energy absorbed by a particle when it is hit by the third section of one solar ray is the initial energy that the ray had minus the energy that it has lost when it was absorbed in its first and second reflections times the absorptivity α:

$$(\mathrm{E}_{solar}(W) - \mathrm{E}_{solar}(W) \cdot (1-\alpha) \cdot \alpha) \cdot \alpha = \mathrm{E}_{solar}(W) \cdot (1-\alpha)^2 \cdot \alpha$$

The energy absorbed by a particle when it is hit by the fourth section of one solar ray is the initial energy that the ray had minus the energy that it has lost when it was absorbed in its first, second and third reflections times the absorptivity α:

$$(\mathrm{E}_{solar}(W) \cdot (1-\alpha)^2 \cdot \alpha) \cdot \alpha = \mathrm{E}_{solar}(W) \cdot (1-\alpha)^3 \cdot \alpha$$

... and so on.
This way, the energy that one boundary line absorbs can be easilly calculated if it is known how many rays in each section hit it. For example, if one specific boundary line is hit by 5

solar rays that are in their first section, 7 solar rays that are in their second section and 2 solar rays that are in their fifth section, the total absorbed energy:

$$\mathrm{E}_{solar}(W) \cdot \alpha \cdot [5 \cdot (1-\alpha)^0 + 7 \cdot (1-\alpha)^1 + 0 \cdot (1-\alpha)^2 + 0 \cdot (1-\alpha)^3 + 2 \cdot (1-\alpha)^4 + 0...]$$

In our code, this absorption is calculated when the Neumann boundary conditions are applied on the boundary lines (in the file $data\_boundaryconditions\_lines.m$).
It is necessary to highlight that the code does not work with $\mathrm{E}_{solar}(W)$, but with $\frac{\text{Flux}(\frac{W}{m^2})}{\text{arc\_length}(m^2)} = \mathrm{E}_{solar}(W)$.
The exact appearance found in the file $data\_boundaryconditions\_lines.m$ of the energy absorbed by a boundary line is

```
flux*absp/arc_length(p,1))*incident_sun(ie,1,p)
```

being incident_sun a sumation in the style of one shown in the example above. The program will make this calculation for every boundary line of every particle.

The calculation of the energy from rays emitted by other particles that is absorbed by one boundary line when it is hit by them follows the same logic. For every ray in every boundary line, the code calculates the energy absorbed as

$$\mathrm{E}_{emission}(W) \cdot (1-\epsilon)^n \cdot \epsilon = \frac{\text{Flux}(\frac{W}{m^2})}{\text{arc\_length}(m^2)} \cdot (1-\epsilon)^n \cdot \epsilon$$

being n the number of times that the ray has been reflected and absorbed by other particles (without including the times that is has been reflected on the wall).
The exact appearance of this calculation in the code is:

```
(particle_intersections_em(row(j,1),4,p)/ arc_length(p,1))*e*((1 e)^n);
```

In order to calculate the total energy that comes in or goes out of a specific boundary line, the code sums the energy absorbed from solar rays and the rays emitted by other particles, and substracts the energy that the boundary line emits because of its temperature:

Total energy absorbed = Energy absorbed from solar rays + energy absorbed from emitted rays - energy emitted

The appeareance of this calculation in the file $data\_boundaryconditions\_lines.m$ is:

```
(flux*absp/arc_length(p,1))*incident_sun(ie,1,p) + emission_radiation(ie,p)
  emission_radiation_incident(ie,p)
```

A.2 Parameters and Matrices

General parameters

$solar\_flux$: Input value. Concentrated solar flux considered in watts.
$n\_reflections$: Input value. Number of sections that will be simulated for each ray.
s_b: StefanBoltzmann constant.
e: Emissivity of the particles.
$absp$: Absorptivity of the particles.
k: Conductivity of the particles in $\frac{W}{m \cdot K}$.
$dens$: Density of the particles in $\frac{kg}{m^3}$.
cp: Specific heat of the particles in $\frac{J}{kg \cdot K}$.

Part 1: Generation of randomly located particles with different radii

Parameters

$xaxismin$: Input value. Minimum value for the axis of the X-coordinate.
$xaxismax$: Input value. Maximum value for the axis of the X-coordinate.
$yaxismin$: Input value. Minimum value for the axis of the Y-coordinate.
$yaxismax$: Input value. Maximum value for the axis of the Y-coordinate.
$target\_planar\_void\_ratio$: Input value. Planar void content as a percentage.
max_radius: Input value. Maximum value desired for the radius od the particles in *mm*.
min_radius: Input value. Minimum value desired for the radius od the particles in *mm*
$origin\_of\_cartesian\_axes$: Input value. Origin of the cartesian axes for the rectangular domain where particles are created (m).
$rectangle\_width$: Rectangle width (m).
$rectangle\_height$: Rectangle height (m).
$number\_of\_domains$: Number of packed domains to generate.
$number\_of\_particles$: Input value. Number of particles in the first generation seeded
$new\_particles\_per\_generation$: Number of particles added in each new generation
$maximum\_radius$: Minimum radius of the generated particles in *m*, directly derived from the value introduced by the user in max_radius.
$minimum\_radius\_seeded$: Maximum radius the generated particles in *m*, directly derived from the value introduced by the user in min_radius.
$counter$: Counter of the number of particles in each iteration.
can_it_grow: Logical condition that will indicate if a particle is allowed to grow.
$squared\_sum\_of\_radii$: Squared sum of the radii of the particles, later used to obtain the total area of the circles.
$n\_particles$: Number of particles that have been generated. This number will change when a smaller number of particles is chosen for the heat transfer study.
$n\_particles\_initial$: Number of particles that were generated initially.**Matrices**

$radius$: Radii of the particles (m).

$$\begin{array}{cc}
\textbf{Particle number} & \textbf{Radius} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} radius_1 \\ radius_2 \\ ... \\ ... \\ radius_n \end{pmatrix}
\end{array}$$

$centre$: X and Y coordinates of the particles' centers.

$$\begin{array}{cc}
\textbf{Particle number} & \begin{array}{cc} \textbf{X-coordinate} & \textbf{Y-coordinate} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}
\end{array}$$

$angles\_for\_parametric\_circle$: Angles for the equation for parametric circumferences.
$x\_coordinates\_circles$: X-coordinates of the particles.
$y\_coordinates\_circles$: Y-coordinates of the particles.

Part 2: Relocation of the generated particles to the axes of interest

Parameters

$xorig$: Input value. Value of the X-coordinate for the origin of the rays.
$xdest$: Input value. Value of the X-coordinate for the destination of the rays.
$yorigmin$: Input value. Minimum value of the Y-coordinate for the origin of the rays.
$yorigmax$: Input value. Maximum value of the Y-coordinate for the origin of the rays.
$ydestmax$: Input value. Minimum value of the Y-coordinate for the destination of the rays.
$ydestmin$: Input value. Maximum value of the Y-coordinate for the destination of the rays.
dy: Input value. Separation between rays in the Y-axis (m).
$yorigmin\_part$: Input value. Lower limit of the Y-coordinate to relocate the particles (m).
$yorigmax\_part$: Input value. Upper limit of the Y-coordinate to relocate the particles (m).
$r\_standard$: Input value. Desired value for the average radius (m).
$scale$: rstandard times 10.
$xmed$: Middle point in the studied range of the X-coordinate.
$ymed$: Middle point in the studied range of the Y-coordinate.
$x1$: X-coordinates of the rays' origins. Constant value.
$y1$: Range for the Y-coordinates of the rays' origins. Equally spaced values.
n_rays: Number of rays.
$x2$: X-coordinates of the rays' destinations. Constant value.
$y2$: Range for the Y-coordinates of the rays' destination. Random value in this range.

Matrices

$centre\_initial$: Coordinates of the centers of the particles generated in the part 1 of the program.
$radius\_initial$: Radii of the particles generated in the part 1 of the program (m).

$$\begin{array}{cc} \textbf{Particle number} & \textbf{Radius} \\ 1 & \\ 2 & \\ ... & \\ ... & \\ \text{n\_particles} & \end{array} \begin{array}{c} \\ \begin{pmatrix} radius_1 \\ radius_2 \\ ... \\ ... \\ radius_n \end{pmatrix} \end{array}$$

$radius$: Scaled radii of the particles (m).

$$\begin{array}{cc} \textbf{Particle number} & \textbf{Radius} \\ 1 & \\ 2 & \\ ... & \\ ... & \\ \text{n\_particles} & \end{array} \begin{array}{c} \\ \begin{pmatrix} radius_1 \\ radius_2 \\ ... \\ ... \\ radius_n \end{pmatrix} \end{array}$$

Part 3: Generation of a mesh inside the particles

Parameters

nnt: Total number of nodes.
n_bn: Number of boundary nodes.
n_bars: Number of boundary lines (elements with $n = 2$ nodes where the boundary conditions are applied).

Matrices

$vcorg\_stored$: Correct coordinates of the nodes.

$$\begin{array}{c} \textbf{Node number} \\ 1 \\ 2 \\ ... \\ ... \\ \text{nnt} \end{array} \begin{array}{c} \begin{array}{cc} \textbf{X-coordinate} & \textbf{Y-coordinate} \end{array} \\ \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix} \end{array}$$

$vcorg\_original$: Coordinates of the nodes before they were moved to its right location for each particle.

$$\begin{array}{c c}
\begin{array}{c} \textbf{Particle} \\ \textbf{number} \end{array} & \begin{array}{cc} \textbf{X-coordinate} & \textbf{Y-coordinate} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}
\end{array}$$

r_1: Distance between the corresponding node and the center of the corresponding particle (m).

$$\begin{array}{c c}
\begin{array}{c} \textbf{Node} \\ \textbf{number} \end{array} & \textbf{Distance} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{nnt} \end{array} & \begin{pmatrix} distance_1 \\ distance_2 \\ ... \\ ... \\ distance_n \end{pmatrix}
\end{array}$$

r_2: r_1 correctly scaled (m).

$$\begin{array}{c c}
\begin{array}{c} \textbf{Node} \\ \textbf{number} \end{array} & \textbf{Distance} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{nnt} \end{array} & \begin{pmatrix} distance_1 \\ distance_2 \\ ... \\ ... \\ distance_n \end{pmatrix}
\end{array}$$

$vcorg\_scaled$: Coordinates of every particle's nodes.

$$\begin{array}{c c}
\begin{array}{c} \textbf{Node} \\ \textbf{number} \end{array} & \begin{array}{cc} \textbf{X-coordinate} & \textbf{Y-coordinate} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{nnt} \end{array} & \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}
\end{array}$$

$boundary\_nodes$: Number of the boundary nodes that form every boundary line.

$$\begin{array}{ccc} \textbf{Boundary line number} & \textbf{Node number 1} & \textbf{Node number 2} \\ \begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_bars} \end{array} & \multicolumn{2}{c}{\begin{pmatrix} Nodenumber & Nodenumber \\ Nodenumber & Nodenumber \\ ... & ... \\ ... & ... \\ ... & ... \end{pmatrix}} \end{array}$$

$boundary\_nodes\_vector$: List of the numbers of the boundary nodes.

$$\begin{array}{c} \textbf{Node number} \\ \begin{pmatrix} 1 \\ 2 \\ ... \\ ... \\ n\_bn \end{pmatrix} \end{array}$$

$vcorg\_scaled\_boundary$: Coordinates of the boundary nodes of every particle.

$$\begin{array}{ccc} \textbf{Node number} & \textbf{X-coordinate} & \textbf{Y-coordinate} \\ \begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_bn} \end{array} & \multicolumn{2}{c}{\begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}} \end{array}$$

arc_length: Length of the boundary lines of every particle (m).

$$\begin{array}{cc} \textbf{Particle number} & \textbf{Arc length} \\ \begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles\_initial} \end{array} & \begin{pmatrix} arclength \\ arclength \\ ... \\ ... \\ arclength \end{pmatrix} \end{array}$$

Part 4: Initial Solution

Parameters

Region where particles will be initially generated, plotted in Figure 1:
$xorigmin\_part$: Minimum value of the X- coordinate.

$xorigmax\_part$: Maximum value of the X- coordinate.
$yorigmin\_part$: Minimum value of the Y- coordinate.
$yorigmax\_part$: Maximum value of the Y- coordinate.
Choice of smallest region inside the previous one where heat transfer will be studied in the particles:
$xmin\_area$: Input value. Minimum value of the X- coordinate.
$xmax\_area$: Input value. Maximum value of the X- coordinate.
$ymin\_area$: Input value. Minimum value of the Y- coordinate.
$ymax\_area$: Input value. Maximum value of the Y- coordinate.
Choice of the region that will be plotted in the study of heat transfer, plotted in Figure 2. It is recommended to choose an area slightly bigger than the one defined above.
$xaxismin\_area$: Input value. Minimum value of the X- coordinate.
$xaxismax\_area$: Input value. Maximum value of the X- coordinate.
$yaxismin\_area$: Input value. Minimum value of the Y- coordinate.
$yaxismax\_area$: Input value. Maximum value of the Y- coordinate.
$initial\_temperature$: Input value. Initial temperature of the particles.
$distance\_window\_curtain$: Input value. Distance in *m* between the window and the curtain of falling particles. This value directly establishes the position of the window, which is the X-coordinate for the origin of the solar rays.
$count\_fig$: Counts the number of figures plotted.
$time\_previous\_iteration$: Time when the previous iteration ended (s).

Matrices

$area\_study\_1$: Checking the centers of every particle, writes a 1 if the coordinates of their centers are bigger than the minimun of the area of interest, and a 0 otherwise.

$$\begin{array}{cc}
\begin{array}{c}\textbf{Particle}\\ \textbf{number}\end{array} & \begin{array}{cc}\begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the X-coordinate}\\ \textbf{of the center}\end{array} & \begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the Y-coordinate}\\ \textbf{of the center}\end{array}\end{array} \\
\begin{array}{c}1\\ 2\\ ...\\ ...\\ \text{n\_particles\_initial}\end{array} & \begin{pmatrix} 0-1 & 0-1 \\ 0-1 & 0-1 \\ ... & ... \\ ... & ... \\ 0-1 & 0-1 \end{pmatrix}
\end{array}$$

$area\_study\_2$: Checking the centers of every particle, writes a 1 if the coordinates of their centers are smaller than the maximum of the area of interest, and a 0 otherwise.

$$\begin{array}{cc} \begin{array}{c}\textbf{Particle}\\ \textbf{number}\end{array} & \begin{array}{cc}\begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the X-coordinate}\\ \textbf{of the center}\end{array} & \begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the Y-coordinate}\\ \textbf{of the center}\end{array}\end{array} \\ \begin{array}{c}1\\2\\...\\...\\ \text{n\_particles\_initial}\end{array} & \begin{pmatrix} 0-1 & 0-1 \\ 0-1 & 0-1 \\ ... & ... \\ ... & ... \\ 0-1 & 0-1 \end{pmatrix} \end{array}$$

$area\_study\_3$: Sums the numbers (0 or 1) obtained in area_study_1 and area_study_2 for the X coordinates of the centers, and the numbers obtained for the Y coordinates of the centers. The results can be 0, 1 or 2.

$$\begin{array}{cc} \begin{array}{c}\textbf{Particle}\\ \textbf{number}\end{array} & \begin{array}{cc}\begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the X-coordinate}\\ \textbf{of the center}\end{array} & \begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the Y-coordinate}\\ \textbf{of the center}\end{array}\end{array} \\ \begin{array}{c}1\\2\\...\\...\\ \text{n\_particles\_initial}\end{array} & \begin{pmatrix} 0-1-2 & 0-1-2 \\ 0-1-2 & 0-1-2 \\ ... & ... \\ ... & ... \\ 0-1-2 & 0-1-2 \end{pmatrix} \end{array}$$

$area\_study\_4$: Sums the numbers obtained in area_study_3 fot both the X coordinates and the Y coordinates. The results can be 0, 1, 2, 3 or 4, being the particles where a 4 is obtained inside the area of interest. particles_area_study: numbers of the particles that have been found inside the area of interest.

$$\begin{array}{cc} \begin{array}{c}\textbf{Particle}\\ \textbf{number}\end{array} & \begin{array}{c}\textbf{Value correspondent}\\ \textbf{to the center}\end{array} \\ \begin{array}{c}1\\2\\...\\...\\ \text{n\_particles\_initial}\end{array} & \begin{pmatrix} 0-1-2-3-4 \\ 0-1-2-3-4 \\ ... \\ ... \\ 0-1-2-3-4 \end{pmatrix} \end{array}$$

$vsol$: Temperature of the nodes ($°C$).

$$\begin{array}{cc} \begin{array}{c}\textbf{Node}\\ \textbf{number}\end{array} & \textbf{Temperature} \\ \begin{array}{c}1\\2\\...\\...\\ \text{nnt}\end{array} & \begin{pmatrix} temperature_1 \\ temperature_2 \\ ... \\ ... \\ temperature_n \end{pmatrix} \end{array}$$

Part 5: Heat transfer in the particles

Parameters

$flux$: Thermal flux in every solar ray (W).
n_rays_em : number of rays emitted by a particle.
$DV1$: Input value. Default value for the number of rows in the matrices: $rays$, $unions$, $tangents$, $true\_xint$, $true\_int$.
$DV2$: Input value. Default value for the number of rows in the matrices: $xint$, XX, YY, DD.
$DV3$: Input value. Default value for the number of rows in the matrix: $particle\_intersections\_em$.
$DV4$: Input value. Default value for the number of rows in the matrix: $particle\_intersections$.

Matrices

$average\_temp$: stores the average temperature of the nodes of every particle every 10 timesteps.
$time\_axis$: stores the time every 10 timesteps.

Part 5.1: Generation of random rays and its reflections

Parameters

$xorig$: X- coordinate where the window is located, which is also the X-coordinate of the origin of the rays.
$xdest$: X- coordinate where the back wall is located, which is also the X-coordinate of the destination of the rays.
$yorigmin$: minumum value of the Y-coordinate of the window from where the solar rays will be originated.
$yorigmax$: maxiumum value of the Y-coordinate of the window from where the solar rays will be originated.
dy: distance between consecutive rays that come from the window (*m*).
The parameters $yorigmin$ and $yorigmax$ are respectively lower and higher than the area where heat transfer is studied in the particles. This is due to the fact that it is realistic to assume that the particles will not only be irradiated by rays coming from the section on the window located exacly at their height (Y-coordinate), but also by rays coming from lower and higher parts of the window.
$ydestmax$: maximum value of the Y-coordinate of the back wall towards where the solar rays will be directed.
$ydestmin$: minimum value of the Y-coordinate of the back wall towards where the solar rays will be directed.
These parameters can be schematically seen in Figure A.7.

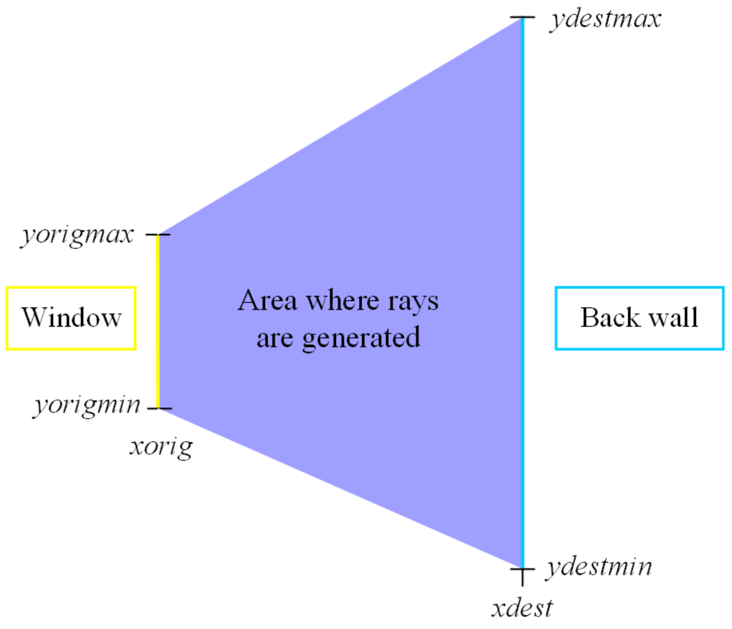


Figure A.7: Area where rays are generated.

$x1$: $xorig$.
$y1$: vector that contains all the Y-coordinates of the origin points of the solar rays.
n_rays: number of solar rays.
$x2$: $xdest$.
$y2$: vector that contains all the possible Y-coordinates of the destination points of the solar rays.
a: X-coordinate of the center of the studied particle.
b: Y-coordinate of the center of the studied particle.
r: Radius of the studied particle.
$prev\_ref\_in\_part$: Indicates if the ray's previous reflection was on a particle (1: Yes, 0: No).
n_wall_refl: Number of times the ray has been reflected on the back wall.

Mathematically speaking, when a line intersects with a circle, there are two intersection points. In reality, rays cannot go through the particles, which is why there is only one intersection point, where reflection happens (reflection point). The other point calculated has been called impossible point. This has been graphically exemplified in Figure A.8.

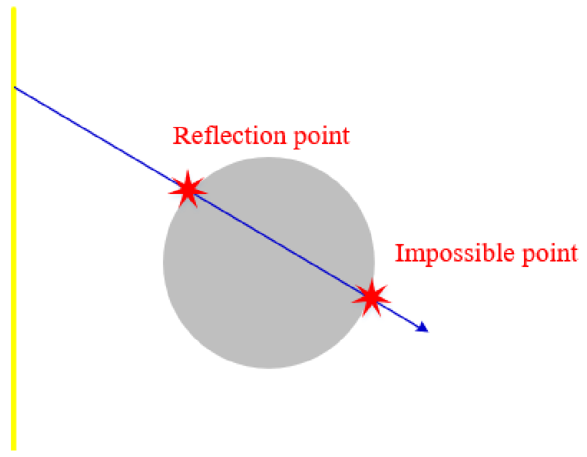


Figure A.8: Graphical example of the two intersection points between a ray and a particle.

The equation of a line is $y = mx + n$, being:
m: Intersection point of the line with the ordinate axis (value of Y when X=0).
n: Slope of the line.

$m1$: m for the studied ray.
$n1$: n for the studied ray.
$xreflection\_point$: X-coordinate of the reflection point.
$yreflection\_point$: Y-coordinate of the reflection point.
$ximpossible\_point$: X-coordinate of the impossible point.
$yimpossible\_point$: Y-coordinate of the impossible point.
C: Indicates if there has been an intersection (1:Yes, 0:No).
$shortest\_d$: Shortest distance in matrix DD.
$row\_previous$: Number of the particle that reflects the studied ray.
$xint\_true$: X-coordinate of the intersection point.
$yint$: Y-coordinate of the intersection point.
x_or: X-coordinate of the origin of the studied ray.
y_or: Y-coordinate of the origin of the studied ray.
$gamma$: Random angle in the range [0,pi] (rad).
$alpha$: Angle between the reflection point and the horizontal line of the first quadrant of the circle (rad). See Figure A.4.
$beta$: Angle necessary to get the right reflection direction for the ray (rad). $beta = alpha - 90°$.
k_length: Input value. Number high enough to make sure the destination point of the ray is out of the studied boundary.
xor_c0: X-coordinate of the origin of the ray.
yor_c0: X-coordinate of the origin of the ray.
$xdest\_c0$: X-coordinate of the initial destination of the ray.
$ydest\_c0$: Y-coordinate of the initial destination of the ray.
x_wall: X-coordinate of the back wall.
y_wall: Y-coordinate of the reflection of the ray on the back wall.

$gamma1$: Random angle in the range [0,pi] (rad).
$beta1$: Input value. Angle necessary to get the right reflection direction for the ray (rad).
x_r: X-coordinate of the final random point that the ray will be reflected towards.
y_r: Y-coordinate of the final random point that the ray will be reflected towards.
$m2$: m for reflected ray.
$n2$: n for the reflected ray.
$prev\_ref\_in\_part\_em$: Indicates if the emitted ray's previous reflection was on a particle (1:Yes,0:No).
$n\_wall\_refl\_em$: Number of times the emitted ray has been reflected on the back wall.
xor_c0_em: X-coordinate of the origin of the ray.
yor_c0_em: X-coordinate of the origin of the ray.
$xdest\_c0\_em$: X-coordinate of the initial destination of the ray.
$ydest\_c0\_em$: Y-coordinate of the initial destination of the ray.
$position$: Keeps the count of the number of reflections in a particle.
n_times: Number of times that the studied ray is reflected on the studied particle.
$node1$: First node of the studied boundary line.
$node2$: Second node of the studied boundary line.
x_node1: X-coordinate of node1.
y_node1: Y-coordinate of node1.
x_node2: X-coordinate of node2.
y_node2: Y-coordinate of node2.
x_orig: X-coordinate of the origin of the emitted ray.
y_orig: Y-coordinate of the origin of the emitted ray.
x_dest: X-coordinate of the initial destination of the emitted ray.
y_dest: Y-coordinate of the initial destinacion of the emitted ray.
$temp\_node1$: Temperature of the node1.
$temp\_node2$: Temperature of the node2.
$temp\_med\_line$: Average temperature between $temp\_node1$ and $temp\_node2$.
$reps$: Number of times that the studied ray is reflected on the studied particle .
$repetitions$: Number of rays that are reflected on the studied boundary line of the studied particle.
$total\_abs\_em$: Total energy is that is absorbed by the studied boundary line of the studied particle coming from other particles' emission (W).
ray_n: Ray that hits the studied boundary line of the particle.
$part\_n$: Number of the particle that emitted said ray.
n: Number that helps calculate the energy of said ray when it hits the studied boundary line.
abs_energy: Energy of that ray that is absorbed by the studied boundary line.
n_rays_em: Number of rays emitted by every particle.

Matrices

$points\_origin$: Coordinates of the solar rays' origins.

$$\begin{array}{c|cc} \textbf{Ray number} & \textbf{X-coordinate} & \textbf{Y-coordinate} \\ 1 & X_1 & Y_1 \\ 2 & X_2 & Y_2 \\ ... & ... & ... \\ ... & ... & ... \\ \text{n\_rays} & X_n & Y_n \end{array}$$

$points\_destination$: Coordinates of the solar ray's inital destinations (they will change if the ray is reflected).

$$\begin{array}{c|cc} \textbf{Ray number} & \textbf{X-coordinate} & \textbf{Y-coordinate} \\ 1 & X_1 & Y_1 \\ 2 & X_2 & Y_2 \\ ... & ... & ... \\ ... & ... & ... \\ \text{n\_rays} & X_n & Y_n \end{array}$$

$rays$: Stores the values m1 and n1 for every ray.

$$\begin{array}{c|cc} \textbf{Ray number} & \textbf{m} & \textbf{n} \\ 1 & m_1 & n_1 \\ 2 & m_2 & n_2 \\ ... & ... & ... \\ ... & ... & ... \\ \text{n\_rays} & m_n & n_n \end{array}$$

$true\_xint$: X-coordinate of every reflection point.

$$\begin{array}{c|cccc} \textbf{Reflection number} & \textbf{Ray 1 X-coordinate} & \textbf{Ray 2 X-coordinate} & \textbf{...} & \textbf{Ray n X-coordinate} \\ 1 & X_1 & X_1 & ... & X_1 \\ 2 & X_2 & X_2 & ... & X_2 \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ \text{n\_sections\_ray} & X_n & X_n & ... & X_n \end{array}$$

$true\_yint$: Y-coordinate of every reflection point.

$$\begin{array}{c|ccccc}
\textbf{Reflection number} & \textbf{Ray 1 Y-coordinate} & \textbf{Ray 2 Y-coordinate} & \textbf{...} & \textbf{Ray n Y-coordinate} \\
\hline
1 & Y_1 & Y_1 & ... & Y_1 \\
2 & Y_2 & Y_2 & ... & Y_2 \\
... & ... & ... & ... & ... \\
... & ... & ... & ... & ... \\
\text{n\_sections\_ray} & Y_n & Y_n & ... & Y_n
\end{array}$$

$ray\_intersections$: For every solar ray, it stores the number of the particle where it has been reflected, the X and Y coordinates of the reflection point, the section of the ray in that reflection, the total number of times that the ray has been reflected on the back wall in that moment and the number of times that the ray has been reflected on the particles on that moment (which is the difference between the two previous values).

$$\begin{array}{c|cccccc}
\textbf{Ray number} & \textbf{Particle number} & \textbf{X-coordinate of the reflection point} & \textbf{Y-coordinate of the reflection point} & \textbf{Section number of the ray} & \textbf{Number of reflections on the wall} & \textbf{Number of reflections on the particles} \\
\hline
1 & p.number & X_1 & Y_1 & 1 & refl.number & refl.number \\
2 & p.number & X_2 & Y_2 & 2 & refl.number & refl.number \\
... & p.number & ... & ... & ... & refl.number & refl.number \\
... & p.number & ... & ... & ... & refl.number & refl.number \\
\text{n\_rays} & p.number & X_n & Y_n & n\_sections\_ray & refl.number & refl.number
\end{array}$$

$xint$: Values of the X-coordinates of all the mathematical intersection points of the studied ray with each particle.

$$\begin{array}{c|cc}
\textbf{Particle number} & \textbf{X-coordinate 1} & \textbf{X-coordinate 1} \\
\hline
1 & X_1 & X_1 \\
2 & X_1 & X_1 \\
... & ... & ... \\
... & ... & ... \\
\text{n\_particles} & X_1 & X_1
\end{array}$$

XX: Stores the real values of the X-coordinate of the possible reflection points, and DV2 in the remaining points.

$$\begin{array}{c c}
\textbf{Particle number} & \begin{array}{cc} \textbf{X-coordinate 1 or DV2} & \textbf{X-coordinate 2 or DV2} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \end{pmatrix}
\end{array}$$

YY: Stores the real values of the Y-coordinate of the possible reflection points, and DV2 in the remaining points.

$$\begin{array}{c c}
\textbf{Particle number} & \begin{array}{cc} \textbf{Y-coordinate 1 or DV2} & \textbf{Y-coordinate 2 or DV2} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \end{pmatrix}
\end{array}$$

DD: Stores the distance between the origin of the ray and the possible reflection point and DV2 in the remaining points.

$$\begin{array}{c c}
\textbf{Particle number} & \begin{array}{cc} \textbf{Distance or DV2} & \textbf{Distance or DV2} \end{array} \\
\begin{array}{c} 1 \\ 2 \\ ... \\ ... \\ \text{n\_particles} \end{array} & \begin{pmatrix} ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \\ ... & ... \end{pmatrix}
\end{array}$$

$particle\_intersections$: Stores the number of the solar ray reflected by the particle at one moment, the X and Y coordinates of that reflection point, the number of the section of the solar ray that is reflected, the number of the boundary line of the particle where said reflection happens and the number of times that the solar ray has been reflected on the particles at that moment. Since it is not possible to know a priori how many solar rays are going to be reflected on the surface of a particle, a big default value *DV4* defined by the user is used to establish the size of the matrix. This way, the matrix $particle\_intersections$ will be able to store a total of *DV4* reflections of solar rays on the surface of every particle.

$$\begin{array}{c|cccccc} & \textbf{Ray number} & \textbf{X-coordinate of the reflection point} & \textbf{Y-coordinate of the reflection point} & \textbf{Section number of the ray} & \textbf{Affected boundary line number} & \textbf{Number of reflections of the ray on the particles} \\ \hline 1 & r.number & X_1 & Y_1 & s.number & b.l.number & refl.number \\ 2 & r.number & X_2 & Y_2 & s.number & b.l.number & refl.number \\ ... & r.number & ... & ... & s.number & b.l.number & refl.number \\ ... & r.number & ... & ... & s.number & b.l.number & refl.number \\ DV4 & r.number & X_n & Y_n & s.number & b.l.number & refl.number \end{array}$$

$distance\_to\_intersection$: Distance from every boundary node to the studied reflection point (m).

$$\begin{array}{c|c} \textbf{Boundary node number} & \textbf{Distance} \\ \hline 1 & ... \\ 2 & ... \\ ... & ... \\ ... & ... \\ \text{n\_bn} & ... \end{array}$$

$n\_intersections$: Stores the number of rays reflected on each particle.

$$\begin{array}{c|cccc} & \textbf{Number} & \textbf{of} & \textbf{the} & \textbf{particle} \\ & 1 & 2 & ... & \text{n\_particles} \\ \hline \textbf{Number of rays reflected} & ... & ... & ... & ... \end{array}$$

$nrays\_l\_z$: Stores the number of rays that are reflected on the same boundary line of a particle and belong to the same section of their corresponding rays.

$$\begin{array}{c|cccc} & \textbf{Number} & \textbf{of the} & \textbf{boundary} & \textbf{line} \\ & \text{b.l.1} & \text{b.l.2} & ... & \text{n\_lines} \\ \hline \textbf{Number of reflections} & ... & ... & ... & ... \\ 1 & ... & ... & ... & ... \\ 2 & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ \text{n\_reflections} & ... & ... & ... & ... \end{array}$$

$rays\_em$: Stores m and n for all the rays emitted by every particle.
$points\_origin\_em$: Coordinates of the emitted rays' origins.

$$\begin{array}{cc}
\begin{array}{c}\textbf{Ray}\\ \textbf{number}\end{array} & \begin{array}{cc}\textbf{X-coordinate} & \textbf{Y-coordinate}\end{array} \\
\begin{array}{c}1\\ 2\\ ...\\ ...\\ \text{n\_rays\_em}\end{array} & \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}
\end{array}$$

$points\_destination\_em$: Coordinates of the emitted rays' initial destinations (they will change if the ray is reflected).

$$\begin{array}{cc}
\begin{array}{c}\textbf{Ray}\\ \textbf{number}\end{array} & \begin{array}{cc}\textbf{X-coordinate} & \textbf{Y-coordinate}\end{array} \\
\begin{array}{c}1\\ 2\\ ...\\ ...\\ \text{n\_rays\_em}\end{array} & \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ ... & ... \\ ... & ... \\ X_n & Y_n \end{pmatrix}
\end{array}$$

$ray\_intersections\_em$: Every particle emits one rays from every one of its boundary lines. Since, in the chosen grid, every particles has 52 boundary lines (n_bars), every particle emits 52 rays. For every ray emitted by every particle, $ray\_intersections\_em$ stores: the number of the particles where it has been reflected, the X and Y coordinates of said reflection points, the section of the ray in that reflection, the total number of times that the ray has been reflected on the back wall in that moment and the number of times that the ray has been reflected on the particles on that moment (which is difference between the two previous values).

$$\begin{array}{cc}
\begin{array}{c}\textbf{Ray}\\ \textbf{number}\end{array} & \begin{array}{ccccccc}\begin{array}{c}\textbf{Particle}\\ \textbf{number}\end{array} & \begin{array}{c}\textbf{X-coordinate}\\ \textbf{of the}\\ \textbf{reflection}\\ \textbf{point}\end{array} & \begin{array}{c}\textbf{Y-coordinate}\\ \textbf{of the}\\ \textbf{reflection}\\ \textbf{point}\end{array} & \begin{array}{c}\textbf{Section}\\ \textbf{number}\\ \textbf{of the ray}\end{array} & \begin{array}{c}\textbf{Number of}\\ \textbf{reflections on}\\ \textbf{the wall}\end{array} & \begin{array}{c}\textbf{Number of}\\ \textbf{reflections on}\\ \textbf{the particles}\end{array}\end{array} \\
\begin{array}{c}1\\ 2\\ ...\\ ...\\ \text{n\_bars}\end{array} & \begin{pmatrix} p.number & X_1 & Y_1 & 1 & refl.number & refl.number \\ p.number & X_2 & Y_2 & 2 & refl.number & refl.number \\ p.number & ... & ... & \ldots & refl.number & refl.number \\ p.number & ... & ... & \ldots & refl.number & refl.number \\ p.number & X_n & Y_n & n\_sections\_ray & refl.number & refl.number \end{pmatrix}
\end{array}$$

$true\_xint\_em$: X-coordinates of every reflection point.

$$\begin{array}{c|ccccc|}
\textbf{Reflection number} & \textbf{Ray 1 X-coordinate} & \textbf{Ray 2 X-coordinate} & \textbf{...} & \textbf{Ray n\_rays\_em X-coordinate} \\
1 & X_1 & X_1 & ... & X_1 \\
2 & X_2 & X_2 & ... & X_2 \\
... & ... & ... & ... & ... \\
... & ... & ... & ... & ... \\
\text{n\_sections\_ray} & X_n & X_n & ... & X_n
\end{array}$$

$true\_yint\_em$: Y-coordinates of every reflection point.

$$\begin{array}{c|ccccc|}
\textbf{Reflection number} & \textbf{Ray 1 Y-coordinate} & \textbf{Ray 2 Y-coordinate} & \textbf{...} & \textbf{Ray n\_rays\_em Y-coordinate} \\
1 & Y_1 & Y_1 & ... & Y_1 \\
2 & Y_2 & Y_2 & ... & Y_2 \\
... & ... & ... & ... & ... \\
... & ... & ... & ... & ... \\
\text{n\_sections\_ray} & Y_n & Y_n & ... & Y_n
\end{array}$$

$emission\_radiation$: Energy emitted by every boundary line of every particle (W).

$$\begin{array}{c|ccccc|}
\textbf{Boundary line} & \textbf{Particle 1} & \textbf{Particle 2} & \textbf{...} & \textbf{Particle n\_particles} \\
1 & emittedenergy & emittedenergy & ... & emittedenergy \\
2 & emittedenergy & emittedenergy & ... & emittedenergy \\
... & ... & ... & ... & ... \\
... & ... & ... & ... & ... \\
\text{n\_bars} & emittedenergy & emittedenergy & ... & emittedenergy
\end{array}$$

$particle\_intersections\_em$: For every particle, it stores: the number of the rays emitted by other particles that have been reflected on the studied particle, the number of the particles that emitted each one of said rays, the section of those rays at the moment they were reflected on the studied particle, the initial energy (W) that those rays had when they were emitted, the boundary lines of the studied particle where they are reflected, and the number of reflections on the particles. Since it is not possible to know a priori how many rays emitted by other particles are going to be reflected on the surface of a particle, a big default value *DV3* defined by the user is used to establish the size of the matrix. This way, the matrix $particle\_intersections\_em$ will be able to store a total of *DV3* reflections of rays emited by other particles on the surface of every particle.

$$\begin{array}{c c c c c c c}
 & \textbf{Ray number} & \textbf{Emitting particle number} & \textbf{Section number of the ray} & \textbf{Initial energy of the ray} & \textbf{Affected boundary line number} & \textbf{Number of reflections on the particles} \\
1 & r.number & p.number & s.number & energy & b.l.number & refl.number \\
2 & r.number & p.number & s.number & energy & b.l.number & refl.number \\
... & r.number & p.number & s.number & energy & b.l.number & refl.number \\
... & r.number & p.number & s.number & energy & b.l.number & refl.number \\
\text{DV3} & r.number & p.number & s.number & energy & b.l.number & refl.number
\end{array}$$

$indices$: Positions (rows) where the studied ray is reflected on the studied particle.
$emission\_radiation\_incident$: Energy absorbed by every boundary line of every particle coming from other particles' emission (W).

$$\begin{array}{c c c c c}
\textbf{Boundary line} & \textbf{Particle 1} & \textbf{Particle 2} & \textbf{...} & \textbf{Particle n\_particles} \\
1 & absorbedenergy & absorbedenergy & ... & absorbedenergy \\
2 & absorbedenergy & absorbedenergy & ... & absorbedenergy \\
... & ... & ... & ... & ... \\
... & ... & ... & ... & ... \\
\text{n\_bars} & absorbedenergy & absorbedenergy & ... & absorbedenergy
\end{array}$$

Part 5.2: Heat Transfer resolution with the FEM

Parameters

$scheme$: Input value. Number that stablishes the resolution squeme used (1: Implicit, 0: Explicit, 0.5: Crank Nicholson).
n_ray_sets: Input value. Number of times that a new set of incoming solar rays is simulated.
dt: Input value. Time step value.
$nsteps$: Input value. Number of time steps.
$node\_number$: Boundary node closest to the reflection point.
$line\_change\_BC$: Boundary line these nodes belong to in this particle.
$ndle$: Input value. Number of nodes per triangular element.
$inel$: Input value. Number of nodes of the studied element (the default value is 2, which is the number of nodes for the line elements).
$itype$: Type of element (1:Boundary line with Neumann BC, 2:Boundary line with Cauchy BC, 3:Triangular element).
$iclass$: Number that specifies the boundary condition that will be applied (1 or 2).
$vres$: Residue for time control.
vdu: Incremental solution.
$nres$: Euclidean norm of vector vres.
$time$: Time of the studied iteration (s).

Matrices

n_BC_matrix: Numbers assigned to every node in every particle that will be used to implement in them their corresponding boundary conditions. For the nodes in the bounday, 11 means that no ray has hit its surrounding boundary line, and 12 means and one or more rays have. For the nodes in the interior of the particle, this number will always be 31, and is used to obtain the material properties stored in other part of the program. The election of these numbers is due to the fact that the original code for the FEM resolution for heat transfer belongs to Emmanuel Lefrançois from the Université de Technologie de Compiegne, and he used them to assign different boundary conditions and material properties to the solids of study. To lean more information about the reasons why, the user should look for his original code.

$$\begin{array}{c c}
\textbf{Node number} & \begin{array}{cccc}\textbf{Particle 1} & \textbf{Particle 2} & \textbf{...} & \textbf{Particle n\_particles}\end{array} \\
\begin{array}{c}1\\2\\...\\...\\\text{nnt}\end{array} & \begin{pmatrix} assignednumber & assignednumber & ... & assignednumber \\ assignednumber & assignednumber & ... & assignednumber \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ assignednumber & assignednumber & ... & assignednumber \end{pmatrix}
\end{array}$$

$vkg\_particles$: Global rigidity (conductivity) matrix of every particle. Its size is $nntxnntxn\_particles$.
$vmg\_particles$: Global mass matrix of every particle. Its size is $nntxnntxn\_particles$.
$vfg\_particles$: Global sollicitation vector of every particle. Its size is $nntx1xn\_particles$.
$incident\_sun$: Total solar energy that incides in every boundary line of every particle (W).

$$\begin{array}{c c}
\textbf{Boundary line} & \begin{array}{cccc}\textbf{Particle 1} & \textbf{Particle 2} & \textbf{...} & \textbf{Particle n\_particles}\end{array} \\
\begin{array}{c}1\\2\\...\\...\\\text{n\_bars}\end{array} & \begin{pmatrix} incidentenergy & incidentenergy & ... & incidentenergy \\ incidentenergy & incidentenergy & ... & incidentenergy \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ incidentenergy & incidentenergy & ... & incidentenergy \end{pmatrix}
\end{array}$$

vkg: Global rigidity (conductivity) matrix of the studied particle. Its size is $nntxnnt$.
vmg: Global mass matrix of the studied particle. Its size is $nntxnnt$.
vfg: Global sollicitation vector of the studied particle. Its size is $nntx1$.
$kloce$: Localization table for the degrees of freedom of the studied element.
$vcore$: X and Y coordinates of the studied element.
vkt: Tangent matrix. Its size is $nntxnntxn\_particles$.

$vsol\_stored$: Stores the temperature of the nodes of every particle every 10 timesteps
.

APPENDIX

B - Main script of the MATLAB code

B - Main script of the MATLAB code

```
clc
clear all
close all

% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

% This code simulates the transient heating process of the particles inside
% a Falling Particle Receiver. A Monte Carlo Ray Tracing Method is used to
% model radiation to and from a polydisperse particle phase, and a Finite
% Element Method is used for the simulation of conduction heat transfer
% inside the particles.

% The code has been divided in five parts:
% Part 1: Generation of randomly located particles with different radii
% Part 2: Relocation of the generated particles to the axes of interest
% Part 3: Generation of a mesh inside the particles
% Part 4: Initial Solution
% Part 5: Heat transfer in the particles
%   Part 5.1: Generation of random rays and their reflections
%   Part 5.2: Heat Transfer resolution with the FEM


%==========================================================================
% THE USER MUST DEFINE THE FOLLOWING PARAMETERS:
%==========================================================================

%==========================================================================
% GENERAL PARAMETERS
%==========================================================================

solar_flux     =  600000; % W/m^2
n_sections_ray = 1;        % Maximum number of sections allowed for each ray
s_b            = 5.670373*10e 8; % Stefan Boltzmann constant [W/m^2 K ^4]

% Material properties
e              = 0.843;   % Emissivity
absp           = 0.934;   % Absorptivity
k              = 2;       % Conductivity [W/ m K]
dens           = 3550;    % Density [kg/m^3]
cp             = 760;     % Specific heat [J/kg K]

%==========================================================================
% INPUT PARAMETERS FOR
% PART 1    GENERATION OF RANDOMLY LOCATED PARTICLES WITH DIFFERENT RADII
%==========================================================================

% Choice of the target planar void ratio on the region of the particles
target_planar_void_ratio = 99;

% Choice of the maximum and minimum radius desires in the particles
% generated in mm
max_radius = 0.35;   % mm
min_radius = 0.5e 3; % mm
%==========================================================================
% INPUT PARAMETERS FOR
% PART 4    INITIAL SOLUTION
%==========================================================================

% From all the particles initially generated in the area X:[0 0.030]
% Y:[0 0.050], the user can chose to model the heat transfer process only
% in the particles located in a smaller region of this area.

% This is recommended because solving the problem for all the particles
% would require too much computational time.

% Choice of the smaller region where particles will be studied, inside the
% area X:[0 0.030] Y:[0 0.050]
xmin_area     = 0;
xmax_area     = 0.005;
ymin_area     = 0;
ymax_area     = 0.005;

% Region that will be plotted. It is recommended to choose an area slightly
% bigger than the one defined above.
xaxismin_area =  0.0030;
xaxismax_area =   0.008;
yaxismin_area =  0.0005;
yaxismax_area =   0.0055;

initial_temperature     = 20; %Initial temperature of the particles [ C ]

distance_window_curtain = 0; % This number determines where the position of
                             % the window in both Figures 1 and 2
%==========================================================================
% INPUT PARAMETERS FOR
% PART 5    HEAT TRANSFER IN THE PARTICLES
%==========================================================================

% PART 5.1    GENERATION OF RANDOM RAYS AND THEIR REFLECTIONS

% Section of the window where solar rays are simulated
% (Parameters to define the origin and destination of the rays)

xorig         =   0 distance_window_curtain;
xdest         =   0.0075; % This number determines the position of the back
yorigmin      =  0.010;  % wall in Figure 2
yorigmax      =   0.015;
dy            =  0.00005;  % Distance between rays in the window
ydestmax      =   0.035;
```

```
ydestmin        =  0.030;

% PART 5.2    HEAT TRANSFER RESOLUTION WITH THE FEM

scheme          = 1;        % Resolution squeme(1:Implicit,0:Explicit,
n_ray_sets      = 3;        % 0.5:Crank Nicholson)
dt              = 0.027;
nsteps          = 10;

%==========================================================================
% END OF INPUT PARAMETERS
%==========================================================================

%%

%==========================================================================
% PART 1    GENERATION OF RANDOMLY LOCATED PARTICLES WITH DIFFERENT RADII
%==========================================================================

% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

% This PART 1 is mainly based on the original code by:
% The MIT License (MIT)
% Copyright (c) 2016
% Andrea Chiarelli, Andrew Dawson, Alvaro Garcia
% The University of Nottingham

% Permission is hereby granted, free of charge, to any person obtaining a
% copy of this software and associated documentation files (the "Software"),
% to deal in the Software without restriction, including without limitation
% the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the
% Software is furnished to do so, subject to the following conditions:
% The above copyright notice and this permission notice shall be included
% in all copies or substantial portions of the Software.

% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
% OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
% MERCHANTABILITY,FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
% IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
% CLAIM, DAMAGES OR OTHERLIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
% OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
% THE USE OR OTHER DEALINGS IN THE SOFTWARE.

%

% THE USER MUST NOT CHANGE THE FOLLOWING PARAMETERS:

% Region where the particles are initially generated    Figure 1
xorigmin_part = 0;
xorigmax_part = 0.030;
```

```
yorigmin_part = xorigmin_part;
yorigmax_part = 0.050;
% xorigmin_part and yorigmin_part must be equal

% Region plotted initially   Figure 1
xaxismin      =  0.0025;
xaxismax      = 0.030;  % This number also determines the position of the
yaxismin      = 0;      % back wall in Figure 1
yaxismax      = 0.050;

desired_width  = abs(xorigmax_part  xorigmin_part);
desired_height = abs(yorigmax_part  yorigmin_part);
factor         = desired_height/desired_width;

origin_of_cartesian_axes = 0;
rectangle_width          = 3;
rectangle_height         = factor*rectangle_width;

xmin_generation = origin_of_cartesian_axes;
xmax_generation = xmin_generation + rectangle_width;
xmed_generation = (xmin_generation + xmax_generation)/2;

ymin_generation = origin_of_cartesian_axes;
ymax_generation = ymin_generation + rectangle_height;
ymed_generation = (ymin_generation + ymax_generation)/2;

%                    INPUT VARIABLES INITIALIZATION (START)                %
  number_of_particles          = 2;
  new_particles_per_generation = 2;
  maximum_radius               = max_radius/10;%This parameter is in meters
  minimum_radius_seeded        = min_radius/10;%This parameter is in meters
%                    INPUT VARIABLES INITIALIZATION  (END)                 %

  % Generation of a vector of radii, one row for each particle
  radius = minimum_radius_seeded*ones(number_of_particles,1);
  % Generation of the centres with random positions, one row for  particle
  centre(:,1) = minimum_radius_seeded+(rectangle_width 2*                ...
                         minimum_radius_seeded)*rand(number_of_particles,1);
  centre(:,2) = minimum_radius_seeded+(rectangle_height 2*               ...
                         minimum_radius_seeded)*rand(number_of_particles,1);
  % Generation of the angle for the equation for parametric circumferences
  angles_for_parametric_circle = 0:0.01:2*pi;
  % Initialisation of the first generation of particles
  x_coordinates_circles = zeros(length(angles_for_parametric_circle),   ...
                                                    number_of_particles);
  y_coordinates_circles = x_coordinates_circles;
  % Counter for later use
  counter = number_of_particles;
  % Beginning of the packing process
  iteration = 2;
  planar_void_ratio(2) = 100; % This needs to start at 2 because in the
      %loops below it compares the current void ratio to its previous value

  while planar_void_ratio(iteration)~=planar_void_ratio(iteration 1) && ...
```

```
                      planar_void_ratio(iteration)>target_planar_void_ratio
    iteration = iteration+1;
    for k=1:number_of_particles
      % Condition 1: Is the distance between particle k and all other
      % particles smaller than the sum of their radii plus the minimum
      % radius growth? This must return 1, because in order to grow this
      % condition must be 0 when particle k is compared to all other
      % particles but when it will be compared to itself the result will
      % be 1.
      condition(1) = sum(sqrt((centre(:,1) centre(k,1)).^2+(centre(:,2) ...
                centre(k,2)).^2)<(radius(k)+minimum_radius_seeded+radius));
      % Conditions 2 5: If the radius grows by 'minimum_radius_seeded',
      % will it still be inside the rectangle defined above?
      condition(2) = (centre(k,1)+radius(k)+minimum_radius_seeded<      ...
                                                        rectangle_width);
      condition(3) = (centre(k,1) radius(k) minimum_radius_seeded>      ...
                                               origin_of_cartesian_axes);
      condition(4) = (centre(k,2)+radius(k)+minimum_radius_seeded<      ...
                                                       rectangle_height);
      condition(5) = (centre(k,2) radius(k) minimum_radius_seeded>      ...
                                               origin_of_cartesian_axes);
      % Condition 6: If the radius grows by 'minimum_radius_seeded', will
      % it still be smaller than the maximum radius allowed?
      condition(6) = (radius(k)+minimum_radius_seeded<=maximum_radius);
      % Logical condition: if all the conditions above are true (=1), the
      % particle k is allowed to grow
      can_it_grow = condition(1)==1 && condition(2)==1 &&               ...
                                  condition(3)==1 && condition(4)==1   ...
                                  && condition(5)==1 && condition(6)==1 ;
      % The radius of particle k is increased by 'minimum_radius_seeded'
      if can_it_grow==1
        radius(k)=radius(k)+minimum_radius_seeded;
        x_coordinates_circles(:,k)=radius(k).*                          ...
                           cos(angles_for_parametric_circle)+centre(k,1);
        y_coordinates_circles(:,k)=radius(k).*                          ...
                           sin(angles_for_parametric_circle)+centre(k,2);
      end
    end
    % The planar void ratio is calculated for the current iteration
    squared_sum_of_radii = sum(radius(1:number_of_particles).^2);
    planar_void_ratio(iteration) = (rectangle_width*rectangle_height    ...
          pi*squared_sum_of_radii)/(rectangle_width*rectangle_height)*100;
    % If the planar void ratio is lower than the target the calculation
    % must stop
    if planar_void_ratio(iteration)<target_planar_void_ratio
      quit_the_calculation = 1; % If this is equal to 1, the algorithm is
      break                     % stopped
    end
    quit_the_calculation = 0;
  end
  % From now on, new generation of particles are created and added.
  % When the planar void ratio drops below certain values, it is
  % best to seed more particles per generation than the default value.
  % This is done to have a better chance that they will fall in empty space.
```

```
 increase_particles_per_generation = 0;
% In the 'if' conditions below this is increased to 1 and then 2 and 3
% just so that the increase in 'new_particles_per_generation' happens
% only once at each planar void ration mentioned
 while planar_void_ratio(iteration)>target_planar_void_ratio &&          ...
                                                  quit_the_calculation==0
   i_new_generation = counter;
   if planar_void_ratio(iteration)<35 &&                                 ...
                                     increase_particles_per_generation==0
     new_particles_per_generation=new_particles_per_generation+1000;
     increase_particles_per_generation = 1;
   end
   if planar_void_ratio(iteration)<25 &&                                 ...
                                     increase_particles_per_generation==1
     new_particles_per_generation=new_particles_per_generation+5000;
     increase_particles_per_generation = 2;
   end
   if planar_void_ratio(iteration)<15 &&                                 ...
                                     increase_particles_per_generation==2
     new_particles_per_generation=new_particles_per_generation+10000;
     increase_particles_per_generation = 3;
   end
  % The next condition avoids that no new generation of particles is empty
   while counter == i_new_generation
     for q=i_new_generation+1:                                           ...
                          i_new_generation+new_particles_per_generation
       % A new particle is initialised
       centre(q,1) = minimum_radius_seeded+(rectangle_width              ...
                                          2*minimum_radius_seeded)*rand;
       centre(q,2) = minimum_radius_seeded+(rectangle_height             ...
                                          2*minimum_radius_seeded)*rand;
       radius(q)   = minimum_radius_seeded;
       % Is the particle acceptable? If yes, it is added to the vectors of
       % centres and radii
       if sum(sqrt((centre(:,1) centre(q,1)).^2+(centre(:,2)             ...
             centre(q,2)).^2)<(radius(q)+minimum_radius_seeded+radius))==1
         counter = counter+1;
         centre(counter,:) = centre(q,:);
         radius(counter)   = radius(q);
       end
     end
   end

   % The vectors of centres and radii are cleaned from the unused values
   centre(counter+1:end,:) = [];
   radius(counter+1:end)   = [];
   iteration = iteration+1;
   % This is re initialised due to the counters used below
   planar_void_ratio(iteration) = 100;

   while planar_void_ratio(iteration)~=planar_void_ratio(iteration 1)   ...
                                              && quit_the_calculation==0
     iteration = iteration+1;
     for k = i_new_generation+1:counter
```

```
        % For the meaning of these conditions see above.
        condition(1) = sum(sqrt((centre(:,1) centre(k,1)).^2+            ...
                              (centre(:,2) centre(k,2)).^2)<(radius(k)+  ...
                                            minimum_radius_seeded+radius));
        condition(2) = (centre(k,1)+radius(k)+minimum_radius_seeded<     ...
                                                          rectangle_width);
        condition(3) = (centre(k,1) radius(k) minimum_radius_seeded>     ...
                                                   origin_of_cartesian_axes);
        condition(4) = (centre(k,2)+radius(k)+minimum_radius_seeded<     ...
                                                          rectangle_height);
        condition(5) = (centre(k,2) radius(k) minimum_radius_seeded>     ...
                                                   origin_of_cartesian_axes);
        condition(6) = (radius(k)+minimum_radius_seeded<=maximum_radius);

        can_it_grow  = condition(1)==1 && condition(2)==1 &&             ...
                                     condition(3)==1 && condition(4)==1  ...
                                    && condition(5)==1 && condition(6)==1;
        if can_it_grow == 1
          radius(k) = radius(k)+minimum_radius_seeded;
        end
      end
      % The planar void ratio is calculated for the current iteration
      squared_sum_of_radii = sum(radius(1:counter).^2);
      planar_void_ratio(iteration) = (rectangle_width*rectangle_height   ...
           pi*squared_sum_of_radii)/(rectangle_width*rectangle_height)*100;

      % If the planar void ratio is lower than the target the calculation
      % must stop
      if planar_void_ratio(iteration)<target_planar_void_ratio
        quit_the_calculation = 1; % If this is equal to 1, the algorithm is
        break                     % stopped
      end
    end
    planar_void_ratio(end);
  end
  % The planar void ratio vector is saved as a simple number
  planar_void_ratio = planar_void_ratio(end);
  clear  x_coordinates_circles y_coordinates_circles planar_void_ratio

n_particles = counter;
n_particles_initial = n_particles;
hold on

%%

%=========================================================================
% PART 2   RELOCATION OF THE GENERATED PARTICLES TO THE AXES OF INTEREST
%=========================================================================
% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

scale          = 0.01; % 1 m becomes 1 cm, 1 cm becomes 0.1 mm
```

```
xmax             = xorigmax_part;
xmin             = xorigmin_part;
ymax             = yorigmax_part;
ymin             = yorigmin_part;
xmed             = (xmax+xmin)/2;
ymed             = (ymax+ymin)/2;
centre_initial = centre;
radius_initial = radius;
radius           = radius_initial*scale;

% Scalation of the centres to the axis of interest
for i = 1:n_particles_initial
    if     xmin_generation<=centre_initial(i,1) && centre_initial(i,1)    ...
                                                  <=xmed_generation && ...
           ymax_generation>=centre_initial(i,2) && centre_initial(i,2)    ...
                                                         >ymed_generation
   % If the particle is in the quadrant 1:
      centre(i,1) = (((xmed xmin)/xmed_generation)*centre_initial(i,1))+xmin;
      centre(i,2) = (((ymax ymed)/ymed_generation)*(centre_initial(i,2)   ...
                                                    ymed_generation))+ymed;

    elseif xmin_generation<=centre_initial(i,1) && centre_initial(i,1)    ...
                                                  <=xmed_generation && ...
           ymed_generation>=centre_initial(i,2) && centre_initial(i,2)    ...
                                                        >=ymin_generation
   % If the particle is in the quadrant 2:
      centre(i,1) = (((xmed xmin)/xmed_generation)*centre_initial(i,1))+xmin;
      centre(i,2) = (((ymed ymin)/ymed_generation)*centre_initial(i,2))+ymin;

    elseif xmax_generation>=centre_initial(i,1) && centre_initial(i,1)    ...
                                                   >xmed_generation && ...
           ymed_generation>=centre_initial(i,2) && centre_initial(i,2)    ...
                                                        >=ymin_generation
   % If the particle is in the quadrant 3:
      centre(i,1) = (((xmax xmed)/xmed_generation)*(centre_initial(i,1)   ...
                                                    xmed_generation))+xmed;
      centre(i,2) = (((ymed ymin)/ymed_generation)*centre_initial(i,2))+ymin;

    elseif xmax_generation>=centre_initial(i,1) && centre_initial(i,1)    ...
                                                   >xmed_generation && ...
           ymax_generation>=centre_initial(i,2) && centre_initial(i,2)    ...
                                                         >ymed_generation
   % If the particle is in the quadrant 4:
      centre(i,1) = (((xmax xmed)/xmed_generation)*(centre_initial(i,1)   ...
                                                    xmed_generation))+xmed;
      centre(i,2) = (((ymax ymed)/ymed_generation)*(centre_initial(i,2)   ...
                                                    ymed_generation))+ymed;
   end
end

%%

%==========================================================================
% PART 3    GENERATION OF A MESH INSIDE THE PARTICLES
```

```
%=========================================================================
% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

%                      Generic grid of radius R = 1 mm                   %
% Data reading
datageneral_Gmsh

vcorg_stored   = zeros(nnt,2,n_particles_initial);
vcorg_original = vcorg;

% Relocation of the nodes considering the centers of the particles. The
% real size of the particles is not taken into account yet
for i=1:n_particles_initial
 vcorg = [vcorg_original(:,1)+centre(i,1) vcorg_original(:,2)+centre(i,2)];
 vcorg_stored(:,:,i)  = vcorg;
end

% Scaling of the grid in each particle considering its real radius:
% relocation of the nodes to their final positions
r_1           = zeros(nnt,2);
r_2           = zeros(nnt,2);
vcorg_scaled  = zeros(nnt,2,n_particles_initial);

for i=1:n_particles_initial
  r_1(:,1)     =  vcorg_stored(:,1,i)   centre(i,1);
  r_1(:,2)     =  vcorg_stored(:,2,i)   centre(i,2);

  r_2          = r_1*(radius(i)/0.001);

  vcorg_scaled(:,1,i) = r_2(:,1) + centre(i,1);
  vcorg_scaled(:,2,i) = r_2(:,2) + centre(i,2);
end

% Plotting of the particles
for i=1:n_particles_initial
  meshplot(vcorg_scaled(:,:,i),kconec)
  hold on
end
axis ([ xaxismin xaxismax yaxismin yaxismax ])

% Information about the mesh
boundary_nodes         = kconec(find(kconec(:,3)==0),1:2);
boundary_nodes_vector = unique(boundary_nodes);
vcorg_scaled_boundary = zeros(max(size(boundary_nodes_vector)),2,       ...
                                                    n_particles_initial);
%Stores the coordinates of the nodes in the boundary
for p = 1:n_particles_initial
  vcorg_scaled_boundary(:,:,p) = vcorg_scaled(boundary_nodes_vector,:,p);
end
n_bars                    = numel(find(kconec(:,3)==0));
n_bn                      = max(size(boundary_nodes_vector));
```

```
arc_length           = zeros(n_particles_initial ,1);
for p=1:n_particles_initial
  arc_length(p,1)=(2*pi*radius(p))/n_bars;
end

%%

%========================================================================
% PART 4    INITIAL SOLUTION
%========================================================================
% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

vsol = initial_temperature.*ones(ndlt,n_particles_initial);

% Plotting of the initial solution
% Initial solution for all the particles generated
figure(1)
for p=1:n_particles_initial
  caxis([20 1200])
  femplot(vcorg_scaled(:,:,p),kconec,vsol(:,p),1)
  shading interp;
  title_plot = ['Time: 0 s'];
  title(title_plot)
  colorbar

  % This yellow line represents the window
  rectangle(...
  'Position',[0 distance_window_curtain,yaxismin,0.0001,...
                                              abs(yaxismax  yaxismin)], ...
  'EdgeColor', 'y',...
  'LineWidth', 1.5,...
  'LineStyle',' ')

  % This blue line represents the back wall
  rectangle(...
  'Position',[xaxismax,yaxismin,0.0001,abs(yaxismax  yaxismin)],...
  'EdgeColor', 'b',...
  'LineWidth', 1.5,...
  'LineStyle',' ')

  % This red rectangle represents the area where heat transfer is modeled
  rectangle(...
  'Position',[xmin_area,ymin_area,abs(xmax_area  xmin_area),          ...
                                             abs(ymax_area  ymin_area)],...
  'EdgeColor', 'r',...
  'LineWidth', 1.5,...
  'LineStyle',' ')

  set(gca,'fontsize',16);
  set(gca, 'FontName', 'Times New Roman');
  axis ([ xaxismin xaxismax yaxismin yaxismax ])
  hold on
  title(colorbar,' C')
  xlabel('Depth (m)')
  ylabel('Window length (m)')
end
time_previous_iteration = 0;

%           Selection of the particles inside the area of study          %

xmin_area_vector = xmin_area*ones(n_particles_initial,1);
xmax_area_vector = xmax_area*ones(n_particles_initial,1);
ymin_area_vector = ymin_area*ones(n_particles_initial,1);
ymax_area_vector = ymax_area*ones(n_particles_initial,1);

area_study_1     = [xmin_area_vector<=centre(:,1) ymin_area_vector<=    ...
                                                            centre(:,2)];
area_study_2     = [centre(:,1)<=xmax_area_vector centre(:,2)<=         ...
                                                        ymax_area_vector];
area_study_3     = [area_study_1(:,1)+area_study_2(:,1)                 ...
                              area_study_1(:,2)+area_study_2(:,2)];
area_study_4     = [area_study_3(:,1)+area_study_3(:,2)];

particles_area_study = find(area_study_4==4);

centre           = centre(particles_area_study,:);
n_particles      = size(centre,1);
radius           = radius(particles_area_study,:);
average_radius   = sum(radius)/n_particles;

vcorg_scaled     = vcorg_scaled(:,:,particles_area_study);
vsol             = initial_temperature.*ones(ndlt,n_particles);

% Initial solution only for the particles located in the area of study
figure(2)
for p=1:n_particles
  caxis([20 1200])
  femplot(vcorg_scaled(:,:,p),kconec,vsol(:,p),1)
  shading interp;
  title_plot = ['Time: 0 s'];
  title(title_plot)
  colorbar

  % This yellow line represents the window
  rectangle(...
  'Position',[xorig,yorigmin,0.000001,abs(yorigmax  yorigmin)],...
  'EdgeColor', 'y',...
  'LineWidth', 1.5,...
  'LineStyle',' ')

 % This blue line represents the back wall
  rectangle(...
  'Position',[xdest,ydestmin,0.000001,abs(ydestmax  ydestmin)],...
  'EdgeColor', 'b',...
  'LineWidth', 1.5,...
  'LineStyle',' ')

  set(gca,'fontsize',16);
  set(gca, 'FontName', 'Times New Roman');
  hold on
  title(colorbar,' C')
  xlabel('Depth (m)')
  ylabel('Window length (m)')
  axis equal
  axis ([ xaxismin_area xaxismax_area yaxismin_area yaxismax_area ])
end
time_previous_iteration = 0;
count_fig = 2;
%%

%=================================================================
% PART 5   HEAT  TRANSFER IN THE PARTICLES
%=================================================================
% Code by:
% Copyright (c) 2017
% Marta Mu oz M nguez
% Volunteer Staff at SDSU, student at University of Valladolid

% The FEM resolution code was taken from
% E. Lefrancois  UTC/2014

% GENERATION OF THE RAYS

% Rays' coordinates
% Origin points of solar rays
x1        = xorig;
y1        = yorigmax:dy:yorigmin;
n_rays    = length(y1);

% Destination points of solar rays
x2        = xdest;
y2        = ydestmax:dy:ydestmin;

%                INPUT VARIABLES INITIALIZATION (START)             %
% Total energy emitted from the studied section of the window
flux_window    = solar_flux*abs(yorigmax yorigmin); % W
flux      = flux_window/n_rays;
n_rays_em = n_bn;

DV1       = 300;
DV2       = 500;
DV3       = 500;
DV4       = 100;
%                INPUT VARIABLES INITIALIZATION (END)               %
average_temp        = zeros(n_particles,n_ray_sets+1);
time_axis           = zeros(1,n_ray_sets+1);
average_temp(:,1)   = initial_temperature;
time_axis(1,1)      = 0;

```

```

for set_rays=1:n_ray_sets
%==========================================================================
% PART 5.1    GENERATION OF RANDOM RAYS AND THEIR REFLECTIONS
%==========================================================================

%                         Solar rays generation (start)                       %
  points_origin = zeros(n_rays,2);

  for i=1:n_rays
    points_origin(i,1) = x1;
    points_origin(i,2) = y1(i);
  end

  points_destination = zeros(n_rays,2);

  for i=1:n_rays
    points_destination(i,1) = x2;
    points_destination(i,2) = ydestmax (((ydestmax (ydestmin))/(1 0))*  ...
                                                                   (1 rand));
  end

  % Reflection of the rays in the particles
  rays      = ones(n_sections_ray,2,n_rays)*DV1;
  true_xint = ones(n_sections_ray,n_rays)*DV1;
  true_yint = ones(n_sections_ray,n_rays)*DV1;

  ray_intersections = zeros(n_sections_ray,6,n_rays);

  for i=1:n_rays            % Runs ray by ray
    prev_refl_in_part = 0;
    n_wall_refl = 0;
    for z=1:n_sections_ray % Runs all the reflections of one ray
      xint = ones(n_particles,2)*DV2;
      if z==1 %Initial section of the ray
        % Deduction of the line ecuation
        m1 = (points_destination(i,2) points_origin(i,2))/              ...
                               (points_destination(i,1) points_origin(i,1));
        n1 = (points_destination(i,1)*points_origin(i,2)                ...
                             points_origin(i,1)*points_destination(i,2))/...
                               (points_destination(i,1) points_origin(i,1));
        rays(z,1,i) = m1;
        rays(z,2,i) = n1;
      else
        m1 = rays(z,1,i); % Already stored, in the previous iteration of z
        n1 = rays(z,2,i);
      end

      for j=1:n_particles % Runs particle by particle
        a = centre(j,1);
        b = centre(j,2);
        r = radius(j,1);
        % Calculation of the intersection points with each particle
        xint(j,1) = ( sqrt( a^2*m1^2 + 2*a*b*m1   2*a*m1*n1   b^2 +      ...
                    2*b*n1 + m1^2*r^2  n1^2 + r^2)+a+b*m1 m1*n1)/(m1^2+1);
        xint(j,2) =  (sqrt( a^2*m1^2 + 2*a*b*m1  2*a*m1*n1  b^2 +     ...
                    2*b*n1 + m1^2*r^2  n1^2 + r^2)+a+b*m1 m1*n1)/(m1^2+1);
      end

      XX = ones(n_particles,2)*DV2;

      for h=1:n_particles
        real_1 = isreal(xint(h,1));
        if real_1 == 1
          XX(h,1)= xint(h,1);
        end
        real_2 = isreal(xint(h,2));
        if real_2 == 1
          XX(h,2)= xint(h,2);
        end
      end

      YY = ones(n_particles,2)*DV2;

      for u=1:n_particles
        if XX(u,1)~= DV2
          YY(u,1)= m1*XX(u,1)+n1;
        end
        if XX(u,2)~= DV2
          YY(u,2)= m1*XX(u,2)+n1;
        end
      end
      for p=1:n_particles
        if YY(p,1)==DV2
          XX(p,1)=DV2;
        end
        if YY(p,2)==DV2
          XX(p,2)=DV2;
        end
      end

      % This loop prevents the reflected rays to go through the particles
      if prev_refl_in_part==1 && z~=1
        if abs(XX(row_previous,1) true_xint(z 1,i)) <= 0.0000000001
          xreflection_point = XX(row_previous,1);
          yreflection_point = YY(row_previous,1);
          ximpossible_point = XX(row_previous,2);
          yimpossible_point = YY(row_previous,2);
        end
        if abs(XX(row_previous,2) true_xint(z 1,i)) <= 0.0000000001
          xreflection_point = XX(row_previous,2);
          yreflection_point = YY(row_previous,2);
          ximpossible_point = XX(row_previous,1);
          yimpossible_point = YY(row_previous,1);
        end
        if yreflection_point > yimpossible_point
          % Points lower than yreflection_point are erased
          for ii=1:n_particles
          if YY(ii,1) < yreflection_point
            YY(ii,1) = DV2;
            XX(ii,1) = DV2;
          end
          if YY(ii,2) < yreflection_point
            YY(ii,2) = DV2;
            XX(ii,2) = DV2;
          end
        end
      else % if yreflection_point < yimpossible_point
        % Points higher than yreflection_point are erased
        for ii=1:n_particles
          if YY(ii,1) > yreflection_point
            YY(ii,1) = DV2;
            XX(ii,1) = DV2;
          end
          if YY(ii,2) > yreflection_point
            YY(ii,2) = DV2;
            XX(ii,2) = DV2;
          end
        end
      end
      % The intersections with the same particle that reflected the ray
      % are erased
      XX(row_previous,1) = DV2;
      XX(row_previous,2) = DV2;
      YY(row_previous,1) = DV2;
      YY(row_previous,2) = DV2;
    end

    A = XX~=DV2;
    B = A(:);
    C = any(B); % If C=1: There are intersections with particles
                % If C=0: There are no intersections, all the possible
                %         ones were ideal
    if C==1
      if z==1
        x_or = points_origin(i,1);
        y_or = points_origin(i,2);
      end
      % If z>1, the origin of the ray will be the intersection point of
      %  the previous iteration of z
      DD = ones(n_particles,2)*DV2;
      for s=1:n_particles
        if XX(s,1)~= DV2
          DD(s,1)= sqrt((abs(x_or   XX(s,1)))^2+(abs(y_or   YY(s,1)))^2);
        end
        if XX(s,2)~= DV2
          DD(s,2)= sqrt((abs(x_or   XX(s,2)))^2+(abs(y_or   YY(s,2)))^2);
        end
      end

      shortest_d      = min(min(DD));
      [row,col]       = find(DD==shortest_d); % Position of the shortest
```

```
        row_previous    = row;                    % distance in matrix DD
        xint_true       = XX(row,col);            % (row:particle intersected)
        true_xint(z,i)  = xint_true;
        yint            = m1*xint_true+n1;
        true_yint(z,i)  = yint;

        ray_intersections(z,1,i) = row;        % Stores the number of the
                                               % particle intersected
        ray_intersections(z,2,i) = xint_true;  % Stores the X coordinate of
                                               % the intersection
        ray_intersections(z,3,i) = yint;       % Stores the Y coordinate of
                                               % the intersection
        ray_intersections(z,4,i) = z;          % Stores the number of the
                                               % reflection
        ray_intersections(z,5,i) = n_wall_refl;

        ray_intersections(z,6,i) = ray_intersections(z,4,i)           ...
                                                  ray_intersections(z,5,i);

        x_or   = xint_true; % This will be the origin of the ray for the
        y_or   = yint;      % next iteration of z

        a      = centre(row,1);
        b      = centre(row,2);

        % Generation of a random angle gamma in the range 0 180 degrees
        gamma = pi*rand;

        % Calculation of the angle alpha between the intersection point and
        % a horizontal line that goes across the center of the circle
        alpha = mod(atan2((yint  b),(xint_true  a)),2*pi);

        beta   = alpha  (pi/2);

        % Calculation of the final random point that the ray will be
        % reflected towards
        k_length = 30; % This number must simply be high enough so that the
                       % random point(x_r,y_r) is outside the area of study

        x_r  =  xint_true  +  k_length*cos(gamma+beta);
        y_r  =  yint       +  k_length*sin(gamma+beta);

        % Line parameters for the ecuation of the reflected ray
        m2 = (y_r  yint)/(x_r  xint_true);
        n2 = (x_r*yint  xint_true*y_r)/(x_r  xint_true);

        rays(z+1,1,i) = m2;
        rays(z+1,2,i) = n2;

        % Plotting of the rays (Postprocessing)
        if z==1
          plot([points_origin(i,1) true_xint(z,i)],[points_origin(i,2)  ...
                                                        true_yint(z,i)],'b')
          hold on
```

```
      else
        plot([true_xint(z 1,i) true_xint(z,i)],[true_yint(z 1,i)       ...
                                                    true_yint(z,i)],'b')
        hold on
      end
      prev_refl_in_part = 1;
    elseif C==0  %There is no intersection
      if z==1
        xor_c0    = points_origin(i,1);
        yor_c0    = points_origin(i,2);
        xdest_c0 = points_destination(i,1);
        ydest_c0 = points_destination(i,2);
      elseif z~=1
        xor_c0    = true_xint(z 1,i);
        yor_c0    = true_yint(z 1,i);
        if prev_refl_in_part==1
          if yreflection_point < yimpossible_point
            ydest_c0 =  0.030;
            xdest_c0 = (ydest_c0  rays(z,2,i))/rays(z,1,i); % x=(y  n)/m
          elseif yreflection_point > yimpossible_point
            ydest_c0 = 0.030;
            xdest_c0 = (ydest_c0  rays(z,2,i))/rays(z,1,i); % x=(y  n)/m
          end
        elseif prev_refl_in_part==0
          xdest_c0 =  0.030;
          ydest_c0 = rays(z,1,i)*xdest_c0+rays(z,2,i); % y=mx+n
        end
      end

      if xdest_c0>xor_c0
        x_wall = xdest;
        y_wall = rays(z,1,i)*x_wall + rays(z,2,i);

        true_xint(z,i) = x_wall;
        true_yint(z,i) = y_wall;

        ray_intersections(z,1,i) = 100;    % Stores the number of the
                                           % particle intersected
                                           %(100: reflection in wall)
        ray_intersections(z,2,i) = x_wall; % Stores the X coordinate of
                                           % the intersection
        ray_intersections(z,3,i) = y_wall; % Stores the Y coordinate of
                                           % the intersection
        ray_intersections(z,4,i) = z;      % Stores the number of the
                                           % reflection


        n_wall_refl = n_wall_refl+1; % Increase of 1 because the
                                     % reflection is in the back wall
        ray_intersections(z,5,i) = n_wall_refl;

        ray_intersections(z,6,i) = ray_intersections(z,4,i)          ...
                                               ray_intersections(z,5,i);

          x_or = x_wall; % This will be the origin of the ray for the
          y_or = y_wall; % next iteration of z

          gamma1     = pi*rand;
          beta1      = (pi/2);

          % Calculation of the final random point that the ray will be
          % reflected towards
          k_length = 30; % This number must simply be hight enough so that
                         % the random point(x_r,y_r) is outside the area of
                         % study
          x_r = x_wall + k_length*cos(gamma1+beta1);
          y_r = y_wall + k_length*sin(gamma1+beta1);

          m2 = (y_r  y_wall)/(x_r  x_wall);
          n2 = (x_r*y_wall  x_wall*y_r)/(x_r  x_wall);

          rays(z+1,1,i) = m2;
          rays(z+1,2,i) = n2;

          plot([xor_c0  x_wall],[yor_c0  y_wall],'b')
          hold on
          prev_refl_in_part = 0;
        elseif xdest_c0<xor_c0
          plot([xor_c0  xdest_c0],[yor_c0  ydest_c0],'b')
          hold on
        end
      end % from if C==1
      if C==0 && xdest_c0<xor_c0 % If there is not a reflection, finish
                                 % the corresponding ray
        break
      end
       axis ([ xaxismin_area xaxismax_area yaxismin_area yaxismax_area ])
    end
  end
%                        Solar rays generation (end)                       %

  % Storage of information about the solar rays reflections
  particle_intersections = zeros(DV4,6,n_particles);
  for p=1:n_particles
    position = 0;
    for j=1:n_rays
      if find(ray_intersections(:,1,j)==p) % If the studied ray is
                                           % reflected by the studied
                                           % particle
        [row,col] = find(ray_intersections(:,1,j)==p);
        n_times = max(size(row));
        for z = 1:max(size(row))
          position = position+1;
          particle_intersections(position,1,p) = j;
          particle_intersections(position,2,p) =                      ...
                                           ray_intersections(row(z),2,j);
          particle_intersections(position,3,p) =                      ...
                                           ray_intersections(row(z),3,j);
          particle_intersections(position,4,p) =                         ...
                                           ray_intersections(row(z),4,j);
          % Storage of the distance from every boundary node to the
          % intersection point
          distance_to_intersection = zeros(n_bn,1);
          for jj = 1:n_bn
            distance_to_intersection(jj,1) =                              ...
                            sqrt((particle_intersections(position,2,p)    ...
                                     vcorg_scaled_boundary(jj,1,p))^2 +   ...
                                 (particle_intersections(position,3,p)    ...
                                         vcorg_scaled_boundary(jj,2,p))^2);
          end
          % Calculation of the two nodes closest to the reflection point:
          % node_number1 and node_number2
          [min_distance1, node_number1]    = min(distance_to_intersection);
          distance_to_intersection(node_number1) = Inf;
          [min_distance2, node_number2]    = min(distance_to_intersection);
          % Finding of the boundary line where the reflection occurs
          [roww, columnn] = find(boundary_nodes==node_number1);
          if boundary_nodes(roww(1),columnn(2))==node_number2
            particle_intersections(position,5,p) = roww(1);
          elseif boundary_nodes(roww(2),columnn(1))==node_number2
            particle_intersections(position,5,p) = roww(2);
          end
          particle_intersections(position,6,p) =                         ...
                                           ray_intersections(row(z),6,j);
        end
      end
    end
  end

  n_intersections = zeros(1,n_particles);

  for p=1:n_particles
    aux = size(find(particle_intersections(:,1,p)~=0));
    n_intersections(1,p) = aux(1,1);
  end

  nrays_l_z = zeros(n_sections_ray,n_bars,n_particles);

  for p=1:n_particles
    for line=1:n_bars
      for z=1:n_sections_ray
        n = z 1;
        %If that particle has an intersection in that line
        if sum(ismember(particle_intersections(:,5,p),line))~=0
          [row]=find(particle_intersections(:,5,p)==line);
          contador=0;
          for i=1:max(size(row))
            if particle_intersections(row(i),6,p)==n
              contador=contador+1;
            end
          end
          nrays_l_z(z,line,p)=contador;
        end
      end
    end
  end

%                         Emitted rays generation (start)                    %

  if set_rays ~=1
    rays_em              = ones(n_sections_ray,2,n_rays_em,n_particles)*DV1;
    points_origin_em     = zeros(n_bn,2,n_particles);
    points_destination_em = zeros(n_bn,2,n_particles);
    for p=1:n_particles
      for i=1:n_bn
        node1   = boundary_nodes(i,1);
        node2   = boundary_nodes(i,2);
        x_node1 = vcorg_scaled(node1,1,p);
        y_node1 = vcorg_scaled(node1,2,p);
        x_node2 = vcorg_scaled(node2,1,p);
        y_node2 = vcorg_scaled(node2,2,p);

        % Origin coordinates
        x_orig  = (x_node1+x_node2)/2;
        y_orig  = (y_node1+y_node2)/2;

        points_origin_em(i,1,p) = x_orig;
        points_origin_em(i,2,p) = y_orig;

        % Destination coordinates
        a = centre(p,1);
        b = centre(p,2);
        if     x_orig > a
          x_dest = xdest; %xdest=0.015
          y_dest = ((x_dest  x_orig)/(a  x_orig))*(b  y_orig)+y_orig;
        elseif x_orig < a
          x_dest =  xdest;
          y_dest = ((x_dest  x_orig)/(a  x_orig))*(b  y_orig)+y_orig;
        end
        if x_orig == a
          if     y_orig > b
            y_dest = xdest;
            x_dest = a;
          elseif y_orig < b
            y_dest =  xdest;
            x_dest = a;
          end
        end
        points_destination_em(i,1,p) = x_dest;
        points_destination_em(i,2,p) = y_dest;
      end
    end

    ray_intersections_em = zeros(n_sections_ray,6,n_rays_em,n_particles);

    for p=1:n_particles
      % Reflection of the rays in the particles
      true_xint_em = ones(n_sections_ray,n_rays_em)*DV1;
      true_yint_em = ones(n_sections_ray,n_rays_em)*DV1;
      for i=1:n_rays_em          % Runs ray by ray from one particle
        prev_refl_in_part_em = 0;
        n_wall_refl_em       = 0;
        for z=1:n_sections_ray  % Runs all the reflections of one ray from
                                % one particle
          xint = ones(n_particles,2)*DV2; % Stores de values of all the
                                          % ”intersections”, real or ideal
          if z==1 %Initial section of the ray
            % Deduction of the line ecuation:
            m1 = (points_destination_em(i,2,p) points_origin_em(i,2,p))/...
                 ((points_destination_em(i,1,p) points_origin_em(i,1,p)));
            n1 = (points_destination_em(i,1,p)*points_origin_em(i,2,p)  ...
                 points_origin_em(i,1,p)*points_destination_em(i,2,p))/...
                   (points_destination_em(i,1,p) points_origin_em(i,1,p));
            rays_em(z,1,i,p) = m1;
            rays_em(z,2,i,p) = n1;
          else
            m1 = rays_em(z,1,i,p); % Previously stored, in the previous
            n1 = rays_em(z,2,i,p); % iteration of z
          end
          for j=1:n_particles
            a = centre(j,1);
            b = centre(j,2);
            r = radius(j,1);
            % Calculation of the intersection points with each particle
            xint(j,1) = ( sqrt( a^2*m1^2 + 2*a*b*m1   2*a*m1*n1   b^2 + ...
                    2*b*n1 + m1^2*r^2   n1^2 + r^2)+a+b*m1 m1*n1)/(m1^2+1);
            xint(j,2) =  (sqrt( a^2*m1^2 + 2*a*b*m1   2*a*m1*n1   b^2 + ...
                    2*b*n1 + m1^2*r^2   n1^2 + r^2)+a+b*m1 m1*n1)/(m1^2+1);
          end

          XX = ones(n_particles,2)*DV2;

          for h=1:n_particles
            real_1 = isreal(xint(h,1));
            if real_1 == 1
              XX(h,1)= xint(h,1);
            end
            real_2 = isreal(xint(h,2));
            if real_2 == 1
              XX(h,2)= xint(h,2);
            end
          end

          YY = ones(n_particles,2)*DV2;

          for u=1:n_particles
            if XX(u,1)~= DV2
              YY(u,1)= m1*XX(u,1)+n1;
            end
            if XX(u,2)~= DV2
              YY(u,2)= m1*XX(u,2)+n1;
            end
          end
          for q=1:n_particles
            if YY(q,1)==DV2
              XX(q,1)=DV2;
            end
            if YY(q,2)==DV2
              XX(q,2)=DV2;
            end
          end
          if z==1
            % points_origin_em(i,2,p)) is Y from the origin of the emission
            if points_origin_em(i,2,p) > centre(p,2)
              % Points lower than yreflection_point are erased
              for ii=1:n_particles
                if YY(ii,1) < points_origin_em(i,2,p)
                  YY(ii,1) = DV2;
                  XX(ii,1) = DV2;
                end
                if YY(ii,2) < points_origin_em(i,2,p)
                  YY(ii,2) = DV2;
                  XX(ii,2) = DV2;
                end
              end
            elseif points_origin_em(i,2,p) < centre(p,2)
              % Points higher than yreflection_point are erased
              for ii=1:n_particles
                if YY(ii,1) > points_origin_em(i,2,p)
                  YY(ii,1) = DV2;
                  XX(ii,1) = DV2;
                end
                if YY(ii,2) > points_origin_em(i,2,p)
                  YY(ii,2) = DV2;
                  XX(ii,2) = DV2;
                end
              end
            end
            XX(p,1) = DV2; % The intersections with the particle that EMITS
            XX(p,2) = DV2; % the ray are erased (they are not possible)
            YY(p,1) = DV2;
            YY(p,2) = DV2;
          end
          % This loop prevents the reflected rays to go through the
          % particles
          if prev_refl_in_part_em==1 && z~=1
            if abs(XX(row_previous,1) true_xint_em(z 1,i)) <= 0.0000000001
              xreflection_point = XX(row_previous,1);
              yreflection_point = YY(row_previous,1);
              ximpossible_point = XX(row_previous,2);
              yimpossible_point = YY(row_previous,2);
            end
            if abs(XX(row_previous,2) true_xint_em(z 1,i)) <= 0.0000000001
              xreflection_point = XX(row_previous,2);
              yreflection_point = YY(row_previous,2);
              ximpossible_point = XX(row_previous,1);
              yimpossible_point = YY(row_previous,1);
            end
            if yreflection_point > yimpossible_point
              % Points lower than yreflection_point are erased
              for ii=1:n_particles
                if YY(ii,1) < yreflection_point
                  YY(ii,1) = DV2;
                  XX(ii,1) = DV2;
                end
                if YY(ii,2) < yreflection_point
                  YY(ii,2) = DV2;
                  XX(ii,2) = DV2;
                end
              end
            elseif yreflection_point < yimpossible_point
              % Points higher than yreflection_point are erased
              for ii=1:n_particles
                if YY(ii,1) > yreflection_point
                  YY(ii,1) = DV2;
                  XX(ii,1) = DV2;
                end
                if YY(ii,2) > yreflection_point
                  YY(ii,2) = DV2;
                  XX(ii,2) = DV2;
                end
              end
            end

            XX(row_previous,1) = DV2; % The intersections with the particle
            XX(row_previous,2) = DV2; % that reflects the ray are erased
            YY(row_previous,1) = DV2; % (they are not possible)
            YY(row_previous,2) = DV2;
          end
          A = XX~=DV2;
          B = A(:);
          C = any(B);
          if C==1
            if z==1
              x_or = points_origin_em(i,1,p);
              y_or = points_origin_em(i,2,p);
            end % If z>1, the origin will be the intersection point of the
                % previous iteration of z
            DD = ones(n_particles,2)*DV2;
            for s=1:n_particles
              if XX(s,1)~= DV2
                DD(s,1)= sqrt((abs(x_or XX(s,1)))^2+(abs(y_or YY(s,1)))^2);
              end
              if XX(s,2)~= DV2
                DD(s,2)= sqrt((abs(x_or XX(s,2)))^2+(abs(y_or YY(s,2)))^2);
              end
            end

```

```
        shortest_d       = min(min(DD));
        [row,col]        = find(DD==shortest_d);
        row_previous     = row;

        xint_true          = XX(row,col);
        true_xint_em(z,i)  = xint_true;
        yint               = m1*xint_true+n1;
        true_yint_em(z,i)  = yint;

        ray_intersections_em(z,1,i,p) = row;        % Stores the number
                                                    % of the particle
                                                    % intersected
        ray_intersections_em(z,2,i,p) = xint_true;  % Stores the
                                                    % X coordinate of
                                                    % the intersection
        ray_intersections_em(z,3,i,p) = yint;       % Stores the
                                                    % Y coordinate of
                                                    % the intersection
        ray_intersections_em(z,4,i,p) = z;          % Stores the section
                                                    % of the reflection
        ray_intersections_em(z,5,i,p) = n_wall_refl_em;

        ray_intersections_em(z,6,i,p) =                              ...
           ray_intersections_em(z,4,i,p) ray_intersections_em(z,5,i,p);

        x_or = xint_true;
        y_or = yint;

        a    = centre(row,1);
        b    = centre(row,2);

        % Generation of a random angle gamma in the range 0 180 degrees
        gamma = pi*rand;
        % Calculation of the angle alpha between the intersection point
        % and a horizontal line that goes across the center of the
        % circle
        alpha = mod(atan2(yint b,xint_true a),2*pi);

        beta  = alpha  (pi/2);

        % Calculation of the final random point that the ray will be
        % reflected towards
        x_r   = xint_true + k_length*cos(gamma+beta);
        y_r   = yint      + k_length*sin(gamma+beta);

        % Line parameters for the ecuation of the reflected ray:
        m2    = (y_r  yint)/(x_r  xint_true);
        n2    = (x_r*yint  xint_true*y_r)/(x_r  xint_true);

        rays_em(z+1,1,i,p) = m2;
        rays_em(z+1,2,i,p) = n2;

        % Plotting of the rays
        if z==1
```

```
              plot([points_origin_em(i,1,p) true_xint_em(z,i)],          ...
                           [points_origin_em(i,2,p) true_yint_em(z,i)],'r')
              hold on
            else
              plot([true_xint_em(z 1,i) true_xint_em(z,i)],              ...
                               [true_yint_em(z 1,i) true_yint_em(z,i)],'r')
              hold on
            end
            prev_refl_in_part_em = 1;
          elseif C==0 % There is no intersection
            if z==1
              xor_c0_em   = points_origin_em(i,1,p);
              yor_c0_em   = points_origin_em(i,2,p);
              xdest_c0_em = points_destination_em(i,1,p);
              ydest_c0_em = points_destination_em(i,2,p);
            elseif z~=1
              xor_c0_em   = true_xint_em(z 1,i);
              yor_c0_em   = true_yint_em(z 1,i);
              if prev_refl_in_part_em==1
                if yreflection_point < yimpossible_point
                  ydest_c0_em =  0.030;
                  xdest_c0_em = (ydest_c0_em rays_em(z,2,i,p))/          ...
                                                rays_em(z,1,i,p); %x=(y n)/m
                elseif yreflection_point > yimpossible_point
                  ydest_c0_em = 0.030;
                  xdest_c0_em = (ydest_c0_em rays_em(z,2,i,p))/          ...
                                                rays_em(z,1,i,p); %x=(y n)/m
                end
              elseif prev_refl_in_part_em==0
                xdest_c0_em =  0.030;
                ydest_c0_em = rays_em(z,1,i,p)*xdest_c0_em+rays_em(z,2,i,p);
              end                                                  %y=mx+n
            end
            if xdest_c0_em>xor_c0_em
              x_wall = xdest;
              y_wall = rays_em(z,1,i,p)*x_wall + rays_em(z,2,i,p);

              true_xint_em(z,i) = x_wall;
              true_yint_em(z,i) = y_wall;

              ray_intersections_em(z,1,i,p) = 100;    % Stores the number
                                                      % of the particle
                                                      % intersected (100:
                                                      % reflection in wall)
              ray_intersections_em(z,2,i,p) = x_wall; % Stores the
                                                      % X coordinate of the
                                                      % intersection
              ray_intersections_em(z,3,i,p) = y_wall; % Stores the
                                                      % Y coordinate of the
                                                      % intersection
              ray_intersections_em(z,4,i,p) = z;      % Stores the number
                                                      % of the reflection

              n_wall_refl_em = n_wall_refl_em+1; % Increase of 1 because
```

```
                                             % the reflection is in
                                             % the back wall
          ray_intersections_em(z,5,i,p) = n_wall_refl_em;

          ray_intersections_em(z,6,i,p) =                           ...
           ray_intersections_em(z,4,i,p)  ray_intersections_em(z,5,i,p);

          x_or = x_wall; % This will be the origin of the ray for the
          y_or = y_wall; % next iteration of z

          gamma1 = pi*rand;
          beta1  = (pi/2);

          % Calculation of the final random point that the ray will be
          % reflected towards
          x_r = x_wall + k_length*cos(gamma1+beta1);
          y_r = y_wall + k_length*sin(gamma1+beta1);

          m2 = (y_r  y_wall)/(x_r  x_wall);
          n2 = (x_r*y_wall  x_wall*y_r)/(x_r  x_wall);

          rays_em(z+1,1,i,p) = m2;
          rays_em(z+1,2,i,p) = n2;

          plot([xor_c0_em  x_wall],[yor_c0_em  y_wall],'r')
          hold on
          prev_refl_in_part_em = 0;
        elseif xdest_c0_em<xor_c0_em
          plot([xor_c0_em  xdest_c0_em],[yor_c0_em  ydest_c0_em],'r')
          hold on
        end
      end % from if C==1

      if C==0 && xdest_c0_em<xor_c0_em % If there is not a reflection ,
        break                          % finish the corresponding ray
      end
    end
  end
end
axis ([ xaxismin_area xaxismax_area yaxismin_area yaxismax_area ])

% Calculation of the emitted energy
emission_radiation = zeros(n_rays_em,n_particles);
for p = 1:n_particles
  for i=1:n_bars
    node1 = boundary_nodes(i,1);
    node2 = boundary_nodes(i,2);

    temp_node1 = vsol(node1,p);
    temp_node2 = vsol(node2,p);

    temp_med_line = (temp_node1+temp_node2)/2;

    emission_radiation(i,p) = e*s_b*(temp_med_line+273.15)^4;
```

```
      end
    end

    % Storage of the emitted energy INCIDENT in the boundary lines
    particle_intersections_em = zeros(DV3,6,n_particles);
    for particle=1:n_particles % Studied particle
      counter = 0;
      for p=1:n_particles % Particle by particle, studying their emitted
                          % rays
        for i=1:n_rays_em % Ray by ray emitted by the corresponding
                          % particle
          % If the ray i emitted by particle p hits the particle "particle"
          if ismember(particle,ray_intersections_em(:,1,i,p))==1
            indices = find(ray_intersections_em(:,1,i,p)==particle);
            reps    = numel(find(ray_intersections_em(:,1,i,p)==particle));
            for j=1:reps
              counter = counter+1;
              particle_intersections_em(counter,1,particle) = i;
              particle_intersections_em(counter,2,particle) = p;
              particle_intersections_em(counter,3,particle) =           ...
                                       ray_intersections_em(indices(j),4,i,p);
              particle_intersections_em(counter,4,particle) =           ...
                                  emission_radiation(i,p)*arc_length(p,1); % W

              % Calculation of the affected boundary line
              distance_to_intersection = zeros(n_bn,1);
              x_int = ray_intersections_em(indices(j),2,i,p);
              y_int = ray_intersections_em(indices(j),3,i,p);

              % Storage of the distance from every boundary node to the
              % intersection point
              for jj = 1:n_bn
                distance_to_intersection(jj,1) = sqrt((x_int            ...
                            vcorg_scaled_boundary(jj,1,p))^2 + (y_int   ...
                                         vcorg_scaled_boundary(jj,2,p))^2);
              end

              % Calculation of the two nodes closest to the reflection
              % point: node_number1 and node_number2
              [min_distance1, node_number1]= min(distance_to_intersection);
              distance_to_intersection(node_number1) = Inf;
              [min_distance2, node_number2]= min(distance_to_intersection);

              % Finding of the boundary line where the reflection occurs
              [row, column] = find(boundary_nodes==node_number1);
              if boundary_nodes(row(1),column(2))==node_number2
                particle_intersections_em(counter,5,particle)=row(1);
              elseif boundary_nodes(row(2),column(1))==node_number2
                particle_intersections_em(counter,5,particle)=row(2);
              end
              particle_intersections_em(counter,6,particle) =           ...
                                       ray_intersections_em(indices(j),6,i,p);
            end
          end
        end
      end
    end

    emission_radiation_incident = zeros(n_bars,n_particles);
    for p = 1:n_particles
      for i = 1:n_bars
        repetitions = numel(find(particle_intersections_em(:,5,p)==i));
        [row, column] = find(particle_intersections_em(:,5,p)==i);
        total_abs_em = 0;
        for j=1:repetitions
          ray_n  = particle_intersections_em(row(j,1),1,p);
          part_n = particle_intersections_em(row(j,1),2,p);
          n      = particle_intersections_em(row(j,1),6,p);
          abs_en = (particle_intersections_em(row(j,1),4,p)/          ...
                                               arc_length(p,1))*e*((1 e)^n); % W/m^2
          total_abs_em  = total_abs_em + abs_en;
        end
        emission_radiation_incident(i,p) = total_abs_em; % W/m^2
      end
    end
  end

%==========================================================================
% PART 5.2    HEAT TRANSFER RESOLUTION WITH THE FEM
%==========================================================================

  n_BC_matrix     = zeros(nelt,1,n_particles);
  vkg_particles   = zeros(ndlt,ndlt,n_particles);
  vmg_particles   = zeros(ndlt,ndlt,n_particles);
  vfg_particles   = zeros(ndlt,1,n_particles);
  incident_sun    = zeros(n_bars,1,p);

  for p = 1:n_particles
    n_BC_matrix(:,:,p) = n_BC;
  end
  if set_rays==1
    emission_radiation          = zeros(n_rays_em,n_particles);
    emission_radiation_incident = zeros(n_bars,n_particles);
    particle_intersections_em   = zeros(DV3,5,n_particles);
  end

  % Change of the BC vector according to the rays reflections
  for p = 1:n_particles
    for i = 1:n_intersections(1,p)
      n_BC_matrix(particle_intersections(i,5,p),p) = 12;
    end
  end

  % Assembling of vmg, vkg and vfg
  for p = 1:n_particles
    vkg  = zeros(ndlt);
    vmg  = zeros(ndlt);
    vfg  = zeros(ndlt,1);
    ndle = 3;
    kloce= 0*(1:ndle);

    for ie = 1:nelt
      inel = 2;
      itype = mod(floor(n_BC_matrix(ie,1,p)/10),10); % Number of tens
      iclass = mod(floor(n_BC_matrix(ie,1,p)),10);   % Number of units
      if(itype == 3)
        inel = 3;
      end
      vcore = vcorg_scaled(kconec(ie, 1:inel),:,p);
      if ie<=n_bars
        for ii = 1:n_sections_ray
          n = ii 1;
          incident_sun(ie,1,p) = incident_sun(ie,1,p) +                ...
                                             nrays_l_z(ii,ie,p)*(1 absp)^n;
        end
      end
      switch itype
        case 1
          data_boundaryconditions_lines
          vprel = vprel_Neumann(iclass,:);
          barre_neumann;
        case 2
          data_boundaryconditions_lines
          vprel = vprel_Cauchy(iclass,:);
          barre_cauchy;
        case 3
          data_boundaryconditions_triangles
          vprel = vprel_T3(iclass,:);
          ther_T3;
      end
      kloce = [];
      for in = 1:inel
        kloce = [kloce,(kconec(ie, in) 1)*ndln+(1:ndln)];
      end
      % Assembling of vke
      vkg(kloce,kloce) = vkg(kloce,kloce) + vke;
      % Assembling of vme
      vmg(kloce,kloce) = vmg(kloce,kloce) + vme;
      % Assembling of vfe
      vfg(kloce,1) = vfg(kloce,1) + vfe;
      clearvars vcore
    end
    vkg_particles(:,:,p) = vkg;
    vmg_particles(:,:,p) = vmg;
    vfg_particles(:,:,p) = vfg;
  end

  vkt = zeros(ndlt,ndlt,n_particles);
  for p = 1:n_particles
    vkt(:,:,p) = vmg_particles(:,:,p) + (scheme*dt)*vkg_particles(:,:,p);
    vkt(:,:,p) = sparse(vkt(:,:,p)); % sparse
  end
```

```

  vsol_stored = zeros(nnt,1,(nsteps/10),n_particles); % Stores the solution
                                                       % every 10 timsteps
  for istep = 1:nsteps
    time = time_previous_iteration + istep*dt;
    for p = 1:n_particles
      vres       = vfg_particles(:,:,p)    vkg_particles(:,:,p)*vsol(:,p);
      % Residue for time control
      vdu        = vkt(:,:,p)\(dt*vres);
      vsol(:,p) = vsol(:,p) + vdu; % The solution is updated

    end
    % Display of the solution every 10 timesteps
    if(mod(istep,10)==0)
      count_fig = count_fig + 1;
      figure(count_fig)
      for p = 1:n_particles
        vsol_stored(:,1,istep/10,p) = vsol(:,p);
        caxis([20 500])
        femplot(vcorg_scaled(:,:,p),kconec,vsol_stored(:,:,istep/10,p),1)
        shading interp;
        set(gca,'fontsize',16);
        set(gca, 'FontName', 'Times New Roman');
        title_plot = ['Time: ' num2str(time),' s'];
        title(title_plot)
        colorbar

        rectangle('Position',[xorig,yorigmin,0.000001,                    ...
              abs(yorigmax  yorigmin)],'EdgeColor','y','LineWidth', 1.5,  ...
                                                        'LineStyle','  ')

        rectangle('Position',[xdest,ydestmin,0.000001,                    ...
             abs(ydestmax  ydestmin)],'EdgeColor', 'b','LineWidth', 1.5,  ...
                                                        'LineStyle','  ')
        hold on
        title(colorbar,'  C ')
        xlabel('Depth (m)')
        ylabel('Window length (m)')
        axis equal
        axis ([ xaxismin_area xaxismax_area yaxismin_area yaxismax_area ])
      end
      time_axis(1,set_rays+1) = time;
      for p=1:n_particles
        average_temp(p,set_rays+1) = sum(vsol(:,p))/nnt;
      end
    end
  end
  time_previous_iteration = time;
end

[radius_ascend,indexes] = sort(radius);
sorted_average_temp = average_temp(indexes,:);
radius_ascend_mm=radius_ascend*10^3;

```

```
figure(200)
for p=1:n_particles
  title(' Temporal evolution of the average temperature of the particles')
  xlabel('Time (s)')          % x axis label
  ylabel('Temperature ( C)') % y axis label
  plot(time_axis,sorted_average_temp(p,:))

  hold on
end

legend(strcat('r = ', num2str(radius_ascend_mm),' mm'),'Location',     ...
                                                              'northwest')
```

*Note: The symbols - have not been properly displayed.