Thermal Analysis in Railway Electrification Systems

by

Alejandro Gancedo Montes

Submitted to the Department of Electrical Engineering and Electronics Systems
in partial fulfillment of the requirements for the degree of
Electrical Energy Conversion and Power Systems Master's Degree
at the
UNIVERSIDAD DE OVIEDO
July 2019

Author . . . .

Certified by . . . .
Peru Bidaguren Sarricolea
Engineering Division - CAF TE
Thesis Supervisor

Certified by . . . .
Pablo Arboleya Arboleya
Associate Professor
Thesis Supervisor

Thermal Analysis in Railway Electrification Systems

by

Alejandro Gancedo Montes

Submitted to the Department of Electrical Engineering and Electronics Systems
on July 22, 2019, in partial fulfillment of the
requirements for the degree of
Electrical Energy Conversion and Power Systems Master's Degree

Abstract

In this Master's Thesis, a tool to calculate the thermal behavior of railway electrification systems has been developed by means of iterative calculations based on heat transfer and energy conversion equations. This application is called RailThermal app. The calculation method is implemented in Matlab, allowing to study the thermal capacity of the system. The aim of this tool is to size the section of the conductors (catenary, feeder) and calculate the recovery time of the system against unexpected contingencies.

With the results supplied by the program, a thermal map of the system is created. For lines, a evolution of the temperature against time and distance is plotted.

The input data is obtained from RailNeos 2.0, a simulator of railway electrification systems which is able to calculate the total energy consumption of the system, taking into account the implementation of accumulation devices. This tool was developed by Lemur Research Group of the University of Oviedo and CAF Tunrkey & Engineering.

Thesis Supervisor: Peru Bidaguren Sarricolea
Title: Engineering Division - CAF TE

Thesis Supervisor: Pablo Arboleya Arboleya
Title: Associate Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The current transport system shows sustainable limitations in terms of pollution and congestion. Electrical railway is a means of transport that solves some of these problems, being an efficient, safe and clean system. The railway system allows the massive transport of goods and passengers in urban and suburban areas.

Within EU28, the transport sector is responsible for 24% of greenhouse gases (GHG) emissions, being the only sector in which emissions have increased since 1990. From 1990 to 2012, GHG increased by 14%. However from the same period, European railway system reduced the total CO_2 emissions by 39 %. In Fig 1-1, the percentages of emissions by sectors is shown. Direct emissions from the railway sector by using diesel as fuel are 0.6%. The emissions of railway electrified system from electricity production reached 1.5%. CO_2 emissions from the road sector is 70.9%, as well as aviation and shipping 12.6% and 14.4%, respectively [35].

Therefore, a change to a new transportation model is necessary. In railway system that change has already begun. Since electrical systems show clear advantages over diesel propulsion systems. The electrical traction systems provide a better power to weight ratio than diesel, allowing a faster acceleration and bigger tractive effort on sections with pronounced slopes. In negatives gradients the locomotives operate as generators injecting power to the grid and improving the global efficiency of the system. It is also considered a low emissions means of transport, specially, in places where the majority of electric generation comes from renewable sources. Furthermore, it is a silent means of transport that requires a lower degree of maintenance than diesel traction units. The suburban railway systems are the most attractive for implementing, since the density traffic is high and the distance between stations is relatively short (CAPEX in civil works are lower in comparison with long distances trains) [16].

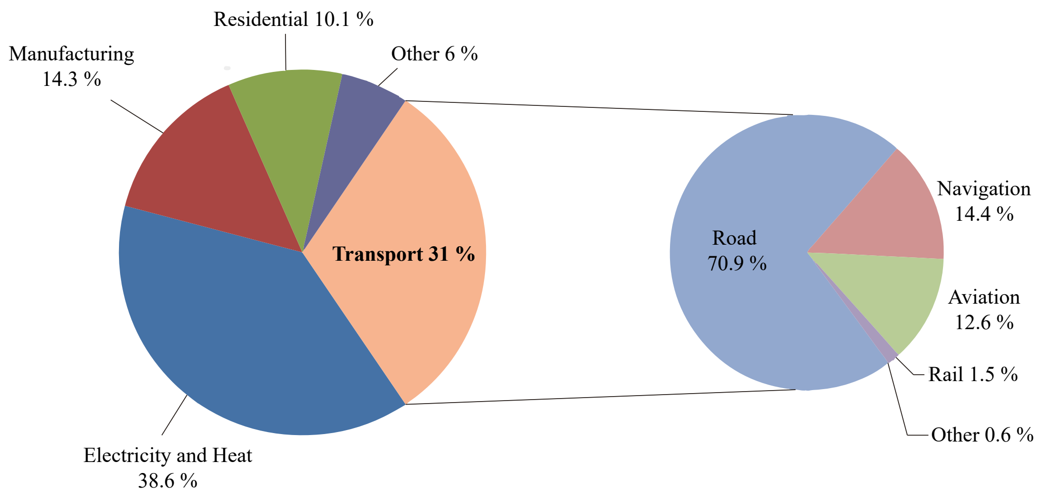

Figure 1-1: EU 27 share of CO_2 from fuel combustion. Note: emissions from aviation and navigation (maritime) international bunkers and electric rail traction system are considered in transport sector.

Table 1.1: EU transport modal share, 2011 [35]

	Passenger (pass-km)	**Freight (tonne-km)**	**Total (TU)**
ROAD	83.60%	46.90%	**70.30%**
AVIATION	8.80%	0.10%	**5.70%**
NAVIGATION	0.60%	41.90%	**15.50%**
RAIL	7.00%	11.10%	**8.50%**

Currently, in the electric generation and distribution system, the trend of smart electric networks predominate, where renewable energies and storage systems play a crucial role. At the same time, the requirements to increase the power capacity in railway systems and the tendency to carry out an active power control between utility, railway grid and energy storage systems, highlight the inconveniences of the conventional systems in order to maintain good power quality ratios. Therefore, the future railway feeding system must be an adaptable infrastructure, which must be capable to deal with the new generation sources and fulfilling the requirements of reliability, efficiency and economic viability.

The need to consolidate a transport system that was capable of moving large masses of passengers and goods, reducing the environmental impact is evident. The road-map foresees that the length of high-speed networks will triple in 2030, thus covering most of the passengers in middle distance. For transport of goods by road, the 30% will be done by train or ship, in distances greater than 300 km, and more than 50% in 2050 [35].

In Tab.1.1, it is presented the European Union transport modal share. In future years, this panorama is expected to change significantly. The EU plans that electricity be the central energy axis of the future, using low emission generation systems. This new energetic model must supply the transport sector, replacing the use of fossil fuels. In this situation, the railway system is the main means of transport capable of adapting to this new energy model. Currently, more than half of the rail network in Europe is electrified. In order to become the first means of transport with zero-carbon emissions, a push must be necessary, allowing favorable policy and investments in the infrastructure [35].

Foreseeing this encouraging scenario, the stage of design and study of new networks takes a great importance. Either, power flows and thermal studies allow taking the correct direction of the new network design. A detailed prior study is the first solution to future problems in the grid, in order to take the correct direction in terms of reliability and profitability for a new project

1.1 Structure of the thesis

This master thesis is split into 9 chapters. Below, a brief description of each of them is shown

- **Chapter 1**: A brief description of the current situation in the energy world is presented.

- **Chapter 2**: In this chapter a review of the main railways feeding systems and a presentation of RailNeos are carried out.
- **Chapter 3**: The thermal model of overhead lines and the main effects that can appear on the conductor are described.
- **Chapter 4**: Along this chapter, the main heat transfer procedures are defined.
- **Chapter 5**: The temperature calculation method implemented in RailThermal app is explained in this chapter.
- **Chapter 6**: The thermal calculation tool is presented, showing the interface and its functionalities.
- **Chapter 7**: A description of the code is carried out in this chapter, explaining the flowchart of the software.
- **Chapter 8**: Along this chapter, an analysis of the main variables of the calculation method and a case study are performed.
- **Chapter 9**: Finally, the conclusions extracted from the thesis are presented. In addition, future development paths are shown.

Chapter 2

State of the art

2.1 Railway Lines Feeding

The energy feeding system is the combination of elements necessaries for supplying, distributing and transforming the electric energy of the grid, into energy able to allow the reliable and interruptible circulation of rolling stock. The main components of the railway feeding system are:

- Traction power substations (TPSS).
- Railway electrification systems composed by contact lines, feeders and return conductors.

The utility grid supplies power to TPSS in which the voltage level is transformed and rectified (in case of DC systems), allowing to feed the rolling stock. The feeding is carried out through medium voltage grids. Then, the power is supplied to the catenary by feeders. The circuit is closed via the rails which are connected to a return circuit towards the substation. The return circuit is isolated from earth. There are protection devices in the return cell of the substation, that are checking the voltage between rail and earth. If voltage value exceeds the limit, rails will be connected to earth, until the current through the rails is dissipated. [33]

The railways lines are split into sections electrically isolated. Short stretches without feeding separate sections. They are called neutral zones. Along these zones the rolling stocks are moving without voltage. The sections with voltage are fed from the high voltage network through TPSS. Usually, the same substation feeds two consecutive sections. For those vehicles that are moving with more than one pantograph connected, the distance between the farthest pantograph must be lower than the length of the neutral zone, otherwise different phases could be connected leading to a short-circuit. For high speed trains, the length of the neutral zone could be around 400 meters [19].

2.1.1 Traction power substations (TPSS)

The power traction substation connects the railway electrified lines and the hight-voltage network. This installation transforms voltages from the three phase grid level to the feeder levels. In DC systems, it is necessary a rectification stage [19]. There are different topologies for feeding TPSS:

- **Direct feeding from the utility network**: the supplier company is in charge of feeding each TPSS. Usually, TPSS presents a topology of single busbar. This configuration leads to a lower cost than the other options. In case that it was necessary to feed several lines from a TPSS, other topologies as ring or double busbar are more advisable.

- **Medium voltage ring**: all TPSS are connected to the feed ring. This topology increases the availability of the substations.

TPSS are supervised from the control center by means of a remote control. This allows a coordinate operation of TPSS, adapting the topology of the grid according to the needs.

2.1.2 Traction circuit

The traction circuit is the installation composed by the conductors in charge of distributing the electric energy. In Fig. 2-1, the main components of the traction circuit are shown.

The conductors with positive voltage are known as catenary. Below, the main parts are presented [19]:

- **Contact wire** is the conductor in which the pantograph gets in contact. This wire is in parallel with the ground in order to facilitate the caption of power. Usually, the conductors are made of Cu-ETP or CuAg0.1.

- **Suspension wire** is the conductor in charge of supporting the weight of the contact wire by means of suspension cables. Commonly, they are copper conductors.

- **Feeder** is an additional conductor added in case that the impedance must be reduced and the permissible current limit must be increased. The most common is to use aluminum conductor with steel core.

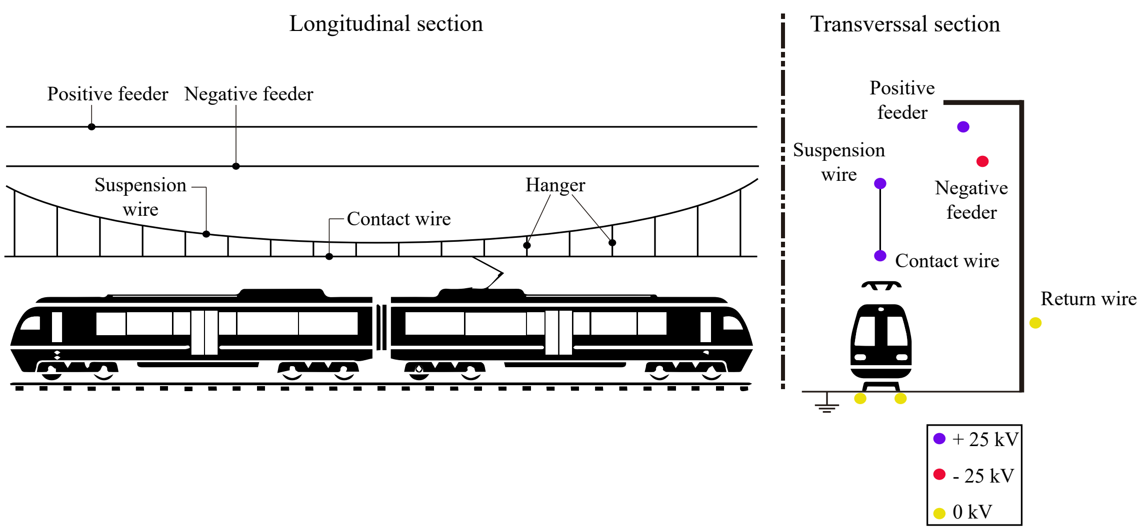

Figure 2-1: Configuration of traction systems in a 2x25 kV topology

Neutral conductors are:

- From the electrical point of view, **rails** works as return conductors and they are made of steel.

- **Return conductors** serve to return currents. Due to its lower impedance, they drive most part of the return current. In that way, the perturbations of the currents will be reduced, avoiding problems with the signaling systems. Commonly, aluminum conductor with steel core are used.

In some topologies of dual AC conductor, as 2x25 kV, an additional wire is added, called negative feeder. The purpose of this conductor is to configure the return circuit. This wire allows to reduce the electromagnetic perturbations. Usually, aluminum conductor with steel core are used.

2.1.3 Electrification systems

The electrical railway system can be split depending on the feeding voltage. The voltages standardized are established by BS EN 50163 and IEC 60850, in which it is considered number of trains circulating and longitudes to substations [16]. In the table 2.1, the standardized voltage range are specified.

Table 2.1: Standardized voltage ranges

System	V_{LNP}	V_L	V_N	V_{HP}	V_{HNP}
600 V DC	400 V	400 V	600 V	720 V	800 V
750 V DC	500 V	500 V	750 V	900 V	1000 V
1500 V DC	1000 V	1000 V	1500 V	1800 V	1950 V
3 kV DC	2 kV	2 kV	3 kV	3.6 kV	3.9 kV
15 kV AC (16.7 Hz)	11 kV	12 kV	15 kV	17.25 kV	18 kV
25 kV AC (50 Hz)	17.5 kV	19 kV	25 kV	27.5 kV	29 kV

Table 2.2: Nomenclature used in Tab.2.1

V_{LNP}	Lowest non-permanent voltage
V_L	Lowest permanent voltage
V_N	Nominal voltage
V_{HP}	Highest permanent voltage
V_{HNP}	Highest non-permanent voltage

At the beginning of the development of railway electrified systems, DC systems with low voltages were the most common. Primarily, because the speed of the motors could be controlled in a easy way through switches and rheostats. However, these low voltages leads to high currents circulating through the lines, going against the fundamental principles of the distribution lines. Coupled to this, in DC networks with high voltages and high currents, it is difficult to dissipate fault currents due to the energy stored in the inductances of the system. Technically, recent studies show that the maximum voltage in DC is 12 kV. Nowadays, the on-board control of torque, i.e, tractive effort, is no limited, therefore the use of topologies at 25 kV (50 Hz/ 60 Hz) is the most predominant option for covering long distances railways. However, for installed conductors with the same characteristics, the impedance in AC systems is higher than in DC. This is due to the flux between output and return overhead conductors and because of the skin effect in the returns rails which makes grow the apparent resistance up to 40 % with 50 Hz. This increase in resistance is more evident for low voltages. Therefore, from the economic point of view it is more beneficial to install DC systems with low voltages and high currents, such as in case of metros or trams [30].

In the table 2.3, it is presented the main topologies of traction systems taking into account the feeding voltage level.

Table 2.3: Traction system in Europe [10]

Rated Voltage	**Km**	**%**
600 V DC 750 V DC	3310	1.4
1.5 kV DC	15318	6.4
3.0 kV DC	72104	30.3
15 kV (16.7 Hz)	32392	13.6
25 kV(50 Hz, 60 Hz)	106437	44.8
Others AC	8039	3.4

2.1.4 DC electrified systems

The direct current (DC) is an simple electrification system used due to it allows an connection to the utility system without introducing unbalance voltages into the railway system [26]. This system requires a short distances between stations because of the resistive losses. The desired distance between two station at 750 V DC is 2.5 km and at 3 kV is 25 km.

Rolling stock in DC has the ability to exchange energy with the train network through power converters. The locomotives operates as generators, at the moment of braking. Thus, this process allows an improvement in the efficiency. Note that the regenerative breaking increases the voltage in DC part what it can lead to blocks in the non-reversible substations [24].

Usually, the trams are fed at 600 V DC or 750 V DC (most popular nowadays). Normally for feeding the trams, overhead lines or storage systems are used, either on-board and off-board. There are cases in which the trams can be powered by a third rail, as in Brussels, but it is not common, since the costs in the security and isolation systems are high. In Brussels the third rail is energized as the tram moves. The solution of a third rail is widely used in subways because of the reduction of capital costs, allowing to minimize the section of the tunnels and due to it is able to transport 41 % of additional power than an AC systems with an identical peak voltage [16]. Other solutions can be used as in the case of the London Underground, with a third rail at 420 V and a quarter rail at -210 V (taking as reference the rails of the subway). 1400 V and 1500 V DC are the widely extended solutions for metro due to economical terms. In the range of 3000 V DC, suburbans and trains are found with overhead lines [30]

DC power supplies

As previously mentioned, traction systems in DC are used mainly in urban, suburban and regional systems. The conversion traction power substations connect the utility grid, of medium or high voltage grid with the railway electric system. The rectifier bridges are the responsible of transform from AC to DC.

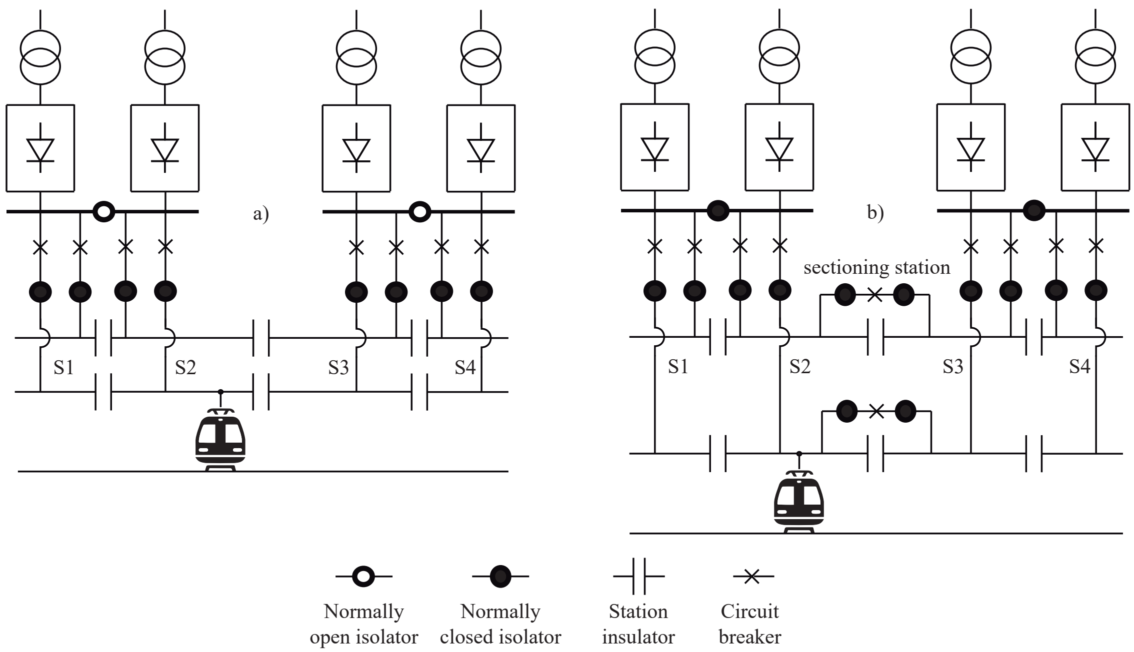

Figure 2-2: DC power feeding topologies: a) Single-end feed b) Double-end fed

Below, the components of a substation are presented [30]:

- Input transformer to transform the voltage from the AC side
- AC circuit breakers
- AC switches and isolators are used in emergency situations to reconnect with other feeders and in maintenance work
- Silicon diode bridges. At 750 V two 6-pulse bridges are connected in parallel and at 1500 V in series, achieving 12-pulse output ripple.
- DC switches to isolate a part of the line
- High speed DC circuit breakers are costly due to the difficulty of breaking a high DC current in an inductive circuit. Another types of this breakers are implemented on-board in order to reduce interruptions, disconnecting regenerative braking systems from nearby trains that may be feeding faults.

The electric traction system is split in sections where each one can be isolated from the others in case of any contingency or failure. The aim is to avoid adjacent sections being fed at different voltages. The electric power supply must operate without an isolated section, with no problems. Sections can be interconnected through track sectioning stations (section insulators), insulated overlapping sections or paralleling stations to connect with other lines [29].

Heeding figure 2-2, it is shown the two main topologies of DC power supplies in railways. Both circuits have a double feeder line and a returning circuit (rails). In this figure, two substations and four sections have been performed. Fig.2-2.a shows a single-end feed diagram where each traction line of a section is connected to an incoming supply. In this example, all the sections are electrically isolated from others. For this scheme the power flow is unidirectional. In Fig2-2.b, a double-end feed is represented. Unlike the previous case, all sections are electrically dependent, being all connected. Both traction lines of section 1 and 2 have been cross-coupled in substation on the left. The same occurs with tractions lines of section 3 and 4 in substation on the right. Besides, a longitudinal coupling is carried out between section 2 and 3 through a sectioning station. In this case, the power flow is bidirectional. The double-end feed results in a more stable voltage profile along the line, being applicable if both substations are powered at the same voltage [29].

2.1.5 AC electrified systems

The alternating current systems are fed through overhead lines which are supplied at higher voltage levels and leads to lower losses. At the beginning of the 20th century, railway electrification systems were developed in AC, in Germany and Switzerland, using 15 kV and 16.7 Hz. This was due to the switching limitations for the variable speed machines fed in AC. Originally, the machines used for 16.7 Hz systems were DC machines fed with AC. The inductance of the windings of the firsts models of big motors make impossible to operate at industrial frequencies. The eddy currents produced in the iron lead to overheating and to reduction on the efficiency [30] [26].

In the AC electrification systems the 25 kV 50/60 Hz is the most common solution used for railway lines and suburban. 50 kV 50/60 Hz characteristics is implemented for heavy-haul railways.

AC power supplies

The electric traction networks of the railways are connected to distribution networks with nominal voltage as high as possible. This is due to the fact that the traction network itself introduces unbalanced, flicker and harmonic loads into the system. In remote areas, the power consumption of a locomotive can represent 1-2% of the short-circuit capacity, which can cause problems for the distribution network. The implementation of on-board PWM converters causes a reduction of the harmonic components and regulates the variations of the power factor. But it is unable to counteract flicker and unbalance problems [30].

In AC traction systems, ground leakage current is considered a major problem. When the current drifts to earth, it is distributed at great depths. This results in a considerable magnetic field between and around the drivers. The two main consequences are [30]:

- The self inductance reaches high values, around 2 mH/km, resulting in increases in voltage drop and reduces the distance between substations.
- This field may cause interference in telecommunication lines and dangerous voltages in fault conditions.

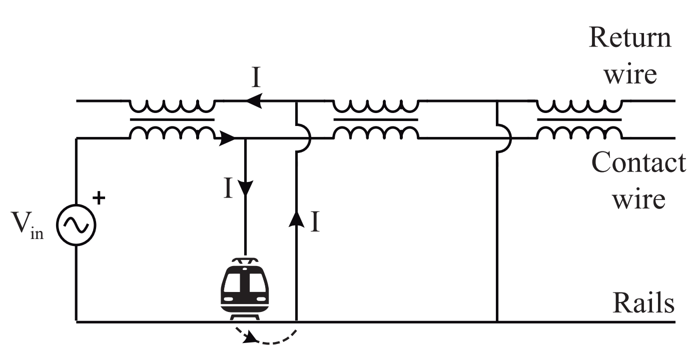

Figure 2-3: 1x25 kV scheme with booster transformer supply

The most common way to reduce the ground leakage current is implementing a return wire connected in series with a booster transformer (see Fig.2-3), where the consecutive transformers can be installed around 3 km.

The autotransformer power supply diagram also known as 2x25 kV, is presented in figure 2-4. The main advantages to use this topology is because of it reduces the dealing current, as well as rail potentials, losses and increase the feeding distance. The voltage between ground feeder (rails) and positive feeder is 25 kV and the voltage between the negative feeder and the ground feeder is -25 kV, leading to 50 kV between positive and negative feeder.

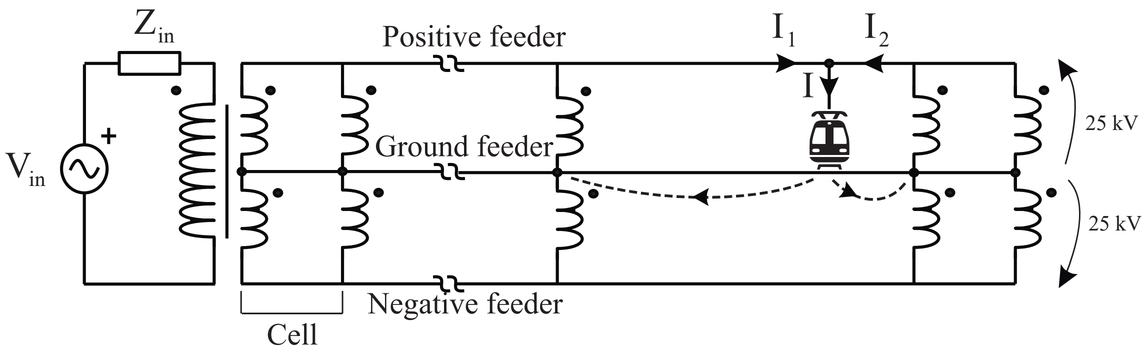

Figure 2-4: Autotransformer power supply scheme or 2x25 kV topology

2.1.6 Energy storage systems for railway applications [5]

Speed and mass are the two main parameters that affect to the energy consumption of rail vehicles. On one hand, speed depends on driving schedule and service requirements. On the other hand, mass is dependent on vehicle structure, number of people transported and devices installed on-board. It is estimated that the energy consumption supposed 30% of the lifetime cost. Thus, the reduction in weight must be one of marked goals when designing rolling stock in order to reduce the energy consumed. The main variables that determine the final weight of the vehicle are:

- Multiple units operation.
- Comfort requirements, either for passenger or driver.

- Power of air conditioning.
- Acceleration levels of the vehicles.
- Amount of passenger seats.
- Number of on-board energy storage systems.

One approach to reduce the energy consumption through the catenary and not wasting electrical energy is to use, this energy developed, during braking by means of ESS. As it is known, the electric motors operate as generators when the vehicle is braking, the kinetic energy is transformed into electric energy, being able to be used in accelerating periods. This leads to an improvement in the global efficiency of the system and a reduction of the flowing currents through catenary.

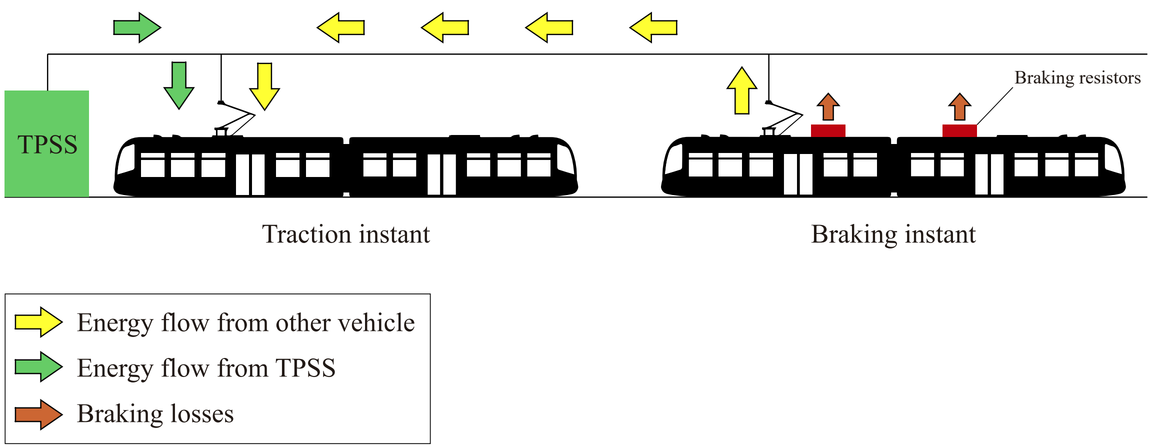

Figure 2-5: Power flow through the systems without on-board ESS

Heeding Fig2-5, it is shown the power flow through the catenary when there is no on-board energy storage systems. In this situation, a percentage of the regenerative power is used for feeding other vehicle, meanwhile other portion is burned into the braking resistors, meaning an unused energy. This braking resistor are implemented in order to avoid over-voltages in the catenary.

In periods with higher power generated than consumed, this surplus is sent to the braking resistors of the vehicles. In order to avoid that, on-board ESS, or trackside energy storage systems can be implemented. When on-board ESS are installed the flowing current is reduced. The ESS provide energy when a vehicle is accelerating (see Fig.2-6). Besides, the efficiency of the system increases, being reduced the resistive losses.

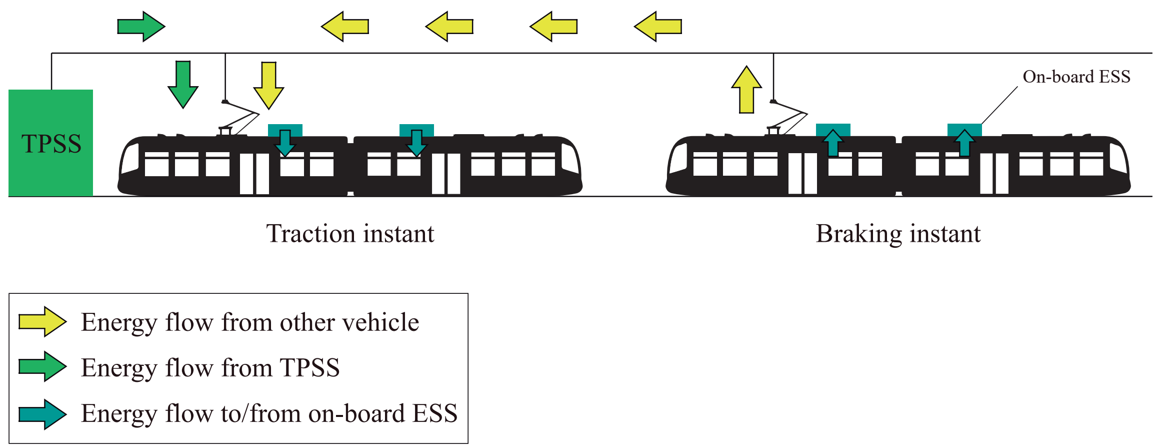

Figure 2-6: Power flow through the systems with on-board ESS

Thus, the main goals of the ESS in railway electrified systems are:

- Recover excess energy in braking process.
- Provide a portion of the energy in accelerating process.
- Substitute the catenary by ESS.
- Cut the peak power through the catenary, decreasing the flowing current and resizing of TPSS.

2.2 RailNEOS 2.0

RailNEOS 2.0 is a web tool simulator developed between CAF Turnkey & Engineering (CAF TE) and the University of Oviedo within the ESTEFI project. This software simulates any railway electrified system, calculating the power flow of the network, and taking into account mix-mode substations and ESS, either on-board or track-side energy storage systems.

This web application calculates the energy consumption of the system from substations point of view, and along the lines, computing the voltage, active power, reactive power, losses profiles...

The results are shown in an interactive web interface that is able to represent all the electric variables, with the purpose of understanding the behavior of the network and permitting to design the most efficient topology of the grid.

RailNEOS 2.0 is a tool that can be used for analyzing the planning operations and the most convenient schedules for the vehicles. In addition, it can serve as support tool to reduce the operational cost, by means of evaluating the implementation of ESS, on-board or off-board and their size.

The results will be exported in a database that will be the data input of the thermal calculations.

Chapter 3

Thermal model of overhead lines

In order to evaluate the operation of a conductor submitted to voltage and circulating current through it, a thermal model will be established, highlighting the most remarkable parts. In this model, the main stages of heat transmission will be specified, from its longitudinal axis to external edges. The conductors under study will be bared, so the insulation layers will not be mentioned.

As consequence of the current flowing through the conductor, losses arise due to Joule effect. A flow heat will take place from the wire to the atmosphere. A gradient of temperature will appear, being the main limiting factor of the maximum current without not damaging the components.

Figure 3-1 represents the thermal model of an overhead line. A section of the common catenary is presented, specifying the direction of the heat transmission. The relative aspects of the conductor and the variations in the climatological conditions have been taken into account to develop this model.

The heat gain is the sum of the heat produced by the electric current flowing through the wire, Joule heating (Q_j) and the solar radiation heating (Q_s).

$$U_{tr}(t) = Q_j + Q_s \tag{3.1}$$

The thermal resistance depicts the capacity of the conductor to dissipate heat through phenomenons of convection (Q_c), evaporation (Q_w) and radiation (Q_r). The cooling

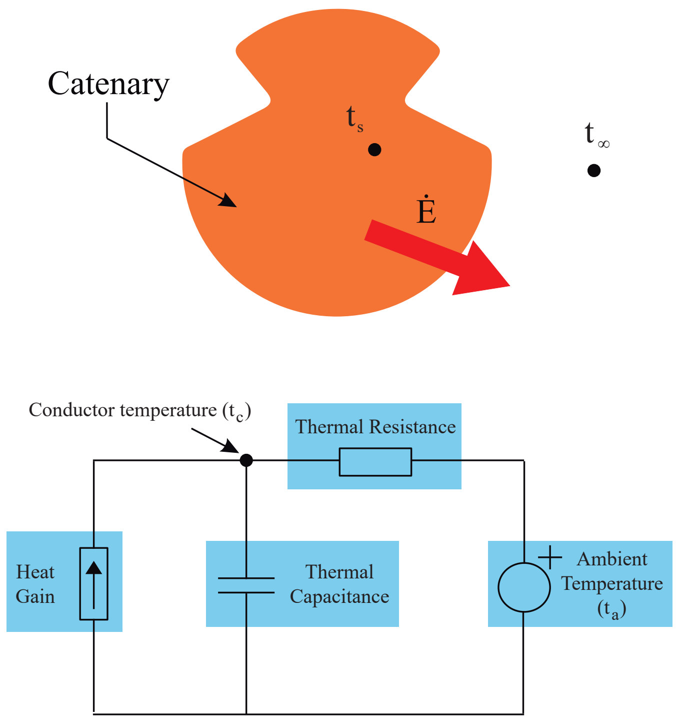

Figure 3-1: Thermal model of an overhead line

factor gathers these effects (see Eq. 3.2). For the thermal calculations that are going to be carried out in Matlab, the evaporation factor has been considered null.

$$K_{cool}(t) = \frac{Q_c + Q_w + Q_r}{T_s - T_\infty} \tag{3.2}$$

Where:

- T_s: conductor surface temperature, $[°C]$
- T_∞: air temperature, $[°C]$

The thermal capacitance (C_{Tr}) is a factor dependent on conductor properties and weather conditions. As can be seen in Eq. 3.3, C_{Tr} is function of the conductor type and of climatological aspects that affects the external edge of the wire. As consequence of the thermal capacitance, a change in the flowing current leads to a gradual variation of the conductor temperature [36].

$$C_{Tr}(t) = m \cdot C_p(t) \tag{3.3}$$

3.1 Electric losses

The electric losses along the feeder causes a heat flow, which must be liberated in order to not perturb the properties of the components.

3.1.1 Joule's losses

As was commented previously the losses in the conductor are because of the Joule effect. The electric energy in a conductor is the consequence of the kinetic energy of the electrons, which are driven by a electric field when the conductor is connected to voltage. However, not all this kinetic energy is used, since the shocks and friction between electrons and ions of the crystal lattice of the conductor exist. The losses due to Joule effect are represented in Eq.3.4

$$Q_c = R \cdot I^2 \tag{3.4}$$

Where:

- R: electric resistance of the conductor, [Ω]
- I: RMS current which flows through the wire, [A]

The electric resistance of a wire shows the non-linearity of the conductor against the flowing current. The value of this resistance is at the nominal current of the conductor. The resistance is given by the Eq.3.5

$$R = \rho_T \cdot \frac{l}{S} \tag{3.5}$$

Where:

- ρ_T: resistivity of the conductor, [$\Omega \cdot m$]

- l: conductor length, [m]

- S: section of the wire, $[m^2]$

Joule's losses are caused by the non-linearity of the material. But another type of losses could appear in the case AC current was flowing through the conductor.

3.1.2 Skin effect

The skin effect is related to the frequency of the AC current. With the increasing of the frequency, the effective section of the wire is reduced, boosting the resistance of the wire. The variations of the magnetic flow induce that the distribution current was not uniform. A higher voltage is created insight the wire so current density decreases, in the external part of the conductor is the opposite. Hence, the density current increase gradually from the center of the conductor due to the voltage induced blocks the variations of the current (see Fig. 3-2) [17].

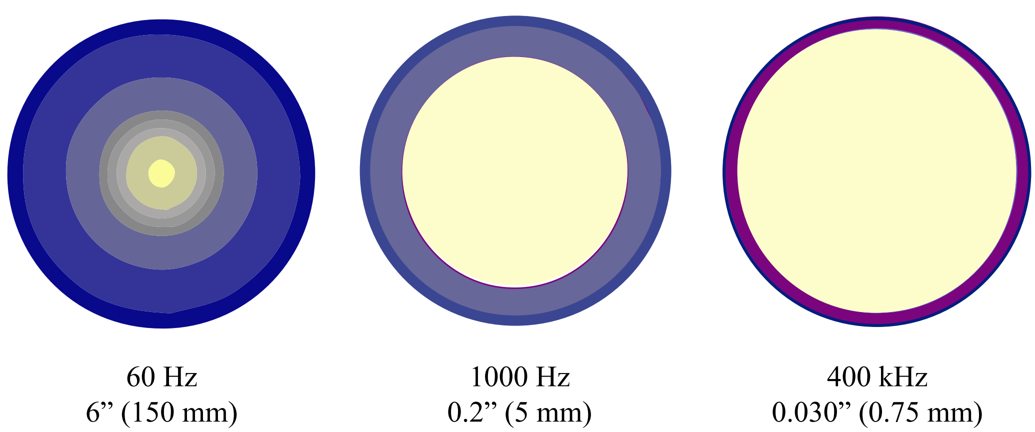

Figure 3-2: Skin effect of conductor at different frequencies. Blues tones represent the currents

3.1.3 Proximity effect

The proximity effect takes place when two or more conductors are relatively close, as well as AC current is flowing through them. In both conductors, magnetics effects appear modifying the distribution of the current density in the transversal section. As result, the effective area of conduction is decreased leading to a higher resistance [21].

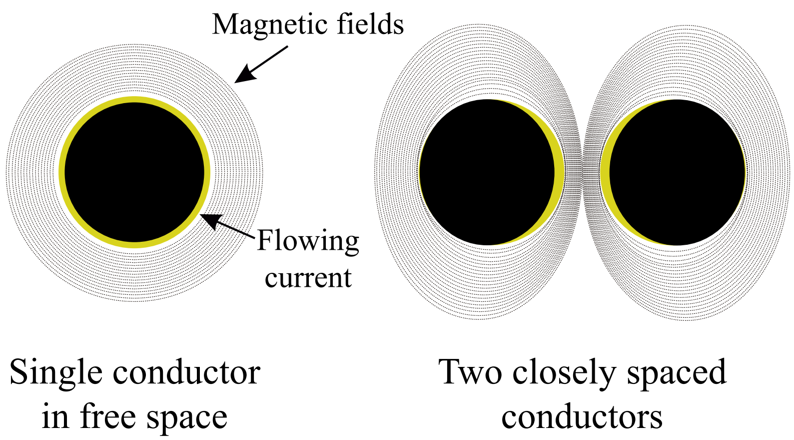

Figure 3-3: Single conductor in free space and two conductors closely spaced, appearing the proximity effect

3.1.4 Dielectric losses

When a wire is in charge a electric field is generated, consequently leaks appears in the isolation, in form of heat. For AC system with shield cables, dielectric losses takes importance due to the current has a new path to circulate. In DC conductors and in AC wires with low voltage, this effect is neglected.

3.1.5 Electrical Resistance

All the effects previously mentioned are gathered in an equivalent electric resistance. These figures are tabulated according to type of the conductor. For the thermal study, the data input will be the Joule's losses, provided by RailNeos.

Chapter 4

Heat Transfer

Heat transmissions occur by means of interactions between a systems and its environment, when a difference of temperature exists, producing heat and work. The thermodynamical analysis allows to know the final state of a process, but it is important to know the origin of this process and the temporary evolution. Below, the main three heat transmissions mechanism will be described.

4.1 Conduction

The heat transmission, by means of conduction, is produced because of the vibration in atoms and particles. The highest temperatures are related to high molecular energy where the areas with higher energy transmit it to zones less energetic. The amount of energy transferred per time is quantified by Fourier's law (see Eq. 4.1). The sign minus indicates that heat flow goes in the direction of decreasing temperature [11].

$$q_c = -S \cdot k \cdot \frac{dT}{dx} \tag{4.1}$$

Where:

- q_c: heat flow for conduction, in a specific direction, $[W]$
- S: perpendicular section to the heat transmission direction, $[m^2]$
- k: thermal conductivity of the material, $[W/(m \cdot K)]$

4.2 Convection

Convection is a specific process of heat transmission where the transfer procedure is made by the displacement and mix of fluids (liquid or gas) at different temperatures. Convection is based on two phenomenons complemented between both. On one hand, due to the random molecular movement of the particles. And on the other hand, thanks to the macroscopic displacement of a fluid in presence of temperature gradient [11].

The movement of fluids caused by densities differences due to temperature gradients, is denominated natural convection. In case of this movement was originated by external forces (gravity, pumps, compressors...), is called forced convection.

The efficacy of the heat transmission by convection depends on the movement of the fluid mix, hence, it is important to know the characteristics of the fluid. The mathematical analysis constitutes a complex field of applied mathematics. In some cases, a good empirical knowledge is important. The Newton's law of cooling is the most common expression to describe this phenomenon (see Eq. 4.2)

$$q_{cv} = S \cdot h \cdot (T_s - T_\infty) \tag{4.2}$$

Where:

- q_{cv}: heat flow by convection, $[W]$.
- S: contact section between body and fluid, $[m^2]$
- h: coefficient of heat transmission by convection, also called film coefficient, $[W/(m^2K)]$
- T_s: temperature of the solid surface, $[K]$
- T_∞: temperature of the global fluid, $[K]$

4.3 Radiation

The radiation is a process in which the heat flows from a body at high temperatures to a body at low temperatures, when both are separated for a space, even, it could be the vacuum. In fact, the heat transmission does not suffer attenuation in vacuum [11].

The term radiation, from the thermal analysis point of view, is applied to phenomenons in which is established an energy transport through transparent means or the vacuum. The energy transmitted is denominated radiant heat or thermal radiation.

The thermal radiation is emitted by a body in form of electromagnetic waves (or photons) as results of the changes in the electronics configurations. The displacement of the radiant heat is similar to the propagation of light in the space and it can be described by the wave theory. When a thermal radiation falls upon a body, a part can be reflected by its surface, part transmitted through it (if it is diathermic) and the rest absorbed by the body, becoming internal energy of it, except in cases where it is induced photochemical or nuclear reactions [11].

The expression that represents the net radiation, either emitted or received, it could be quantified for the case of interchanging between two gray surfaces where the size of one could be neglected against the other (see Eq. 4.3)

$$q_r = S \cdot \varepsilon \cdot \sigma \cdot (T_s^4 - T_\infty^4) \tag{4.3}$$

Where:

- q_r: heat flow by radiation, $[W]$
- S: contact section of the body, $[m^2]$
- ε: emissivity of the material, dimensionless
- σ: Stefan-Boltzmann coefficient, $5.67 \cdot 10^{-8} W/(m^2 \cdot K^4)$
- T_s: temperature of the solid surface, $[K]$
- T_∞: temperature of the global fluid, $[K]$

In this case, the heat is transmitted from the surface to the ambient. The emissivity ($0 \leq \varepsilon \leq 1$) describes the non linearity of the real bodies in comparison with black bodies, which is able to radiate the maximum energy as possible.

4.4 Energy conservation

In the heat transmissions process, it is necessary to recognize the heat transfer mechanisms and to determine if the process is stable or not. When the heat flow is constant with time, the temperature of each point does not change and the conditions of stationary state prevail. The heat flow in every point of the system must be equal to the input flow heat and any change of the internal energy cannot take place.

The heat flow is transitory, instable or not stationary in a system, when the temperatures in several points change with time. A variation on the temperature means a change in the internal energy. Therefore, a portion of the energy is stored and the other constitutes an instable heat flow. These problems becomes more complex than in stationary conditions.

The knowledge of heat transfer and thermodynamic are joined in order to solve thermal problems. The method used consists of selecting a control volume, in which matter and energy exchange can be given. The expression of the energy conservation applied to a control volume must be fulfilled for every instant, i.e, a balance in the energy speeds has to exist (see Eq. 4.4) [11].

$$\underbrace{\frac{dE_{input}}{dt}}_{\dot{E}_{input}} - \underbrace{\frac{dE_{output}}{dt}}_{\dot{E}_{output}} + \underbrace{\frac{dE_{gen}}{dt}}_{\dot{E}_{gen}} = \underbrace{\frac{dE_{stored}}{dt}}_{\dot{E}_{stored}} \tag{4.4}$$

$$\dot{E}_{input} - \dot{E}_{output} + \dot{E}_{gen} = \Delta\dot{E}_{stored} \tag{4.5}$$

Eq.4.5 can be applied to any instant of time and also to an interval of time. Considering an interval of time, the amount of mechanical and thermal energy that enters in the control volume or the amount of thermal energy generated insight, increase the energy stored.

The terms $\dot{E}_{input}$ and $\dot{E}_{output}$ are surface phenomenons, due to they are associated to procedures that, exclusively, take place in surface control and their speeds are proportional to the surface area. Conduction, convection and radiation are examples of theses surface procedures.

The generation term comes from the conversion of electric to thermal energy following the principle of Joule.

Chapter 5

Temperature calculation method

The thermal study applied is based on the method presented in the standard IEEE Std 738-2012 [4]. In [4], the main studies of calculating heat transfer and heat balance for bare overhead transmission line conductors are presented. Below, some of them are enumerated:

- House and Tuttle [12]
- House and Tuttle, as modified by East Central Area Reliability (ECAR) [2]
- Mussen, G. A. [28]
- Pennsylvania-New Jersey-Maryland Interconnection [13] [3]
- Schurig and Frick [31]
- Davis [14]: the heat balance expression is presented as a bi-quadratic equation which is able to calculate the temperature of the conductor directly.
- Morgan [25]
- Black, Bush, Rehberg, and Byrd [7] [9] [8]: the radiation part is linearized and the heat balance expression is calculated by means of linear differential equations.

- Foss, Lin, and Fernandez [15]: similar to previous method but allows a faster calculation, reducing the iterations with a more precise linearized radiation part.

- CIGRE Technical Brochure 207 [6]

The thermal calculation method implemented is the House and Tuttle [12] with some variations from ECAR [2]. In this study, it is considered that at high density currents the conductors could not be isothermal, therefore the variations of radial temperature must be computed. The equations below allow to calculate the variations of wire temperature knowing the current. Also, they could provide the thermal current rating that leads to the maximum allowable conductor temperature.

For an accurate calculation method and with the purpose to reduce the error incurred as far as possible, it is important to know the factors that have influence in the final result. For the thermal problem, the variables that affect the conductor temperature are the following:

- Conductor material characteristics, being the electrical conductivity the most important property.

- Diameter of the conductor.

- Conductor surface properties, i.e, emissivity and absorptivity.

- Climatological states, such as wind speed and direction, solar heating and air temperature.

- Electrical current flowing through the conductor.

In this thesis, the case under study is a dynamic problem where the input current is changing over the time. Regarding climatological inputs, wind speed and heat solar gain are taken into account, considering both constants along the simulation. Nevertheless, House and Tuttle as modified by ECAR is able to calculated other type of cases as:

- Steady-State Case: the current, conductor temperature and the climatological conditions are considered constant in all the simulation

- Transient Case: the current suffers an step change and the weather variables remain constant. The temperature describes a wave similar to an exponential curve that starts with the variation of the current.

- Dynamic Case: the electrical current and the weather conditions are varying over the time. The temperature of the conductor is calculated for each period of time in which the current and the weather conditions hold constant.

5.1 Temperature calculations varying weather and current

The temperature of the conductor is a variable dependent on electrical current and climatological conditions, where the temperature is calculated for each period of time. The simulations have an interval of time equal to 1 second in which the weather variables (air temperature, wind speed, direction ...) and current remain constant.

The increase or decrease in conductor temperature is calculated for each interval of time by means of the heat balance expression which is presented in Eq.5.1 and Eq.5.2. The temperature of the previous period is updated with the temperature change.

$$q_c + q_r + m \cdot C_p \cdot \frac{dT_{avg}}{dt} = q_s + \underbrace{I^2 \cdot R(T_{avg})}_{P_{losses}} \tag{5.1}$$

$$\frac{dT_{avg}}{dt} = \frac{1}{m \cdot C_p} \left[P_{losses} + q_s - q_c - q_r\right] \tag{5.2}$$

Where:

- q_c: heat convective loss, $[W/m]$
- q_r: radiated heat loss, $[W/m]$
- q_s: solar heat gain, $[W/m]$
- m: mass of the conductor, $[kg]$

- C_p: specific heat per length $[J/(Kg \cdot^{\circ} C)]$

5.2 Convective heat loss

The convective procedure can be split into two groups, natural and forced convection. Natural convection happens when air surrounding the conductor is heated and this mass of air is moved gradually. Forced convection appears when a flow of air is falling upon the conductor and the mass of air is moved. Natural convection is equivalent to forced convection for speeds lower than 0.2 m/s. The equation presented by McAdams [23] for the calculation of convective heat loss for cylinders will be implemented in this mathematical model.

5.2.1 Forced convection

In order to distinguish laminar and turbulent flow, House and Tuttle [12] uses two expressions for forced convection. The changing of states is carried out at a Reynolds number equals to 1000 (value calculated as convenience following conductor ampacities). This made that the transition between both states was a discontinuity, instead of a curve as in reality. To avoid the discontinuity, Eq. 5.3 and Eq. 5.4 are used. The crosspoint of both curves indicates the transition from laminar to turbulent air flow. Eq. 5.3 is used for calculating the forced convection at low wind speed and Eq. 5.4 is employed for forced convection at high wind speed. What is recommended in standard [4] is to obtain the convective heat loss by using both equations, and getting the highest value. The upper limit validity of this equations is at Reynolds number equal to 50000.

$$q_{c1} = K_{angle} \left[1.01 + 1.35 \cdot {N_{Re}}^{0.52}\right] \cdot k_f \cdot (T_s - T_\infty),\ [W/m] \tag{5.3}$$

$$q_{c2} = K_{angle} \cdot 0.754 \cdot {N_{Re}}^{0.6} \cdot k_f \cdot (T_s - T_\infty),\ [W/m] \tag{5.4}$$

Where:

- k_f: coefficient of thermal conductivity of air.

- K_{angle}: wind direction factor. Its expression is shown in Eq. 5.5, where ϕ is the angle between the conductor and the wind direction

$$K_{angle} = 1.194 - cos(\phi) + 0.194 \cdot cos(2\phi) + 0.368 \cdot sin(2\phi) \tag{5.5}$$

The expression to calculate the dimensionless Reynolds number is shown in Eq. 5.6.

$$N_{Re} = \frac{D_0 \cdot \rho_f \cdot V_w}{\mu_f} \tag{5.6}$$

Where:

- D_0: conductor diameter, $[m]$

- ρ_f: air density, $[kg/m^3]$

- V_w: air speed, $[m/s]$

- μ_f: dynamic viscosity of the air, $[Kg/(m \cdot s)]$

The range of application of forced convection equations is shown in Tab. 5.1. This valid range is widely higher than the design range of operation.

Table 5.1: Valid range for forced convection equations [1]

Variable	**SI units**
Diameter	0.01-150 mm
Air speed	0-18.9 m/s
Air temperature	15.6-260 $°C$
Wire temperature	21-1004 $°C$
Air pressure	40.5-405 kPa

5.2.2 Natural convection

As was commented previously, the natural convection is performed at wind speeds close to zero. In Eq. 5.7 the expression of the natural convection heat loss is presented.

$$q_{cn} = 3.645 \cdot {\rho_f}^{0.5} \cdot {D_0}^{0.75} \cdot (T_s - T_\infty)^{1.25}, \ [W/m] \tag{5.7}$$

For low wind speeds, Mc Adams [23] recommends obtaining natural and forced convection heat loss, selecting the most restrictive result. This approach is the one implemented in Matlab.

5.3 Radiated heat loss

The thermal losses goes from high temperatures to low temperatures, in this case, the energy is transferred from the conductor to the surrounding air. Eq. 5.8 and Eq. 4.3 represent both the same. For the thermal study of overhead conductors, Eq. 5.8 is used.

$$q_r = 17.8 \cdot D_0 \cdot \varepsilon \left[\left(\frac{T_s + 273}{100} \right)^4 - \left(\frac{T_\infty + 273}{100} \right)^4 \right], [W/m] \tag{5.8}$$

5.4 Solar heat gain

The solar heat received for the conductor is calculated with Eq. 5.9. The heat provided to the conductor leads to an increase on the temperature. The amount of heat absorbed depend on the properties of the material. For instance, bright conductors reflect a high amounts of energy, in contrast with black bodies which absorb big amounts of energy. The solar heat depends on the solar absorptivity (α), the total solar and sky radiated heat intensity corrected for elevation (Q_{se}), effective angle of incidence of the sun's rays (θ) and the projected area of conductor (A') in m^2/m, (see Eq. 5.9).

$$q_s = \alpha \cdot Q_{se} \cdot sin(\theta) \cdot A', [W/m] \tag{5.9}$$

The effective incidence angle of the sun's rays is introduced in Eq. 5.10.

$$\theta = arccos\left[cos(H_c) \cdot cos(Z_c - Z_l)\right] \tag{5.10}$$

Where:

- H_c: altitude of sun (0 to 90), $[deg]$
- Z_c: azimuth of sun, $[deg]$
- Z_l: azimuth of line, $[deg]$

5.5 Conductor heat capacity

Conductor heat capacitance results from the product of mass and specific heat per length. In case that several types of material conform the conductor, the total conductor heat capacitance is obtained as sum of the core and the outer strands as it is shown in Eq. 5.11.

$$m \cdot C_p = \sum m_i \cdot C_{pi} \tag{5.11}$$

The values of the specific heat for typical conductors wires are introduced in Tab. 5.2

Table 5.2: Specific heat of typical conductor metal wire

Material	$C_p[J/(kg \cdot^\circ C)]$
Aluminum	955
Copper	423
Steel	476
Aluminum-clad steel*	534*

* *The heat of aluminum-clad steel depends on the ratio between both. This is a typical value with a conductivity of 20.3 % I.A.C.S*

5.6 Air characteristics, solar angle and solar heat flux

For Natural and Forced heat convection loss, the coefficient of thermal conductivity of air (k_f), air density (ρ_f) and air viscosity (μ_f) are calculated at the average temperature of the boundary layer (T_{film}), see Eq. 5.12.

$$T_{film} = \frac{T_s + T_\infty}{2} \tag{5.12}$$

5.6.1 Dynamic viscosity of air

Dynamic viscosity is a measure to compute the internal resistance between the molecules of a fluid. It relates the stress or local tension in a fluid in motion with the speed of deformation of the fluid particles [22]. For the calculation of dynamic viscosity of air, Eq. 5.13 is used.

$$\mu_f = \frac{1.458 \cdot 10^{-6} \cdot (T_{film} + 273)^{1.5}}{T_{film} + 383.4}, [kg/(m \cdot s)] \ or \ [(N \cdot s)/m^2] \tag{5.13}$$

5.6.2 Air density

The air density depends on the elevation of conductor above sea level (H_e) and air temperature of the external layer of the conductor (T_{film}). Eq. 5.14 represents the mathematical expression of the air density.

$$\rho_f = \frac{1.293 - 1.525 \cdot 10^{-4} \cdot H_e + 6.379 \cdot 10^{-9} \cdot {H_e}^2}{1 + 0.00367 \cdot T_{film}}, [kg/m^3] \tag{5.14}$$

5.6.3 Thermal conductivity of air

The thermal conductivity is a physical property of the materials which measures the capacity of heat conduction, i.e, the capacity of transferring kinetic energy from theirs molecules to others adjacent or to substances in contact [20]. The thermal conductivity of air is calculated taken into account T_{film} (see Eq. 5.15)

$$k_f = 2.424 \cdot 10^{-2} + 7.477 \cdot 10^{-5} \cdot T_{film} - 4.407 \cdot 10^{-9} \cdot {T_{film}}^2, [W/(m \cdot ^\circ C)] \tag{5.15}$$

5.6.4 Altitude of the sun

The expression of the solar altitude of the sun (H_c) in degrees or radians is presented in Eq. 5.16. The solar altitude is the angle between the sun's rays and the Earth's horizon (see Fig. 5-2). The Earth is tilted 23.45 ° in relation of the solar system plane. The value of H_c, changes depending on the time of the day, time of the year and latitude of the location. Those places that are close to the equator have higher solar altitude than regions near to the poles [27].

For obtaining H_c, the latitude of the place, hour angle (ω) and solar declination (δ) must be calculated.

$$H_c = arcsin\left[cos(Lat) \cdot cos(\delta) \cdot cos(\omega) + sin(Lat) \cdot sin(\delta)\right] \quad (5.16)$$

The hour angle (ω) is the angle considering 0 degrees at noon time and taking 15 degrees for each hour. For instance, 1PM is + $15°C$ (see Fig. 5-1).

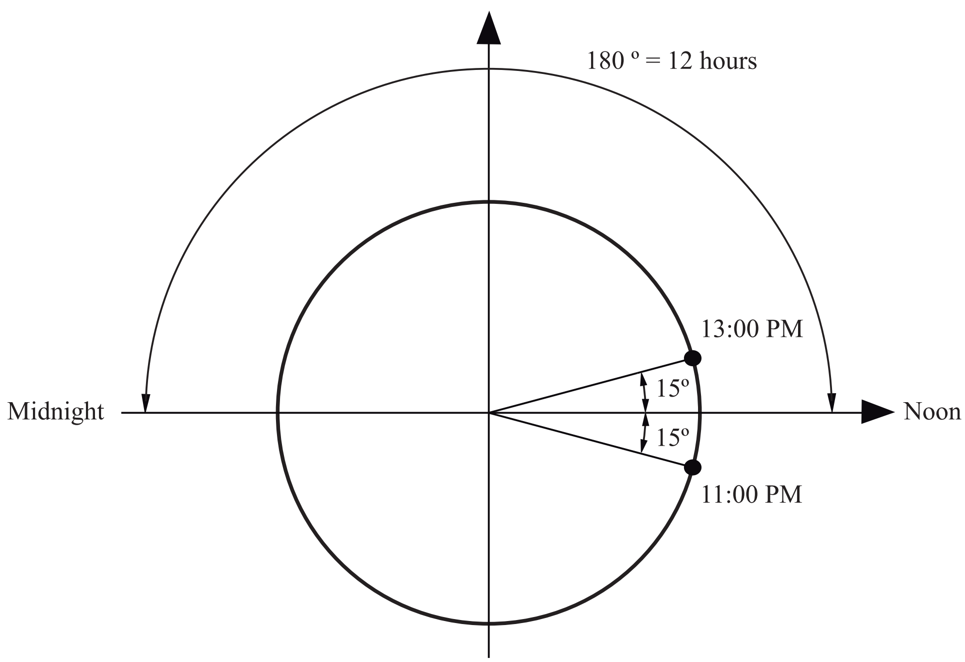

Figure 5-1: Representation of the hour angle (ω) during the day

The solar declination (δ) is the angle between the line Sun-Earth and the plane of Earth's equator (see Fig. 5-2). The expression of solar declination in degrees is shown in Eq. 5.17. This equation is valid, either for northern hemisphere (Lat higher than 0) and for southern latitudes (Lat lower than 0).

$$\delta = 23.46 \cdot sin\left[\frac{284 + N}{365} \cdot 360\right] \quad (5.17)$$

Where N is the day of the year, for instance, January 8 is equal to 8. The solstices are on 172 and 355, days in which the sun reaches the maximum north declination (+23.45°) or south (-23.45 °) with respect to equator.

5.6.5 Azimuth of the sun

The azimuth of the sun (Z_c) is the angle measured between the cardinal North of the Earth and the projection on the horizon of the celestial body which is being observed (see Fig. 5-2). The angle is always measured in a clockwise direction. In Eq. 5.18 is shown the expression of the azimuth in degrees.

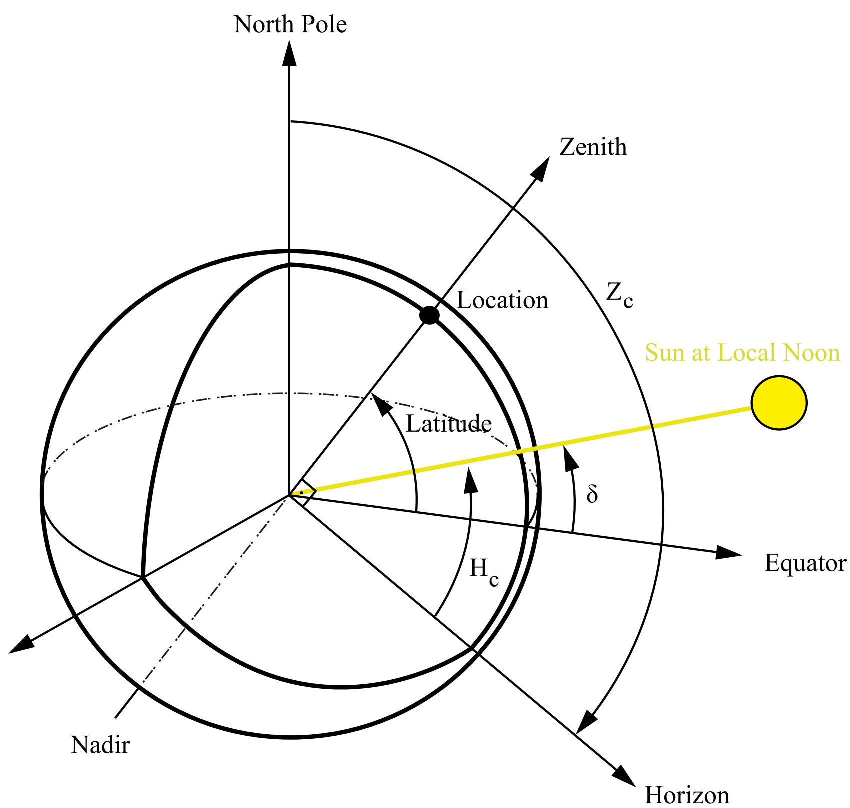

Figure 5-2: Representation of the main parameters for calculating solar heat gain

$$Z_c = C + arctan(\chi) \tag{5.18}$$

The solar azimuth variable is presented in Eq. 5.19.

$$\chi = \frac{sin(\omega)}{sin(Lat) \cdot cos(\omega) - cos(Lat) \cdot tan(\delta)} \tag{5.19}$$

Where C is the solar azimuth constant which is dependent on the hour angle and the solar azimuth variable. In the table 5.3, the solar azimuth constant values are presented.

Table 5.3: Values of solar azimuth constant (C) in degrees, taken into account hour angle (ω) and solar azimuth variable (χ)

ω in degrees	**C if $\chi \geq 0$ degrees**	**C if $\chi < 0$ degrees**
$-180 \leq \omega < 0$	0	180
$0 \leq \omega < 180$	180	360

5.6.6 Total heat flux density

The expression of the total heat flux density (Q_s) is introduced in Eq. 5.20. In this expression the solar altitude in degrees (H_c) and coefficients of the cleanliness atmosphere are used.

$$Q_s = A + B \cdot H_c + C \cdot {H_c}^2 + D \cdot {H_c}^3 + E \cdot {H_c}^4 + F \cdot {H_c}^5 + G \cdot {H_c}^6, [W/m^2] \quad (5.20)$$

In table 5.4 the values of the polynomial coefficients for a clear and an industrial atmosphere as function of H_c are gathered.

5.6.7 Elevation correction factor

The total solar and sky radiated heat intensity corrected for elevation (Q_{se}) is calculated with Eq. 5.21.

$$Q_{se} = K_{solar} \cdot Q_s, [W/m^2] \quad (5.21)$$

Where the solar altitude correction factor (K_{solar}) is obtained following Eq. 5.22

$$K_{solar} = A + B \cdot H_e + C \cdot {H_e}^2 \quad (5.22)$$

The value of the coefficients used in Eq. 5.22 are shown in Tab. 5.5

Table 5.4: Value of coefficients for the total heat flux depending on the state of the atmosphere

	SI
Clear atmosphere	
A	–42.2391
B	63.8044
C	–1.9220
D	3.46921×10^{-2}
E	-3.61118×10^{-4}
F	1.94318×10^{-6}
G	-4.07608×10^{-9}
Industrial atmosphere	
A	53.1821
B	14.211
C	6.6138×10^{-1}
D	-3.1658×10^{-2}
E	5.4654×10^{-4}
F	-4.3446×10^{-6}
G	1.3236×10^{-8}

Table 5.5: Value of coefficients of Eq. 5.22

	SI
A	1
B	$1.148\text{x}10^{-4}$
C	$-1.108\text{x}10^{-8}$

5.7 Emissivity and absorptivity of the conductor

Emissivity (ε) and absorptivity (α) are variables interrelated, both increasing with time, atmospheric pollution and line operating voltage. The researches [18] and [32] define ε and α in the range of 0.2 and 0.3 for new conductors. Depending on the line voltage and the pollution of the air these factor could increase, reaching 0.7 over the time. Studies from EPRI [4], conclude that values of young conductors are in the range of 0.2 and 0.4, in some occasions this band could grow up to 0.5 and 0.9, depending on the line voltage operation and the amount of particles in the atmosphere.

The selection of emissivity and absorptivity has different effect depending on the operation conductor temperature.

- For temperatures lower than 75 °C, the value of α is relevant due to the importance of heat solar gain
- For temperatures higher than 150 °C, the ε has an important impact because of the radiated heat loss.
- For the rest of temperatures range, the values used for ε and α does not have a relevant impact on the temperatures calculation.

Chapter 6

Thermal calculation tool: RailThermal app

RailThermal app is a thermal simulator, developed in the present thesis that uses Matlab as a calculation and representation engine. This tool has been designed to study the temperature of railway electrified systems, allowing to size the catenary conductors. For the input data, RailThermal app can import data from any railway system and configuration, simulated with the software RailNeos 2.0.

The amount of raw data that will be managed is high, therefore it is necessary methods to analyze this data through an easy way. It results very difficult to study and reach conclusions from rows and columns of data. Thus, a visual representation is important due to it works as a quick means of communication. For that reason three methods of representation are implemented, allowing users to analyze and query data interactively.

This chapter has as purpose to serve as guide for the new user, describing the functionalities of the program and the steps to follow for a correct importation and visualization of the data. In the chapter 7, we will go in deep in the code tools employed for its development.

6.1 Application

A standalone application has been created, with the purpose that anyone who does not have a Matlab's license can use RailThermal app. In Fig. 6-1 and Fig. 6-2, the cover and icon of the application are presented.

Figure 6-1: Cover of RailThermal app

Figure 6-2: Icon of RailThermal app

6.2 User interface

The user interface consists of six blocks where the initial conditions of the simulation must be specified. In Fig. 6-3, it is presented the interface window. Below, the parts of RailThermal app interface will be detailed:

- **Initial Temperature**: the initial temperature of the conductor and the air must be specified in order to take them as reference value for the process of iterative calculations.
- **Location Conditions**: the elevation of the location and speed of the wind are necessary to carried out the simulation.
- **Inputs for Solar Heat Gain Calculation**: the date of the simulation and the latitude of the location are important factors that affect to the solar heat gain. With the purpose to search for the latitude of the place under study, a button linked to a web page (https://www.latlong.net/) is set. Also, a check box is implemented for enabling or disabling calculations of solar heat gain.
- **Type of conductor**: this block allows to select the material and section of the conductor through drop down lists.
- **Sectioning of the lines**: this parameter define the length of sections, i.e, the parts in which the line will be split. This is an configurable value that can be adapted depending on the characteristics of the network studied. Furthermore, it is a variable that changes directly the time of simulation.
- **Select range of temperatures to plot**: in this box, it is possible to select those interval of temperatures that will be plotted.
- **State of the simulation**: this field shows in which point is the simulation.
- **Run button**: button to start the simulation.
- **Exit button**: button to close the interface.

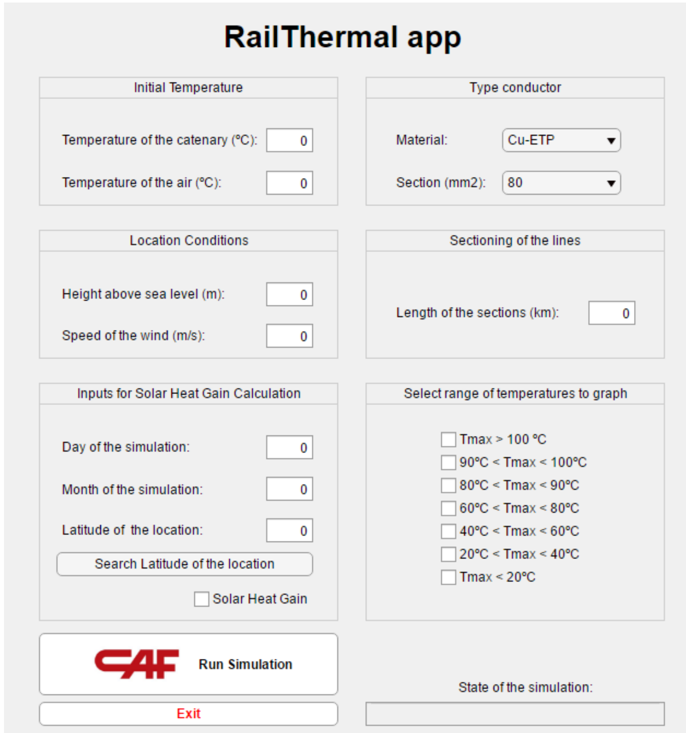

Figure 6-3: User interface of RailThermal app

6.3 Import database

The operation of RailThermal app consists of uploading the file res.db exported by the RailNeos 2.0. This file is a database where characteristics of the systems and the results of the power flow are gathered.

To start the simulation, the run button must be clicked. The file browser window opens and the case study is selected. It is advisable to have the database file in a folder with the name of the simulation, since the name of the folder will be used for the thermal map header.

6.4 Exported graphics

Once the simulation has finished, a message indicating that the simulation has been completed successfully will appear. RailThermal app will show different graphs to analyze the behavior of the network. Each graph studies the temperature of the system from a different point of view.

6.4.1 Thermal Map

In this graph, the grid under study will be drawn, indicating all nodes and lines of the system. The sections will be represented as points along the lines. These points will be distinguished by colors. Each of these colors refers to a temperature range. In the legend, the limit values of each band are shown. Nodes with temperatures greater than 100 °C appear highlighted with a radius higher than the rest (see Fig. 6-4). Using, data cursor tool, it is possible to select the points that compose the grid. In Fig. 6-5, it is shown the messages when a node is chosen. The index, maximum temperature reached along the simulation and minutes with temperature higher than 120 °C are shown.

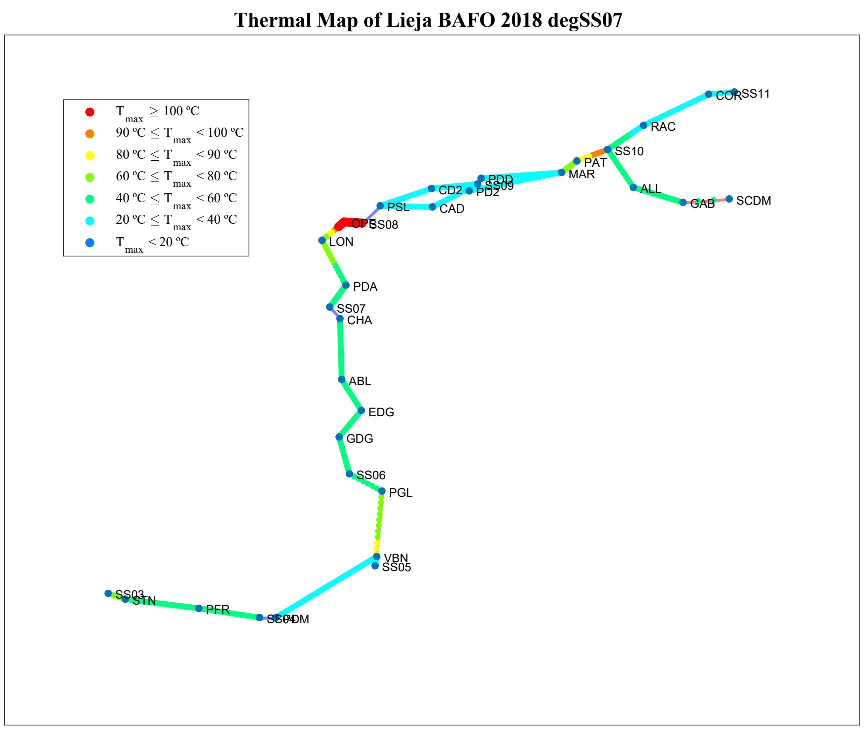

Figure 6-4: Thermal Map of the Lieja's project

6.4.2 Average temperature of the lines

This graph is focused on relating the temperature of the lines with respect the distance in km (see Fig. 6-6). The temperature is shown against the kilometric point of

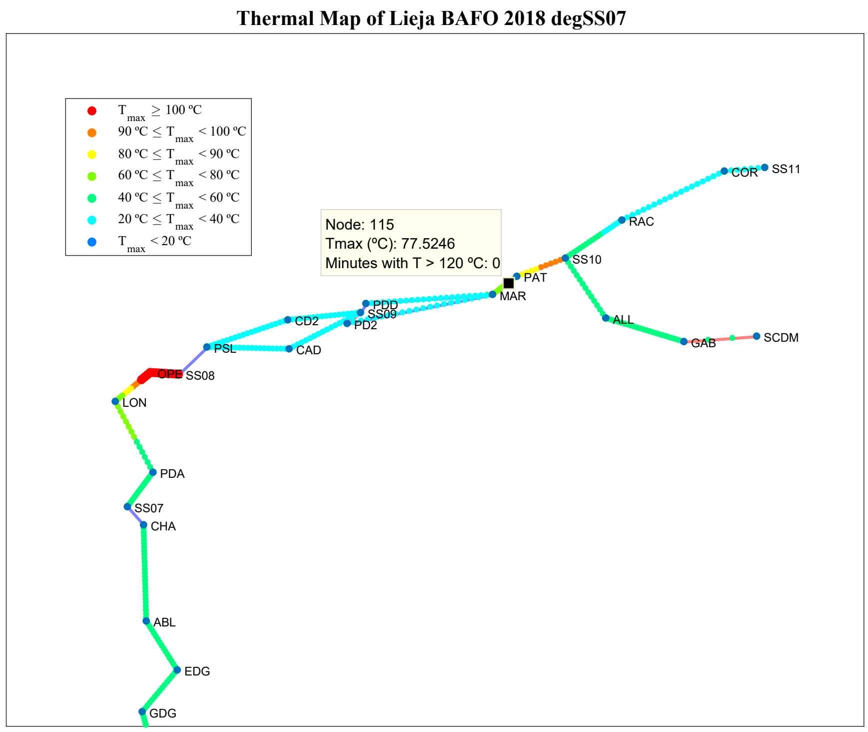

Figure 6-5: Thermal Map of the Lieja's project, using data cursor tool

the line, allowing to visualize which points are more affected by high temperatures. This plot is automatically generated. In the title, the index of the lines, source and destination node are indicated.

6.4.3 Evolution of the temperature over time

In this type of representation, temperature and time are related. As can be seen in Fig. 6-7, the evolution of temperature in a line is plotted. Commonly, the wave of this graph follows a first order system, going from the initial temperature supposed, to the final temperature reached in steady state. The header of the graph indicates which line and which kilometric point is being shown. As it happened in the previous type of graph, the evolution of temperature respect time will only be shown, for those ranges selected in the interface.

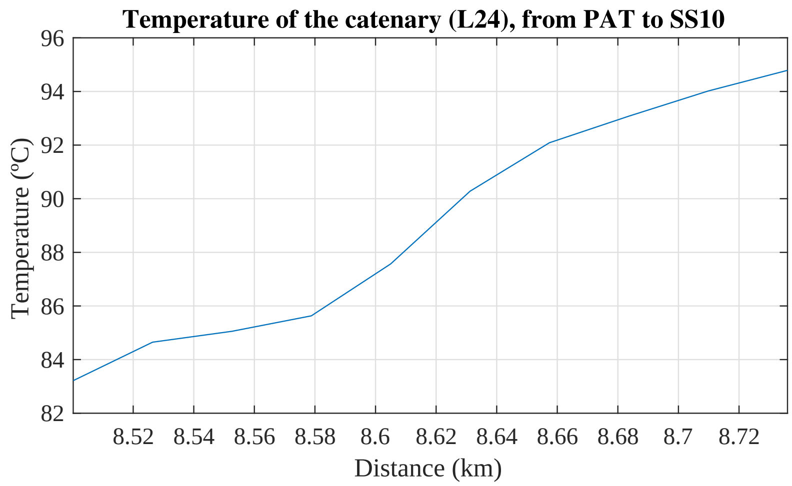

Figure 6-6: Average temperature of line 24 during the simulation

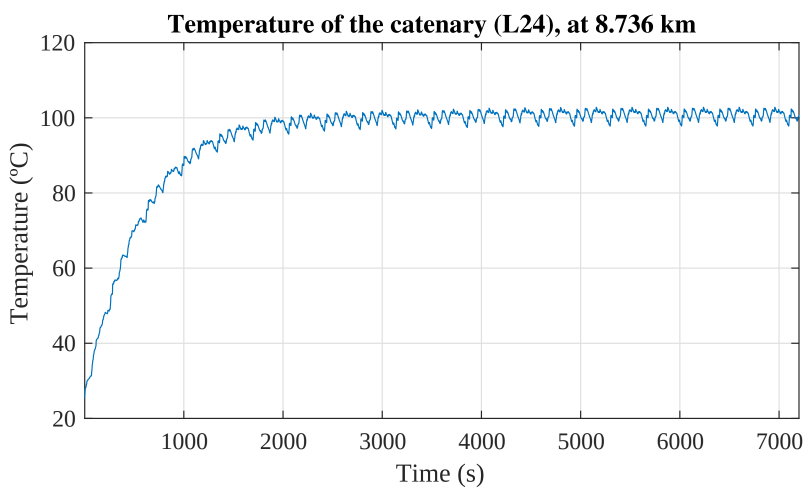

Figure 6-7: Evolution of temperature of line 24 along the simulation

Chapter 7

Structure and code of the RailThermal app

RailThermal app is a program developed with Matlab which consists of a combination of functions that as result, it performs an iterative calculation where an initial temperature and climatological conditions are supposed. In order to carry out this method, all the lines have been split in sections. The length of each section is an adaptable figure that can be changed depending on the amount of data that must be managed. The program calculates the temperature of each section throughout the simulation. For each iteration and section, the variation of temperature is obtained and added to the temperature of the previous instant. Nowadays, the simulator study the temperature of the catenary as a equivalent resistance between feeder and catenary. In future projects a more detailed grid will be define in RailNeos 2.0. The connection between feeder and catenary will be specified. Furthermore, both directions of the catenary will be taken into account.

7.1 Design of RailThermal app

The interface has been designed by means of Matlab's App Designer. This tool allows to set the distribution of the different blocks that compose the interface and their elements, as edit fields, drop down (allowing to select a option for a list), buttons or check box. Furthermore, the graphic properties of the components including color, size, position, typography or images linked to a button are configurable.

The functions in charge of managing the data and carry out the calculation has been programed with Matlab by means of scripts. The flowchart of the code implemented is presented in Fig. 7-1. Heeding this figure, it can be seen that *Thermal_study.m* is the main file of the simulation where the calls for the functions are defined. In the following sections, a breakdown of the files that composed the code is carried out.

7.2 Main script: Thermal_study.m

This script is split into three parts. In the first one, the connection to the SQLite database file is defined, by means of the function *sqlite*. A sqlite object is created, allowing to work directly with the database *res.db*. In the second part, the calls for the primaries functions are established. Finally, the third part is used for plotting the graphs.

7.3 Function: contact_wire.m

In order to cover all the types of catenaries, a database (*Contact_wire.db*) with different types of wires has been set up. In this database, information of the main electrical and mechanical characteristics of the wires is recorded. The database is organized according to the type of conductor and its section. The range of section moves from 80 to 150 mm^2. Once, the database has been read, a structure called *Conduct_Types* is created, where the fields are the types of the cateneries (see Fig. 7-2).

Function *contact_wire.m* has as goal to create the connection to *Contact_wire.db* database, read and filter the data necessary for the thermal calculations. In this study, section, specific mass and specific heat of the conductor metal wire are the parameters needed.

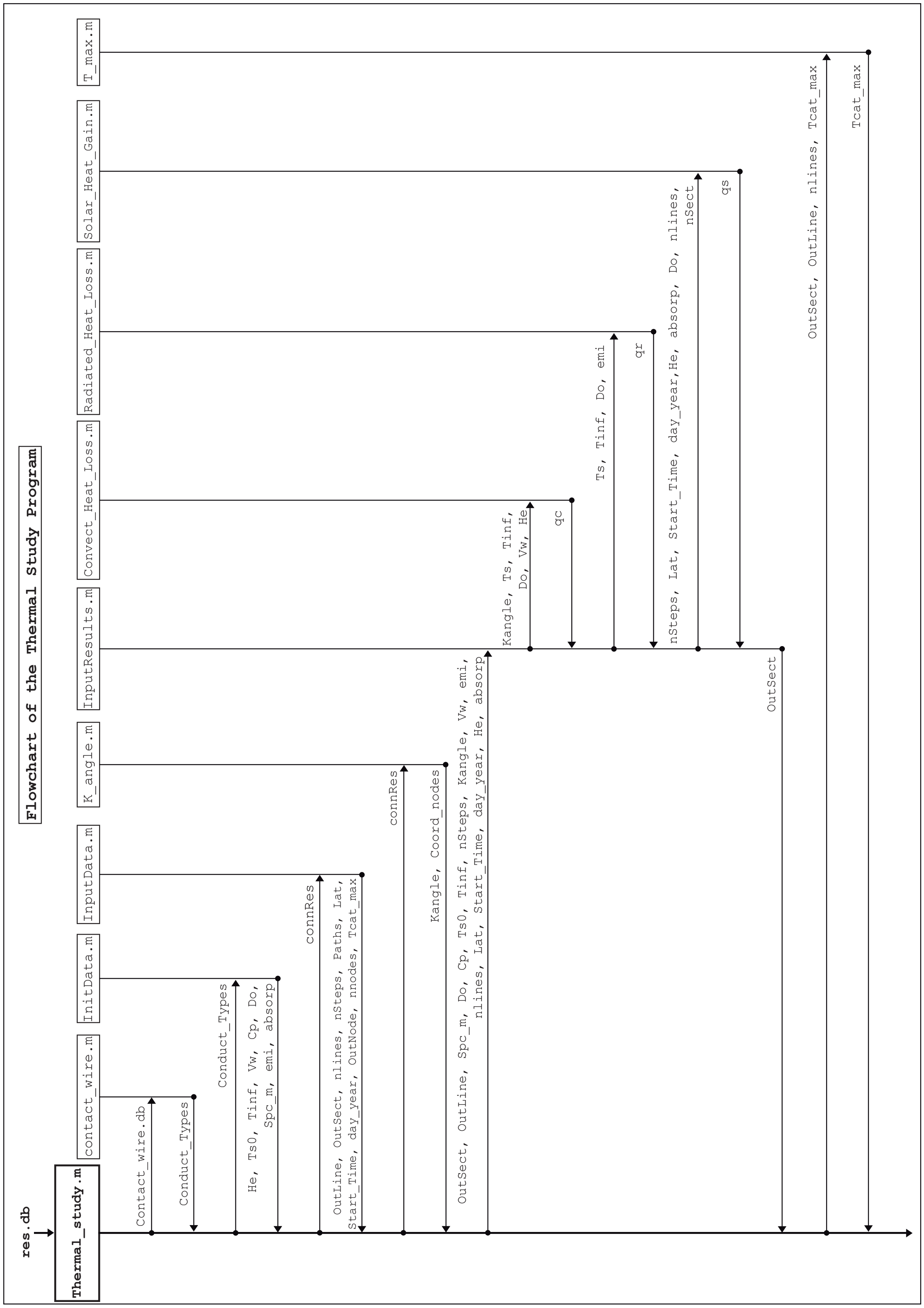

Figure 7-1: Flowchart of the functions implemented

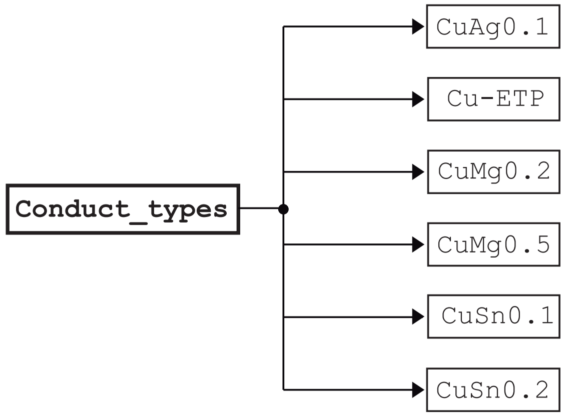

Figure 7-2: Conduct_Types structure and its fields

7.4 Function: InitData.m

This function is used to initialize the main data of the simulation. As previously mentioned, the calculation method used is based on an iterative process in which it is necessary to take a starting temperature. It is in this function, where the initial values of the temperature of the catenary (Ts0) and the ambient temperature (Tinf) are established. Both figures are in Celsius degrees. It is important to note that for the thermal calculation the Ts0 must be higher than Tinf. Otherwise, in the first iteration, complex numbers would be obtained.

For the calculation of the heat convection losses, the values of altitude with respect to sea level of the location (He), in meters, and the wind speed (Vw) estimated during the simulation, in meters per second, are introduced.

Then, the characteristics of the conductor are specified, as well as, Diameter (Do), in meters, specific mass (Sp_m), in kg/m^3, and specific heat of the conductor (Cp), in $J/(kg°C)$.

Also, the figures of emissivity for the calculation of the radiated heat loss and absorptivity for the solar heat gain are established.

7.5 Function: InputData.m

Once the input data is already defined and the connection with the results database is established, it is proceed to the accomplishment of reading and data conditioning tasks for their later use in the calculation functions. Below the parts that make up this function are detailed.

7.5.1 Import data of the nodes

The information related to the nodes will be organized as a structure where their main fields are shown in Fig. 7-3. The field *Pos_XY* gives information of the coordinates of the nodes. It is important to know that all the nodes belongs to the first quadrant. This is an relevant factor that affects the location of the sectioning points. When the different forms of graphic representation are discussed, this topic will be detailed. The field *Name* collects the denomination of the nodes established in RailNeos.

The data of the nodes will be used for graphic representation. On one hand, the information about the location of the nodes will be implemented in the creation of a thermal map. On the other hand, the names will be used as labels of the nodes in the thermal map and for the titles of the graphs of mean temperature and evolution of temperature with respect to time.

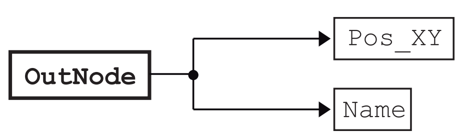

Figure 7-3: OutNode structure and its fields

7.5.2 Import data of the paths

Path is understood as a set of lines that make up a route. A structure called *Path* is created. Each path will have a field (e.g: P1, P2...). Within each of these fields, a matrix will be specified, which follows the structure proposed in Tab. 7.1. Where the first column indicates the lines belonging to that path and the second indicates the direction of the line. (0) points out that the line is traversed in the forward direction, from the source to destination node, and (1) in the reverse direction, from destination to source node.

Table 7.1: Extract data from the simulation *Lieja_BAFO.m*

Line	**Direction**
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

7.5.3 Import data of the lines

In this part, what is done, is to extract the results of the lines from the database. This data is stored in a structure, called *OutLine*, where the main fields are shown in Fig. 7-4. Each of these fields are organized by lines, with the following names, L1, L2 ...

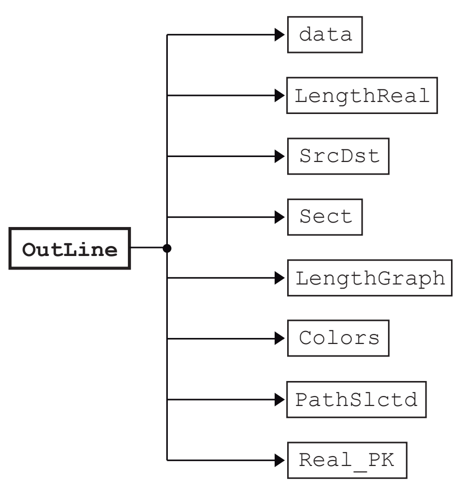

Figure 7-4: OutLine structure and its fields

Within the *data* field and its respective line name (e.g: *OutLine.data.L1*), there is information of the instant of the simulation, in seconds, of the power losses of line, in kW, and the length of the line in km. Table 7.2 shows the structure of this variable.

Heeding Tab. 7.2, it is observed that the instant 28 has two rows. This occurs when a train is circulating along the line at that moment. When this happens, RailNeos creates as many dynamic lines as trains plus one is circulating along the line. The length of these dynamic lines will vary as the vehicle moves. In the field *LengthReal*, the data of total length of the lines is stored. For the graphic representation of the thermal map, it is necessary to know the source and destination nodes. For that reason, *SrcDst* field has been created, where this data is collected.

Table 7.2: Extract data from the simulation *demo_compleja.m*

Instant (s)	**P$_{losses}$ (kW)**	**Lenght (km)**
26	6.929	4.378
27	6.929	4.378
28	0.005	0.264
28	6.503	4.114

For how the calculation model is designed, the lines will be divided into sections, where the length of each of them will be an input parameter of the application. Depending on the length of selected sectioning, the amount of data that must be treated will be greater or lower, having a direct impact on the simulation time. In the case study section, the impact of this variable on the final temperature results will be analyzed. The kilometric points of the sections will be collected in the *Sect* field, taking as zero, the origin node of the line. The sectioning length is rounded allowing that all the sections of the same line will be equal.

In order to make the graphical representation of the data, a thermal map will be plotted, where it is necessary to know the location of the nodes and the distance between them. Therefore, the distance is calculated from their coordinates. These lengths are saved in the *LengthGraph* field.

Each of the lines will be assigned to a color depending on the path. The first six paths will have a generic color and those lines that are not linked to any path or that are linked to a seventh path or higher, will be assigned to a random color. There

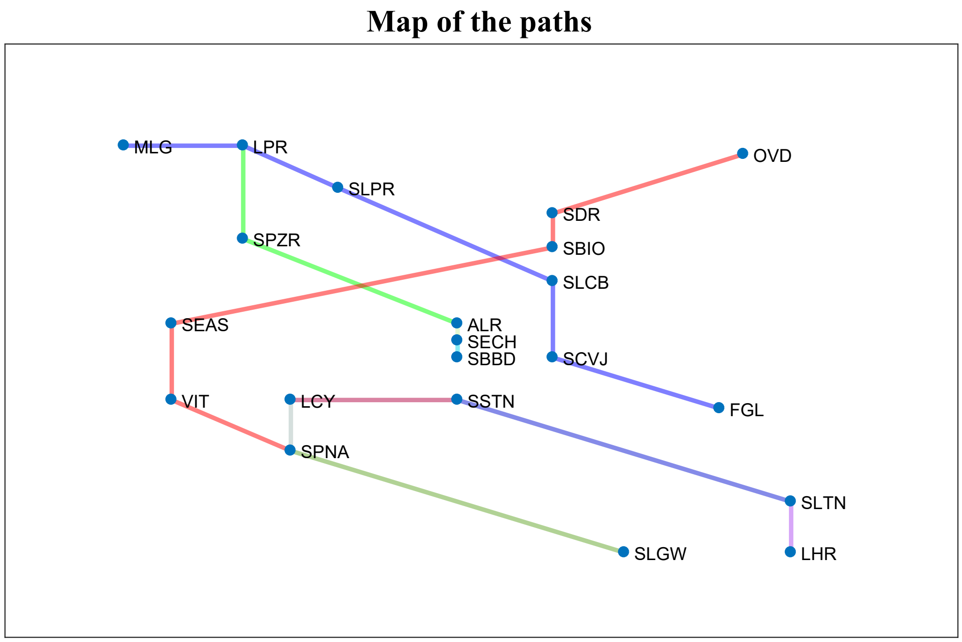

Figure 7-5: Graphical representation of the paths of a simulation, distinguishing the routes by colors

are cases in which a line can belong to several paths, for those situations, only one color is selected. The paths linked to each of the lines are collected in the *PathSlctd* field. According to the selected path, a color is set up, being gathered in *Color* field. The colors of the lines are recorded as svg format. In Fig. 7-5, a map of the paths is plotted.

Finally, in the field *Real_PK*, the information of the real kilometric point of the path is stored, which will be used in the graphical representations. If one of the lines does not belong to any route, the initial kilometric point is taken as 0.

7.5.4 Store data of the sections

Once the data of the lines, nodes and paths have been imported and stored in their respective variables, what is going to be done is a reorganization of this data. A new structure is created, allowing to organize the lines by sections. This new structure is called *OutSect*, where the data of the lines are saved in different fields. In Tab. 7.3, the composition of the matrix for a specific line is shown. As can be seen, the first column, refers to the instant of the simulation in seconds. The second column shows the final kilometric point of the section. And the third column shows the power losses of the catenary in kW. When the convective heat loss, radiation heat loss and solar heat gain are obtained, these will be added to the matrix as new columns. In the section where the *InputResults.m* function is explained, the matrix with all the parameters will be shown.

Table 7.3: *OutSect* matrix of a specific line with the initial data

Instant (s)	**Sect (km)**	$\mathbf{P}_{Losses}$ **(kW)**
8	0.0965	1.8677
8	0.1930	1.8677
9	0.0965	1.2755
9	0.1930	1.2755
10	0.0965	0.6070
10	0.1930	0.6070

Method of calculating the losses by sections

As it has been seen previously, in the database exported by RailNeos, the values of losses in the catenary are related to a length of the line. That length could be the total length of the line or a stretch of the it. From now on, a portion of the line will be called segment. Below, the method proposed to associate the power losses to the sections will be explained.

In order to understand the method of association between power losses and sections, the structure and nomenclature adopted by the lines must be known. The line is divided into segments from the database point of view and into sections from the RailThermal app. The database provides the value of the segments, in kilometers, and their linked values of the power losses, in kW.

Heeding Fig. 7-6, it is possible to observe the representation of a line in a specific instant. The segments are split into two parts. The first part is delimited by *start seg* and *end seg* points. Here, the power losses per distance are constant, therefore the losses in all sections are equal. The second part is related to the section in which the train is allocated, being represented as a blue band. In this section, there will be a power losses by distance different at both sides of the train.

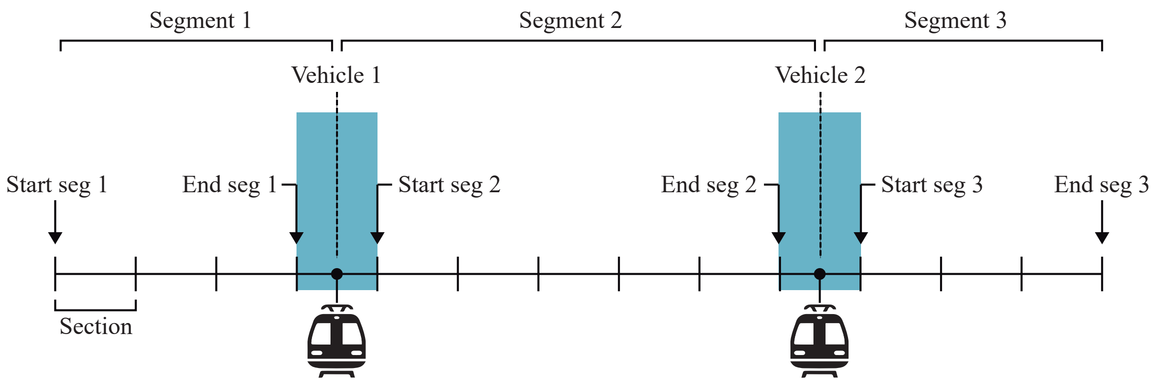

Figure 7-6: Structure and nomenclature of a line in order to arrange the power losses by sections

In Fig. 7-7, the flow diagram for the calculation of losses per section is shown. This method is a process, which goes through all the lines and all the instants, storing the results into *OutSect* variable. In order to perform this method, it is necessary to differentiate between lines with trains circulating and without trains. In case, there are no trains, the losses per section will be the same for all of them. Note that in Tab.7.3, sections have the same power losses, i.e, there is no train circulating through the line. On the contrary if there is, at least, one train circulating, what will be done is to go through all their segments calculating the losses per section. In each iteration, the losses of all sections related to a segment will be calculated, including the section in which the vehicle is located. For this portion, the losses at both side of the train are obtained, taking into account that power losses per km are different. For the next iteration the values of *start seg*, *end seg* and location of the next train are updated.

The segments are classified into three groups, first segment, intermediate segments and final segment of the line. For the first segment, the program must distinguish if there is a train in the first section or not. If there are more than one train, intermediate segment will exist. In that case, four situations can happen (see Fig. 7-7):

1. There is only one train.

2. There are two trains in non-consecutive sections.

3. There are two trains in consecutive sections.

4. There are two or more trains in the same section.

Finally, the last segment make reference to the portion from the last train to the final point of the line.

Considering these possible states, RailThermal app allows to increase the length of the sections leading to a more agile process of data losses reorganization. The impact of the sectioning length will be a factor analysed in the case study part.

7.5.5 Import start time of the simulation

The *res.db* provides data for the start time of the simulation. This factor is important to analyze the impact of the solar heat gain at different hours. It is given in seconds and it will be transformed into hour angle in *Solar_Heat_Gain.m* function. A start time equals to zero means that the simulation started at 00:00:00 with an angle equal to -180°.

7.6 Function: K_angle.m

As was commented previously, K_{angle} is the wind direction factor used to calculate the forced convection. This factor depends on ϕ, which is the angle between the conductor and wind direction. Heeding Fig. 7-8, it is clearly see that the highest factor is obtained with an angle equal to 90 ° and the lowest value with 0 °. *K_angle.m* function stores data of wind direction factor for the range of 0 ° to 90 °.

7.7 Function: InputResults.m

Once, the data of the sections has been structured properly, the thermal calculations will be carried out in *InputResults.m*. An initial temperature of the ambient and a type of conductor are supposed. In order to obtain the catenary temperature per section, convective heat loss, radiated heat loss and solar heat gain must be calculated for

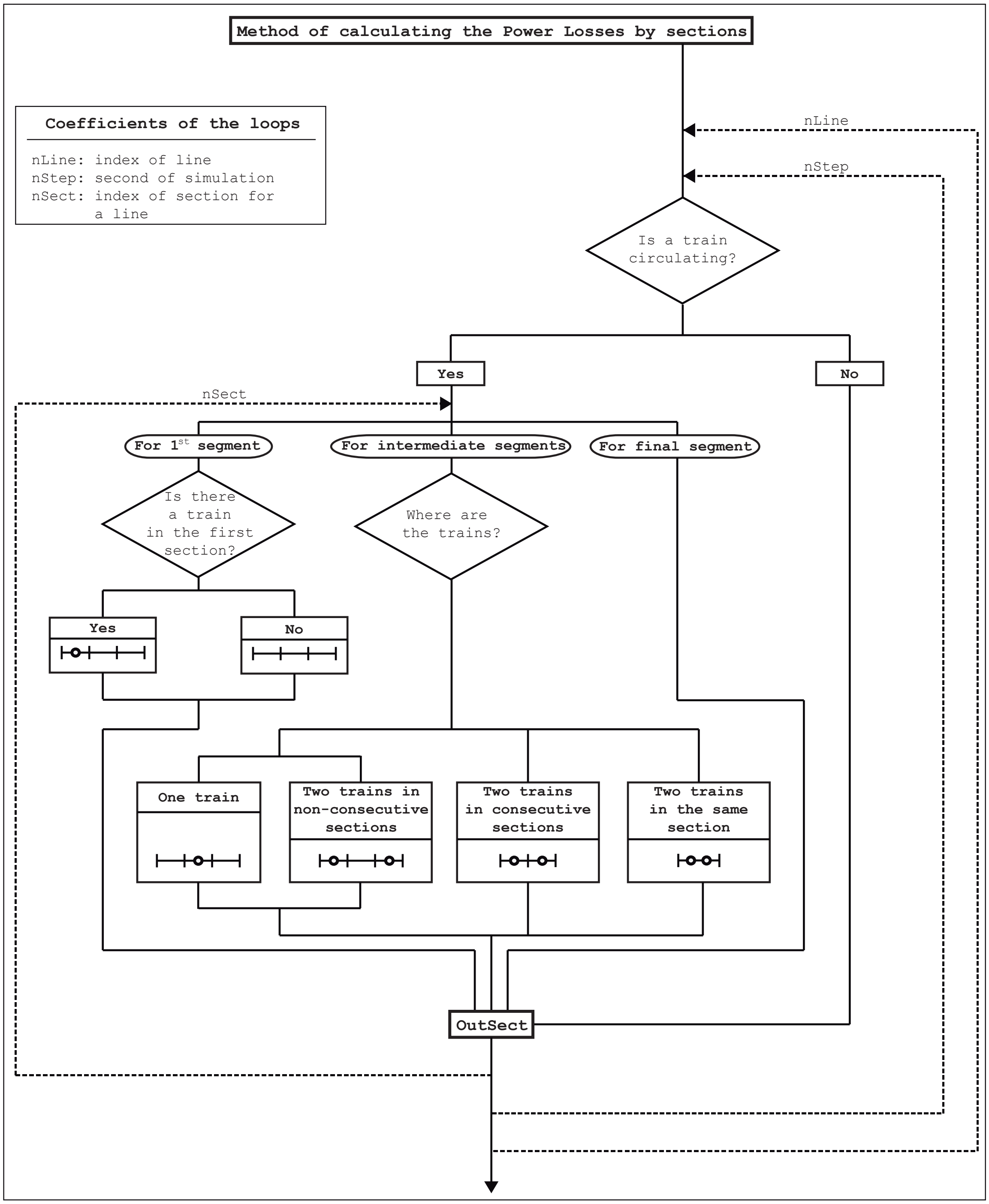

Figure 7-7: Flowchart of the code to obtain the power losses by sections

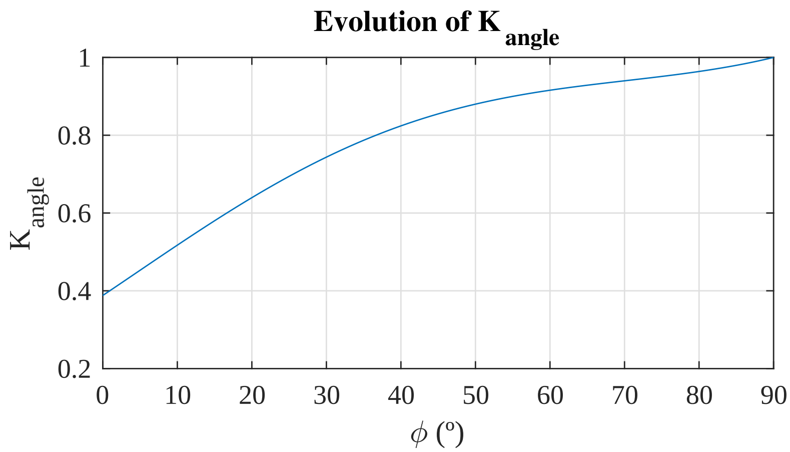

Figure 7-8: Variation of wind direction factor respect the angle between the conductor and wind direction

each instant. As is shown in the flow diagram of the code (see Fig. 7-1), three internal functions are used to perform that, *Convect_Heat_Loss.m*, *Radiated_Heat_Loss.m* and *Solar_Heat_Gain.m*. In these functions, the temperature of the conductor will be updated for each second. The value of K_{angle}, temperature of air, diameter of conductor, wind speed, elevation of the location, emissivity and absorptivity remain constants along the simulation.

As a result of the internal functions, the values of heat losses by convection, losses by radiation and gain by solar incidence in kW per meter, will be obtained. These values are multiplied by the length of each section and introduced in the energy balance equation to obtain the temperature variation for each moment. In Tab. 7.4, the complete matrix of the *OutSect* variable for a line is presented. As can be seen, the values of convective heat loss, radiated heat loss, solar heat gain and temperatures of the sections have been added.

Table 7.4: *OutSect* matrix of a line with the thermal results

Instant (s)	**Sect (km)**	**P_{losses} (kW)**	**C. H. L (kW)**	**R. H. L (kW)**	**S. H. G (kW)**	**Temp (°C)**
8	0.0965	1.8677	0.7589	0.0324	0.2239	26.0538
8	0.1930	1.8677	0.7589	0.0324	0.2239	26.0538
9	0.0965	1.2755	0.7551	0.0327	0.2239	26.0803
9	0.1930	1.2755	0.7551	0.0327	0.2239	26.0803
10	0.0965	0.6070	0.7685	0.0328	0.2239	26.0815
10	0.1930	0.6070	0.7685	0.0328	0.2239	25.0815

* *C.H.L: convective heat loss, R.H.L: radiated heat loss, S.H.G: solar heat gain*

7.7.1 Internal function: Convect_Heat_Loss.m

The first internal function is in charge of calculating the convective heat loss. The distinction between natural or forced convection is made, being 0.2 m/s the breaking speed between both cases. For forced convection, the most unfavorable case is adopted. In order to perform that, two considerations are set:

- The angle between line and wind direction is supposed zero for all the cases, i.e, lowest value of K_{angle}.
- The convective heat loss is calculated by using, Eq.5.3 and Eq.5.4. The highest value of both expression is selected.

Note that in natural convection, if the difference of temperatures is negative (see Eq. 5.7), q_s will be a complex number. Thus, the temperature of the conductor must be consider higher than the air temperature for the initial conditions.

7.7.2 Internal function: Radiated_Heat_Loss.m

For radiated heat loss the Eq. 5.8 is implemented. This term is dependent on the emissivity, the diameter of the wire, temperature of the air and conductor temperature of the previous instant.

7.7.3 Internal function: Solar_Heat_Gain.m

For this function, the latitude, start time, day of the simulation, elevation of the location, absorptivity and diameter of the conductor are taken as input data (see Fig. 7-1).The equations for calculating solar heat gain are implemented. Since start time, in seconds, and day of the simulation, it is proceed with the calculation of hour angle (ω) and solar declination angle (δ).

A start time equal to 0 seconds indicates that the simulation begins at 00:00:00 (midnight) with an angle equal to -180°. Therefore, 12 hours is equivalent to 43200 seconds and 180 degrees. Thus, for 240 seconds the hour angle will vary in 1 degree (see Fig. 5-1). Following this criterion, the expression of ω is reached (see Eq, 7.1). The variable n is used to account for the number of hours of the simulation. ω is increased by 15 degrees for each hour of simulation.

$$\omega = \left(\frac{t_{st}}{240} - 180 + n \cdot 15\right) \tag{7.1}$$

Where:

- t_{st}: start time of the simulation.
- n: counter for the number of hours of the simulation.

Furthermore, the variables for calculating total heat flux density are also defined. Commonly, a clear atmosphere is considered where the sun's rays affect the temperature increase to a greater extent.

For the calculation of effective incidence angle of the sun's rays (see Eq. 5.10), it is considered that the difference between Z_c and Z_l is 0 degrees, in order to study the most unfavorable case.

Regarding the area of incidence, it is estimated that the projected area is the result of the multiplication of the conductor diameter by the length of the line. In this way, the largest possible area of incidence is used.

7.8 Function: T_max.m

Once the thermal calculations are performed, the maximum temperatures of all sections along the simulation are split into temperature ranges:

- $T_{max} \geq 100$ °C
- 90 °C $\leq T_{max} <$ 100 °C
- 80 °C $\leq T_{max} <$ 90 °C
- 60 °C $\leq T_{max} <$ 80 °C

- 40 °C ≤ T_{max} < 60 °C

- 20 °C ≤ T_{max} < 40 °C

- T_{max} < 20 °C

T_max.m function is in charge of this task. A structure called *Tcat_max* is created, its fields are presented in Fig. 7-9. *Tcat_max* allows to distinguish the temperature ranges in the creation of the thermal map. This kind of graph is a representation of the paths, showing the maximum temperatures. What this graphic shows and how it has been done will be discussed later

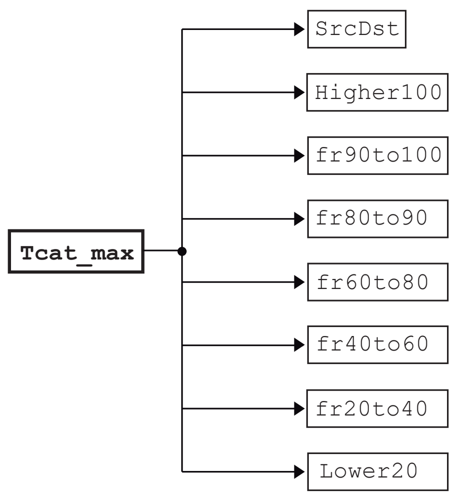

Figure 7-9: Tcat_max structure and its fields

The goal of this function is to look for the maximum temperature of all the points that composed the paths and compute for how many minutes a temperature has exceed 120 °C. This limit is established following Tab. 7.5, where the maximum temperatures of the conductors are defined.

In the paths, there are two types of groups where the temperature is computed. Firstly, the source and destination node of the lines. Its data is stored in the field *SrcDst*, saving their maximum temperature and minutes with a temperature higher than 120 °C. This field is structured according to the index of the nodes. Second group is constituted by the sections. The data of the sections is found in the rest of the fields (*Higher100*, *fr90to100*...). Here, the data is split by lines. Instant with

Table 7.5: Maximum temperature of the conductors in °C [34]

Material	**Up to 1 s (Shortcircuit current)**	**Up to 30 min (pantograph idle)**	**Permanent (operation conditions)**
Cu-ETP	170	120	80
CuAg0.1	200	150	100
CuSn	170	130	100
CuMg0.2	170	130	100
CuMg0.5	200	150	100
ACSR/AACSR	160	-	80

maximum temperature, location of the section, maximum temperature, minutes with temperature higher than 120 ° and the real kilometric point are defined in each line.

7.9 Graphic representation of the thermal calculation

With the thermal calculations and the maximum temperatures organized by ranges, it is proceed to create the graphic representation. As was commented, three forms for plotting the data have been implemented in RailThermal app:

- Thermal Map.
- Average temperature of lines.
- Evolution of the temperature over time.

Thermal Map is implemented as a grid graph of Matlab. This type of plot is constituted by points and lines. In order to draw it, it necessary to define the source and destination nodes of the lines and the XY coordinates of them. This data is already store in the variables *OutLine* and *OutNode*. Firstly, the lines of the grid will be created, defining the names of the nodes and the colors of the lines according to their paths. Then, the lines will be split into sections. The sections of the lines are also considered as points. Hence, it will be required to define the positions of these new nodes. For locating the points of the sections, it is required to specified the direction of the line. Therefore, a new function is created, called *getAngle.m*, which is in charge of calculating the direction of the line, since the position of the source and destination nodes of the line. A four quadrant study will be carried out, where the expressions of each situations are presented in Fig. 7-10. Note that, all XY coordinates of the points are considered positive.

The nodes of the grid will be related to quantitative values as maximum temperature of the simulation, number of minutes with a temperature higher than 120 °C, and to qualitative characteristics as color, size and name. In order to show these features, the data cursor tool of Matlab is used. A callback function is created, *GraphCursor-Callback.m*. Each time a node is clicked, this function is called, locating the node and displaying the index of the node, its maximum temperature and the minutes with temperature higher than 120°C (see Fig.6-5).

Two other types of graphs are also defined. The first one represents the average temperature of the line respect to the distance. What is done is to calculate the average temperature of each section throughout the simulation. In order to calculate the mean value of temperatures, only values of the second half of the simulation are taken into account. Obviating in this way, initial results of the simulation that are far from the final values of permanent regime. Furthermore, a graph of temperature evolution over the time is created. The header of this graph indicates the index of the line and the kilometer plotted. Some check boxes have been designed in the user's interface, so only the graphs selected are going to be displayed.

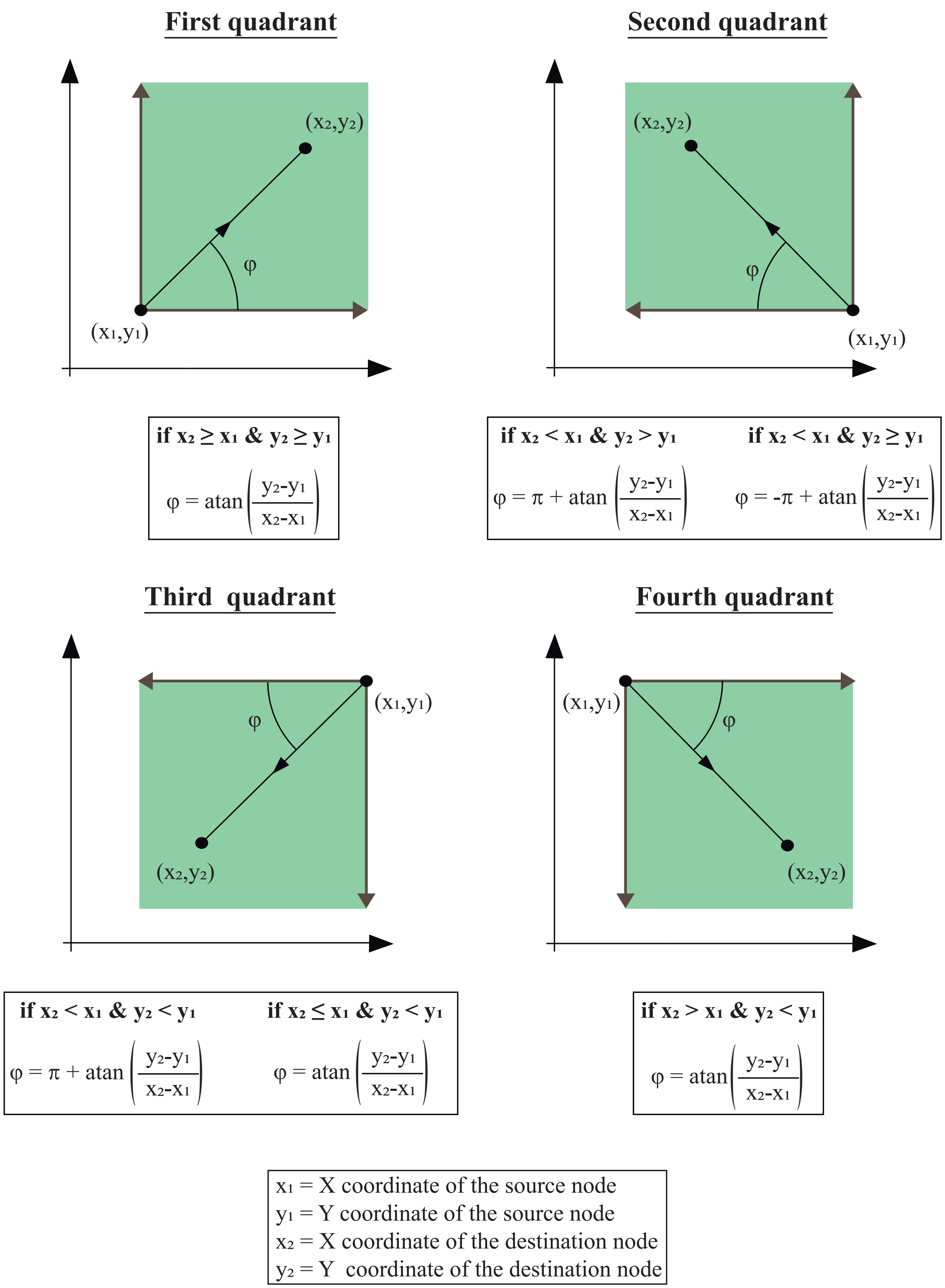

Figure 7-10: Criterion used for calculating the direction angle, showing the possible directions of the lines and the equations

Chapter 8

Analysis of the results provided by RailThermal app

This chapter is divided into three parts. In the first one, the main parameters that affect the variation of temperature will be studied. Their dependence on the inputs parameters will be analyzed. Second part will be focused on studying the influence of the sectioning length, both for error incurred and simulation time. Finally, in the third part a case study will be developed, using RailThermal app.

8.1 Analysis of convective, radiated heat loss and solar heat gain

In this section, the evolution of convective heat loss, radiated heat loss and solar heat gain will be analyze. A series of graphs will be used to compare different case studies and to draw conclusions.

8.1.1 Convective Heat Loss

The convective heat loss (q_c) is a parameter that depends on K_{angle}, temperature of conductor (T_s), temperature of air (T_∞), diameter of conductor (D_o), wind speed (V_w) and elevation of the location (H_e). In order to study it, q_c have been plotted against V_w for different situations. For all graphs a T_s equal to 25 °C and T_∞ equal to 20 °C have been supposed.

In Fig.8-1, the evolution of q_c has been plotted varying the nominal section of the conductor. As can be seen, the graph can be split into two parts. On the left, q_c remains constant for each case. This part represents the natural convection where V_w is considered null. For V_w higher than 0.2 m/s, the wave follows a growing curve. This part refers to the forced convection. At the same time as V_w is increasing the convective heat dissipated increases. With higher section, q_c is greater. For this case, it is supposed a K_{angle}=0.39 and H_e=100 m.

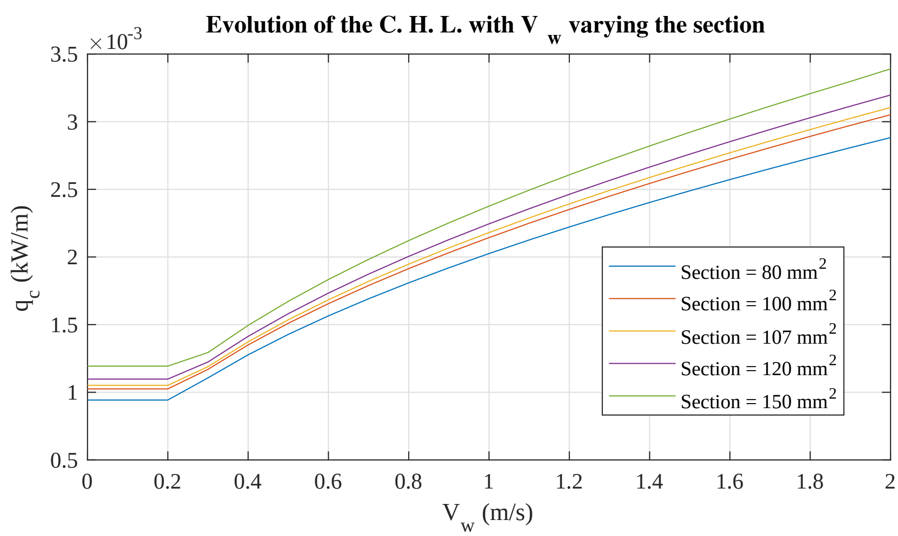

Figure 8-1: Evolution of the q_c with the wind speed and changing the section of the conductor

Heeding Fig 8-2, it is observed that the elevation of the location over sea level alters q_c minimally. It can be said that for locations at sea level, the losses by convection are slightly greater than in places with higher elevation. For plotting this graph, a section equal to 80 mm^2 has been used.

Fig. 8-3 shows the influence of variable K_{angle} on q_c. This factor affects more than conductor section and H_e. Remember that this parameter is related to the angle between wind and line direction (ϕ). In Tab. 8.1, the relationships are gathered. For values of K_{angle} close to 1, i.e, wind direction is perpendicular to line, the heat removed by convection will be higher. This study has been carried out using H_e equal to 100 m and section equal to 80 mm^2

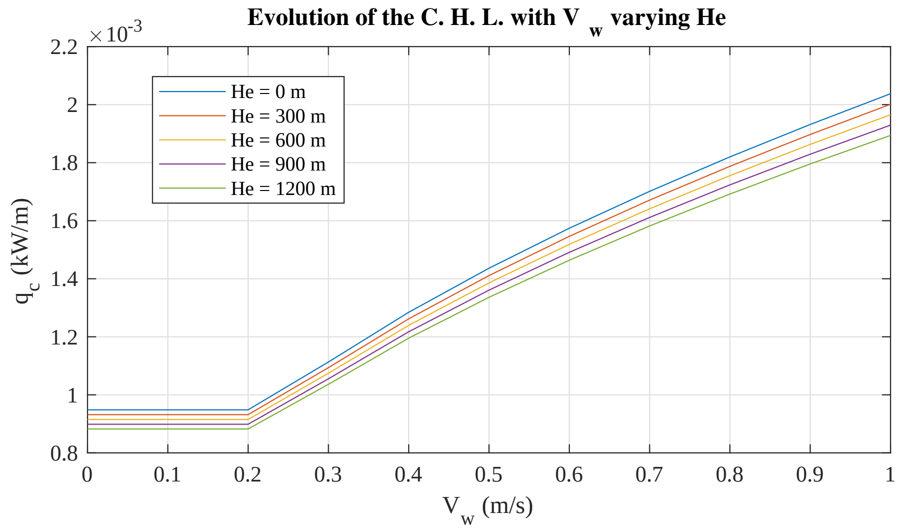

Figure 8-2: Evolution of the q_c with the wind speed and changing the elevation of the location

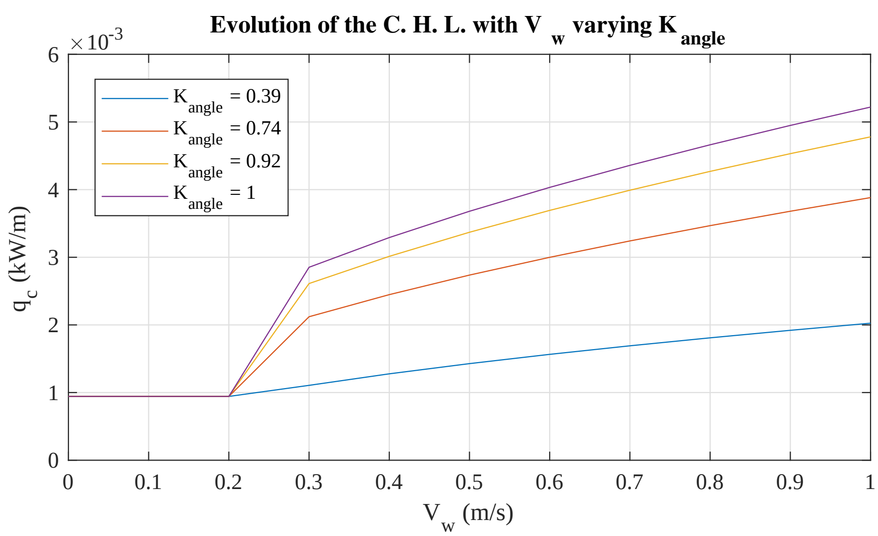

Figure 8-3: Evolution of the q_c with the wind speed and changing K_{angle}

8.1.2 Radiated Heat Loss

Radiated heat loss (q_r) is a parameter that changes depending on T_s, T_∞, D_o and emissivity (ε). In Fig. 8-4, the evolution of q_r for different sections and emissivities are presented. Note that ε is a characteristic of the conductor that increases over the

Table 8.1: Relationship between K_{angle} and ϕ

$\mathbf{K_{angle}}$	ϕ (°)
0.39	0
0.74	30
0.92	60
1	90

time. As can be seen, q_r grows with ε and with high sections. The same temperatures for conductor and air as in convective heat loss study are used.

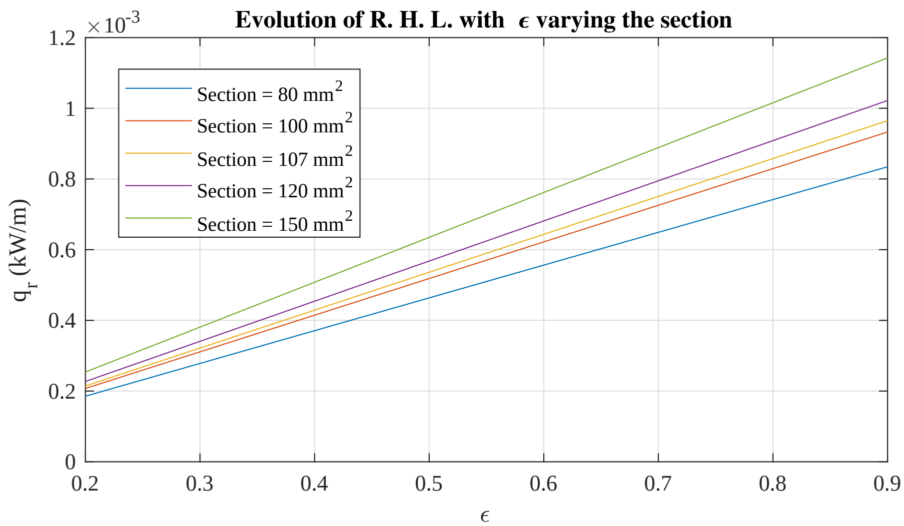

Figure 8-4: Evolution of the q_r with emissivity and changing the section of the conductor

8.1.3 Solar Heat Gain

For studying solar heat gain (q_s) several scenarios will be analyzed, taking into account its input variables. Q_s depends on latitude of location (Lat), star time of the simulation, day of the year, H_e, absorptivity (α) and D_o. For all the graphs shown below, the evolution of q_s along the day will be plotted.

The first study has the goal to compare q_s for different latitudes. In Fig. 8-5, the results of q_s at the summer solstice in the northern hemisphere are presented. Q_s reaches the peak value with a Lat equal to 25°, followed by locations with Lat equal to 0° and 50°. The place in which the conductor will be exposed more time to the sun is with a Lat equal to 50°. The locations on southern hemisphere are in winter solstice, therefore their q_s is smaller. The opposite happens when the summer solstice is in the southern hemisphere. Negatives latitudes and equator will have the highest values of q_s (see Fig. 8-6). For both scenarios, a H_e of 100 m, a section of 80 mm^2 and a α of 0.3 are used.

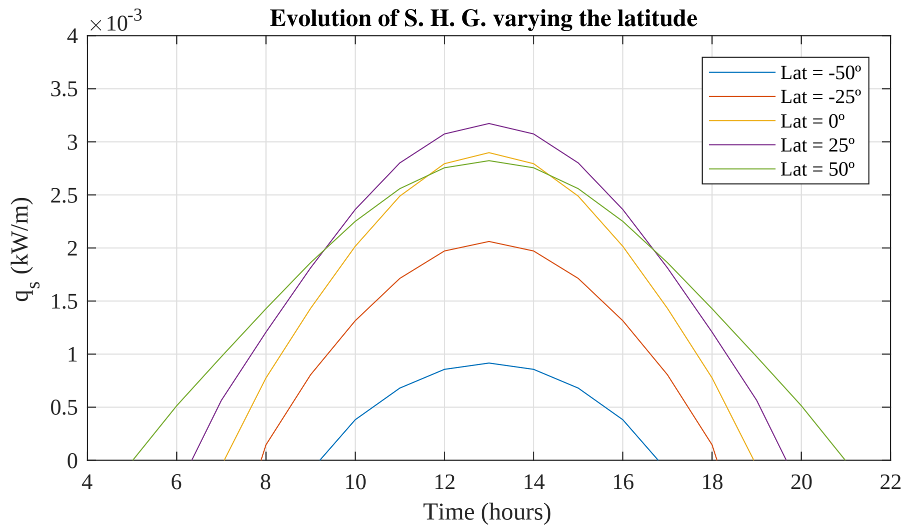

Figure 8-5: Evolution of the q_s with respect the latitude. The day of the simulation is 21-July-2019 (summer solstice in the northern hemisphere)

The second case study of q_s is based on comparing the solar heat gain along the day for different dates. Heading Fig. 8-7, it can be observed that for dates closer to summer, solar heat gain is increasing. For plotting this graph the following values are considered: Lat=30°, H_e=100 m, section of 80 mm^2 and α=0.3.

The third scenario presents q_s dependence respect H_e. As can be seen in Fig. 8-8, there is not a great variation on q_s for different altitudes. It can be highlighted that, for higher values of H_e, the solar heat gain will be greater. In this graph the day of simulation and latitude remain constant. Their values are 21st of July and 50°, respectively.

The next analysis is centered on studying q_s for different α (see Fig. 8-9). Absorptivity is a property of the conductor. For new wires, this value will be around 0.2, reaching 0.9 at the end of its lifetime. It is clearly see that for high values of α, q_s is

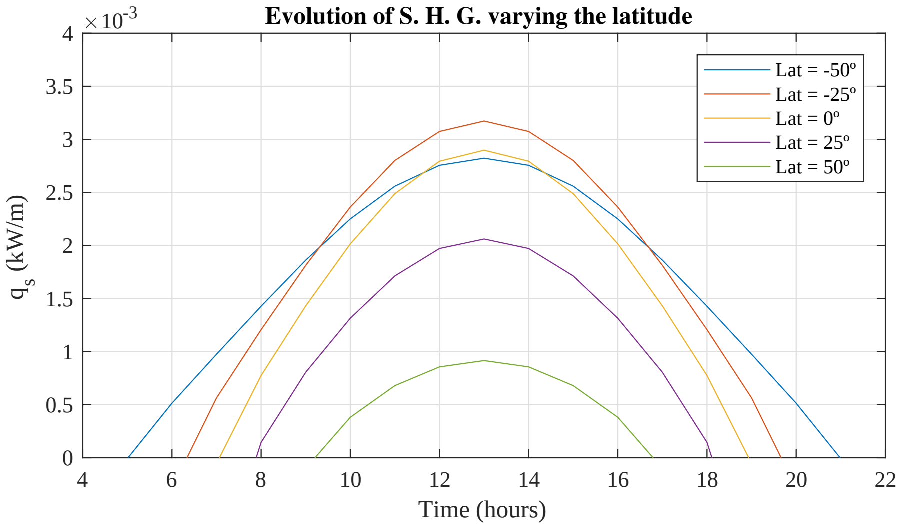

Figure 8-6: Evolution of the q_s with respect the latitude. The day of the simulation is 22-December-2019 (summer solstice in the southern hemisphere)

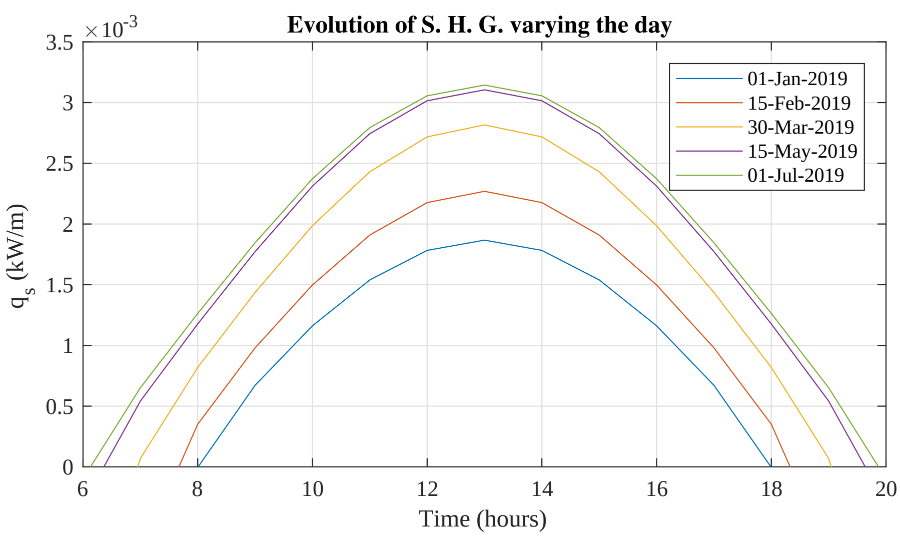

Figure 8-7: Evolution of the q_s with respect day of simulation

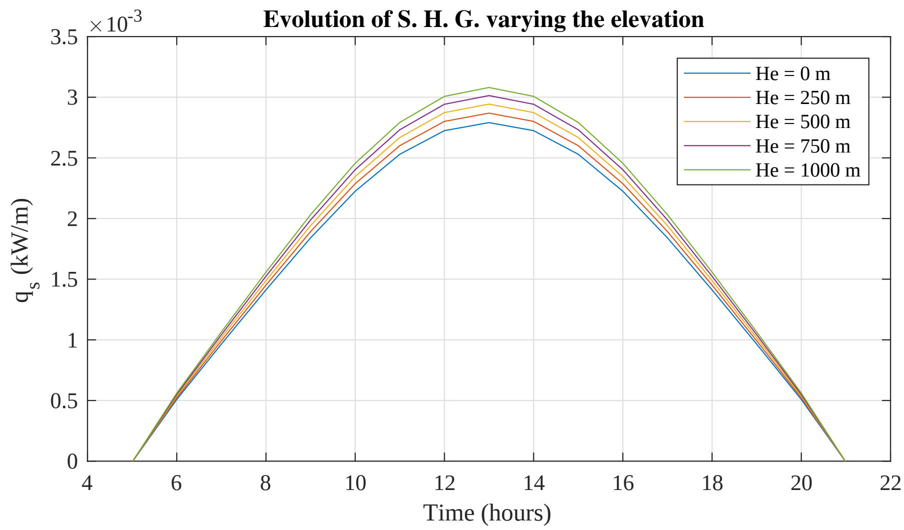

Figure 8-8: Evolution of the q_s with respect elevation of the location

greater. Comparing this graph with the previous ones, it is checked that variations in α affect in greater extent. For obtaining this graph, an elevation of 100 m, a latitude of 50 °, a section of 80 mm^2 and the date of northern solstice have been used.

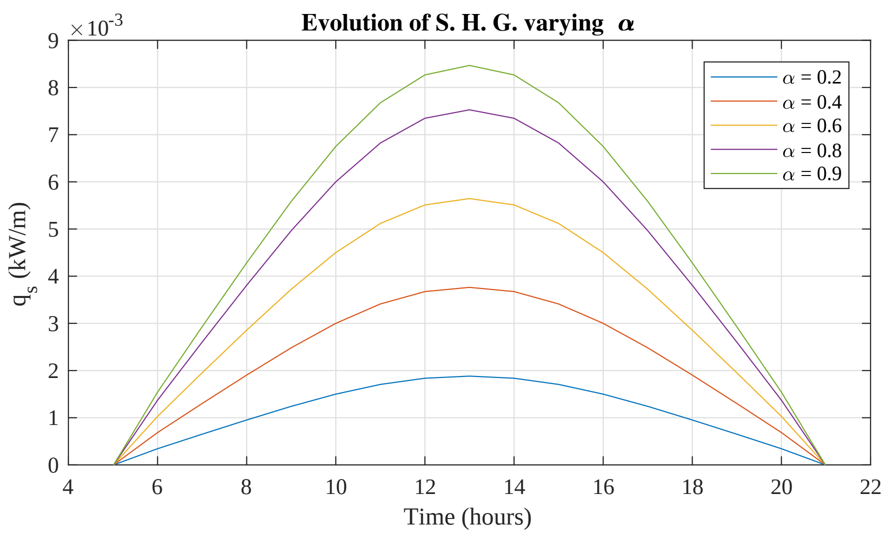

Figure 8-9: Evolution of the q_s with respect absorptivity

Finally, the evolution of q_s against the section of the conductor is tested. With Fig. 8-10, it is confirmed that for high values of the section the solar heat gain increases. The parameters used to perform this plot are: H_e=100 m, Lat=50 °, section of 80 mm^2, α=0.3 and the date of northern solstice.

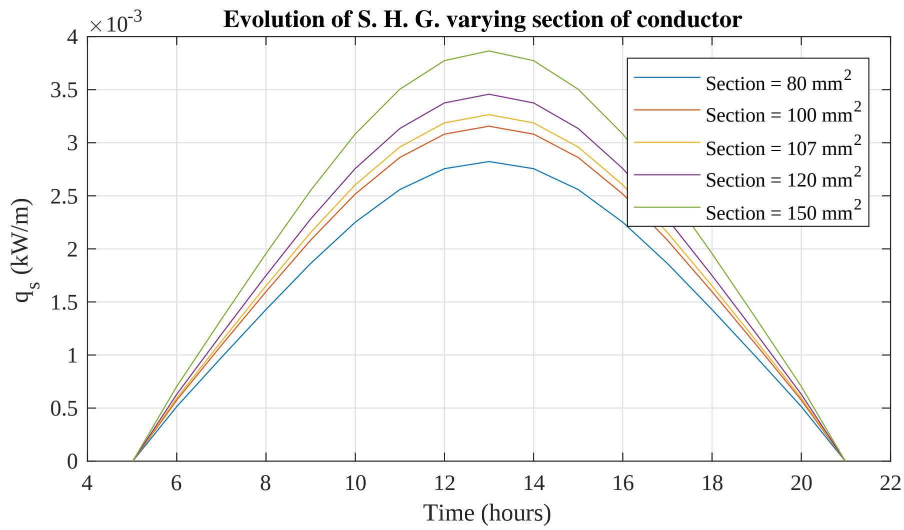

Figure 8-10: Evolution of the q_s with respect section of the conductor

8.2 Influence of the sectioning length in the simulation time and the error incurred

Sectioning length is a parameter that must be introduced at the beginning of the simulation. Mainly, this value will affect the error incurred and the time of simulation. In this section, the main goal is to study how sectioning length changes both. In order to do that, the grid of Liège was selected. The analysis will be carried out, by means of average temperature and the evolution of temperature with respect to time. The graphs will be focused on line 2 of the grid.

Four different length of sections have been used. It is important to remember that the section value entered in the interface is rounded to ensure that all sections are equal. In tab. 8.2, the relation between both values are shown. In order to compute the relative error of the average temperature, it is assumed the results of 25 m of section as reference, i.e, relative error with sections of 25 m are supposed 0%.

In Fig.8-11, it is presented the average temperature of line 2 for the sections. As can be seen, according as length of the section decreases, the waveform of the average temperature is more detailed. Heeding Fig. 8-12, it is clearly see that, higher the length of the section, greater the relative error. With sections of 200 m, there are

Table 8.2: Add caption

Sect. Length (m)	Real Sect. Length (m)
200	177
100	101
50	50.5
25	25.25

peaks of relative error around 7%. When section is reduced to 50 m, the relative error does not exceed 2 %. Note that for central values of the line, the errors are drastically reduced, below 1 %.

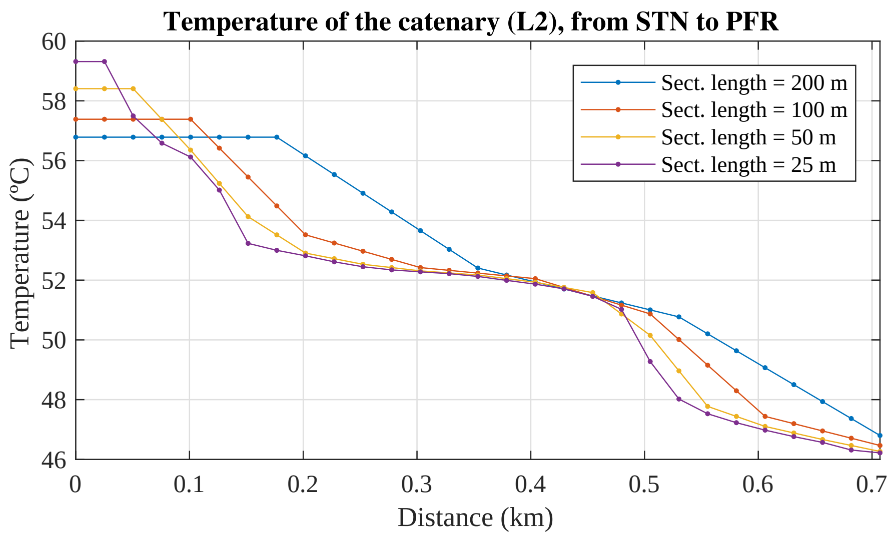

Figure 8-11: Average temperature of L2 for different values of sectioning length

Fig.8-13 has been plotted, for studying how sectioning length changes temperature from a temporal point of view. This figure shows that the highest values of temperature are reached with lowest sectioning length. Using sections of 200 m, leads to relative error between maximums of 4.8 %, i.e, 2°C. With shorter section lengths such as 50 m, the relative error is reduced to 1.7 % (see Fig. 8-14).

Finally, two graphs have been made in which the average value of the error is compared with the total simulation time, showing that both magnitudes are inversely proportional. In Fig. 8-15, the average error of Fig.8-11 has been implemented, and for Fig. 8-16, the average error of Fig.8-13 has been used. It can be seen that as the sectioning length is reduced by half, the simulation time is doubled, it means that

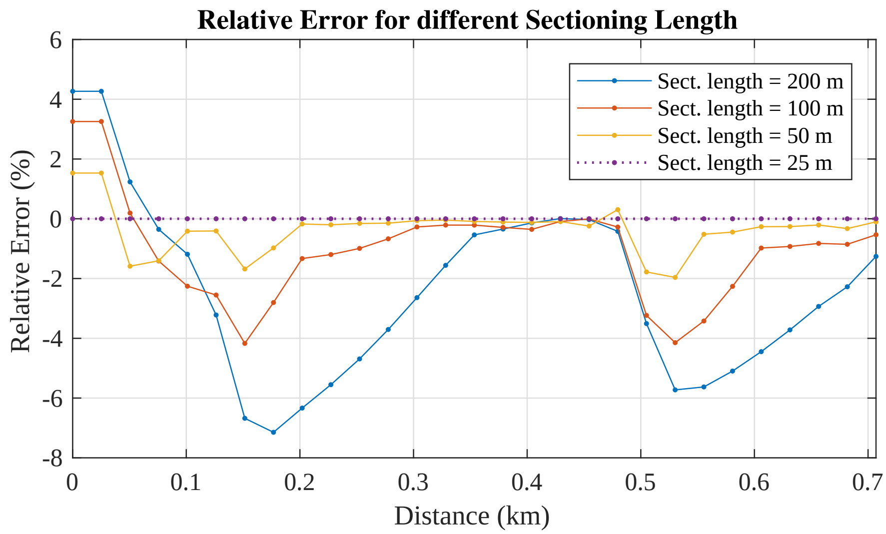

Figure 8-12: Relative error of average temperature along the line

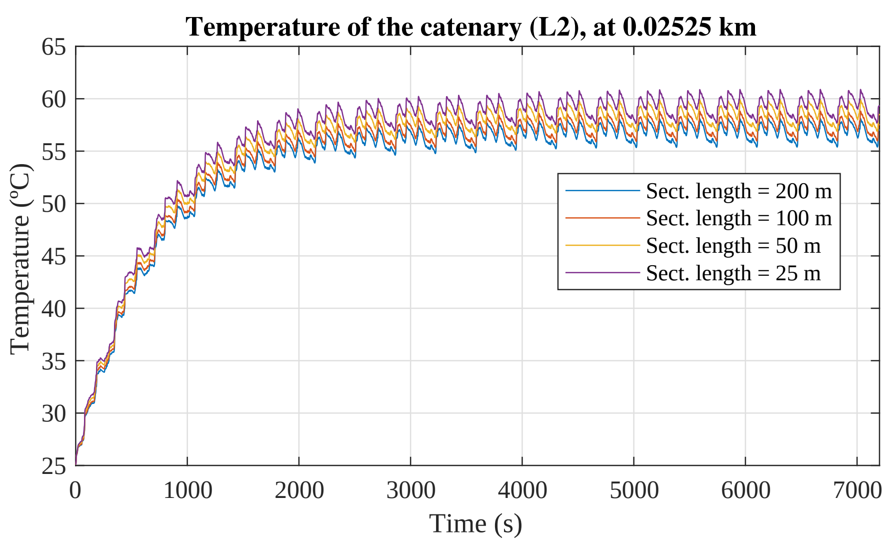

Figure 8-13: Evolution of temperature, at 0.025 km of L2

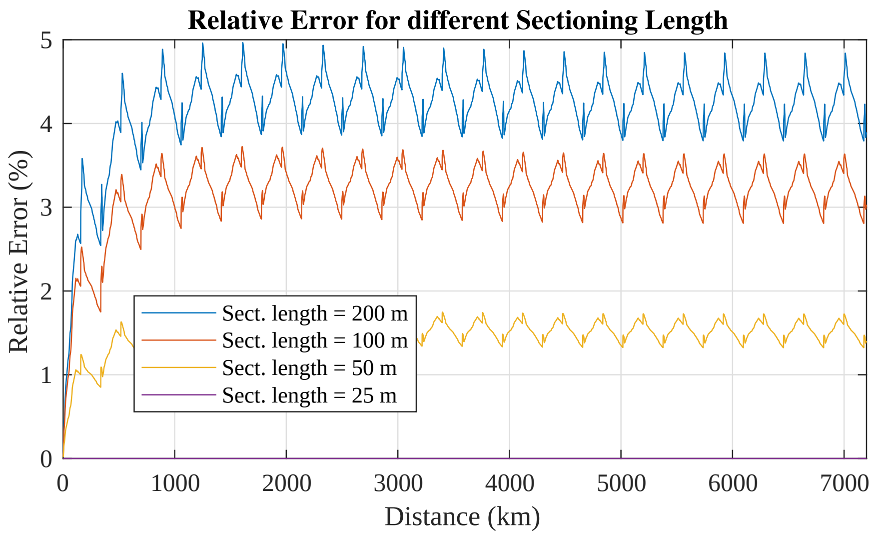

Figure 8-14: Relative error of temperature at 0.025 km of L2, during the simulation

the software must manage the double of data. It can also be observed that the error is slightly greater when the temperature is studied from the time evolution point of view.

These graphics are focused on the study of a particular line in a particular network, so the results of the relative errors and simulation time may vary from one case to another. The main factors that can determine the error are the length of the sections studied, the total simulation time introduced in RailNeos and the vehicle schedules.

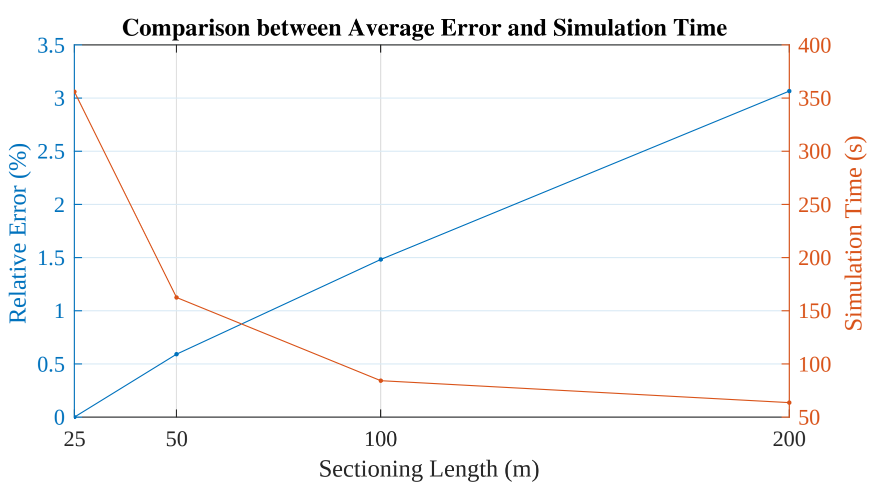

Figure 8-15: Comparison between average error of Fig.8-11 and simulation time

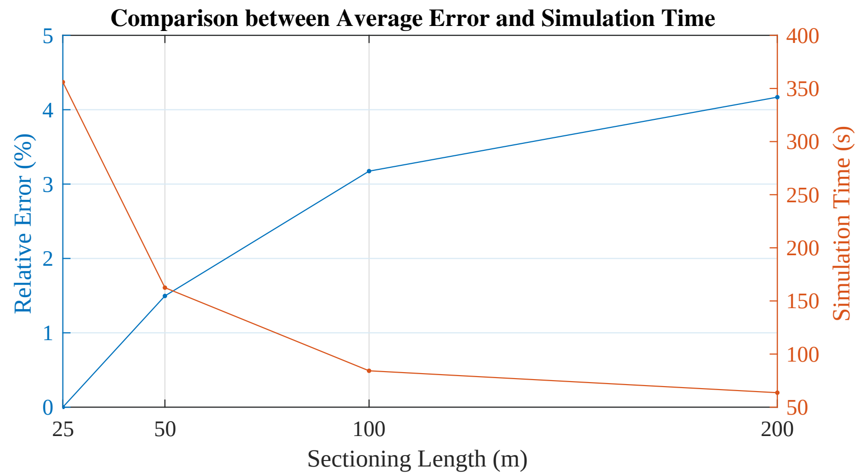

Figure 8-16: Comparison between average error of Fig.8-13 and simulation time

8.3 Case Study: Liège network

This study will be focused on analyzing the thermal behavior of a tram grid in Liège. The goal is to use the RailThermal app to carry out a thermal study, locating the points of the lines with higher temperatures. This will allow to know which is the optimal section of the conductor.

The input data of the simulation is the following:

- Initial temperature of the catenary: 25 °C.
- Initial temperature of the air: 20 °C.
- Elevation of Liège: 66 m.
- Latitude of Liège: 50.63 °.
- Secioning length: 50 m.
- Date: 21^{st} of July.
- Material of the conductor: Cu-ETP.
- Section of the conductor: 80 mm^2.

In Fig.8-17, the thermal map of Liège is plotted. As can be seen, there are two zones of the grid where temperature overcomes 100 °C. These parts coincide with the join sections of the parallel branches. There are three lines that do not fulfill the requirements established in Tab. 7.5. In Fig. 8-18, Fig.8-19 and Fig. 8-20, it is shown that temperature in permanent regime exceed the of 80 °C of permanent operation conditions.

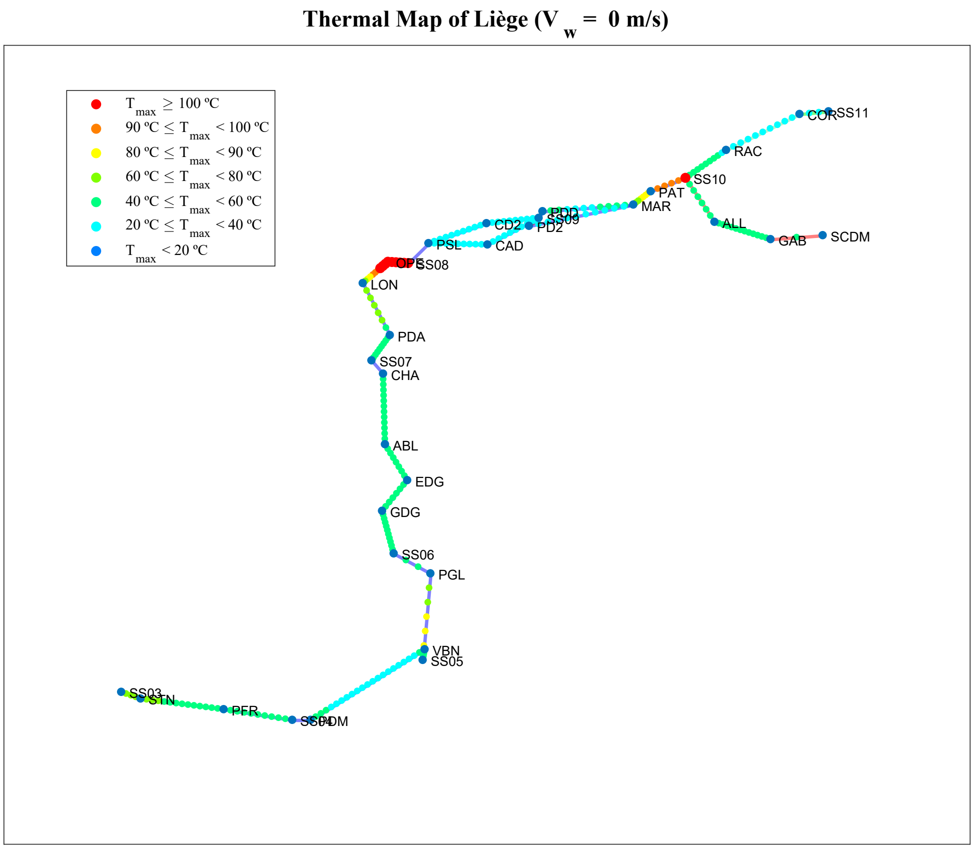

Figure 8-17: Thermal of map of Liège's network with V_w=0 m/s

Some thermal studies consider a certain wind speed so the temperature will decrease. The thermal map with a wind speed of 5 m/s is presented in Fig. 8-21. It can be observed that the overall temperature of all the grid has been reduced, considerably. But in this study, the maximum safety positioning will be used, considering zero wind speed.

Retaking, the case with null wind, two options can be made, increase the conductor section and maintain the type of material used. Or change the type of driver to another with better thermal properties. In this study, results of both options will

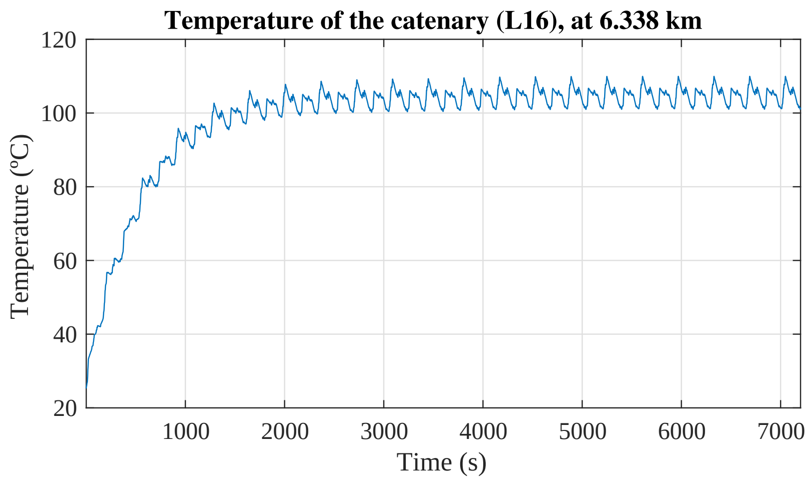

Figure 8-18: Evolution of temperature for line 16, at point 6.338 km

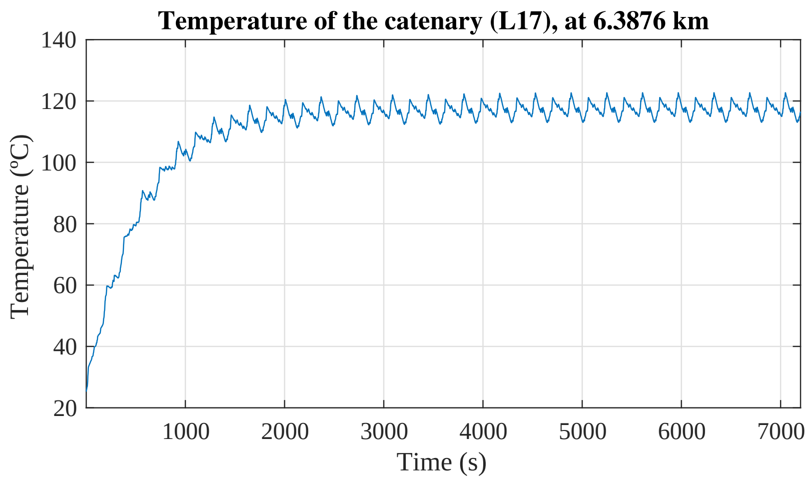

Figure 8-19: Evolution of temperature for line 17, at point 6.388 km

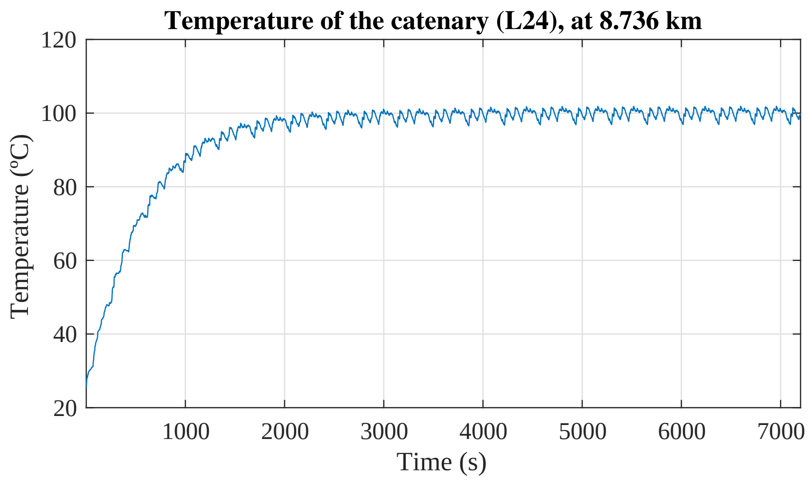

Figure 8-20: Evolution of temperature for line 17, at point 8.736 km

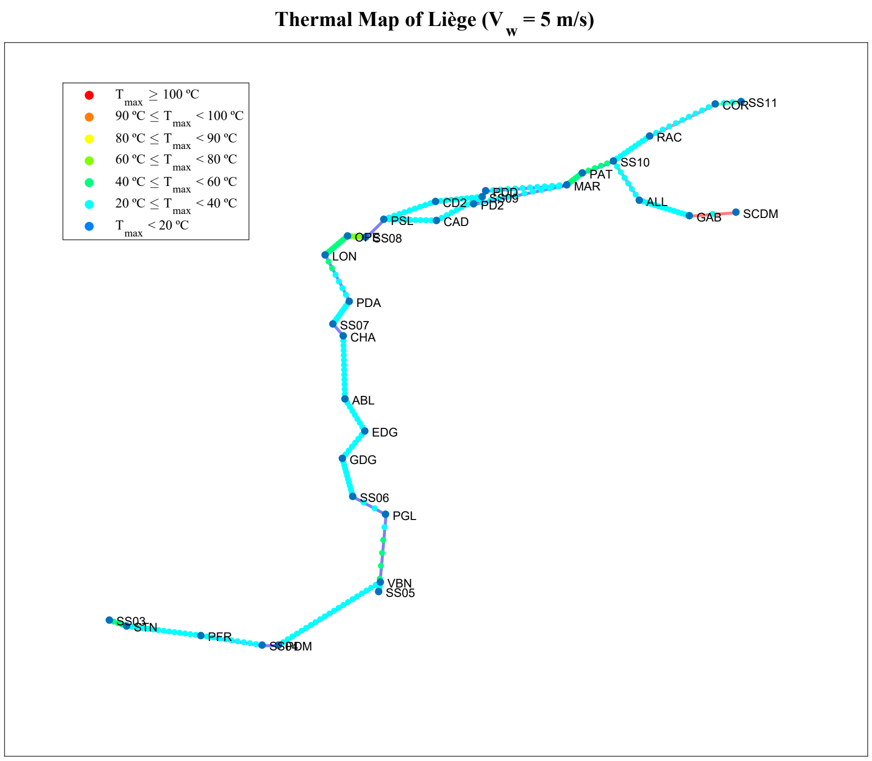

Figure 8-21: Thermal map of Liège's network with V_w=5 m/s

be shown. The results obtained using a Cu-ETP conductor with a section of 150 mm^2 are shown in Fig. 8-22 and Fig. 8-23. Both figures show that on line 17 the operation requirements for nominal conditions are still not met. L16 and L24 have temperatures slightly above the nominal allowable of 80°C. Therefore, it will proceed to use a CuAg0.1 conductor. This type of wire allows to operate at higher temperature conditions.

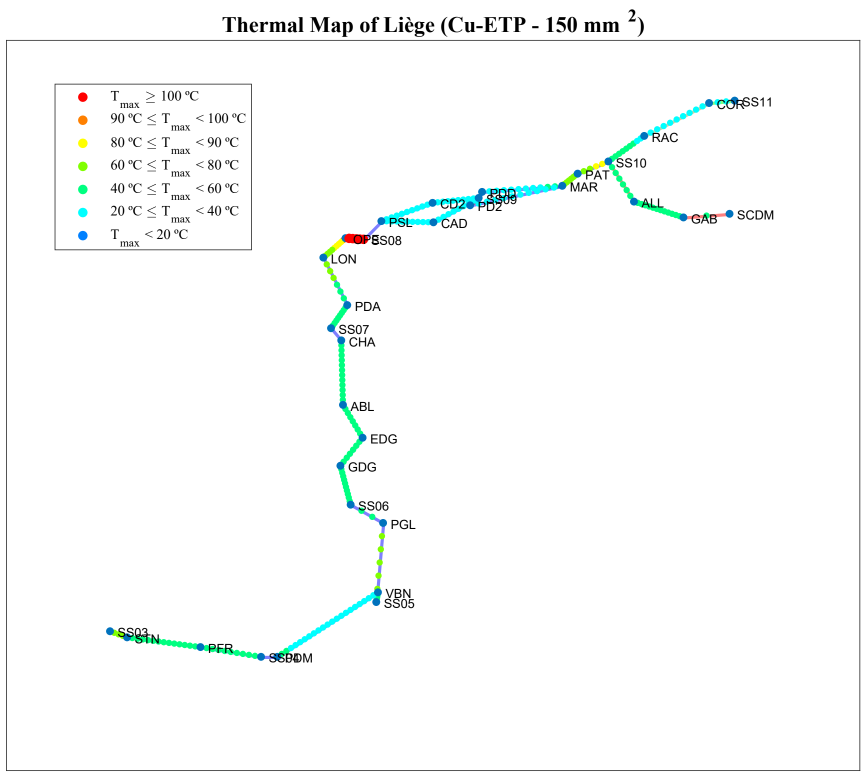

Figure 8-22: Thermal map of Liège's network with V_w=0 m/s and a Cu-ETP conductor with a section of 150 mm^2

In Fig. 8-24, 8-26, 8-25,8-28, 8-27, 8-30 and 8-29, the final results of the line of Liège by using a CuAg0.1 conductor with 150 mm^2 are presented. The results show that lines 16, 17 and 24 are below the operating limits. In these graphs, it is indicated that no point of the network has exceeded temperatures of 100 ºC, being the maximum reached in line 17, in the range of 90-100 °C. In addition, graphs of average temperature along the line are shown.

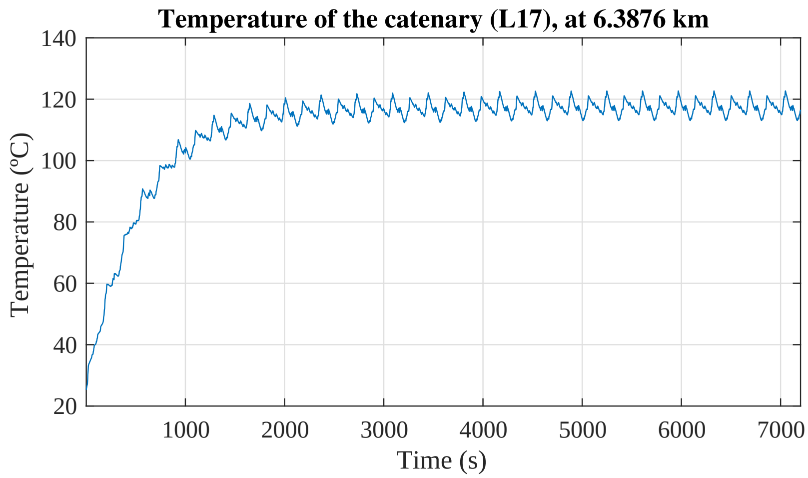

Figure 8-23: Evolution of temperature for line 17, at point 6.388 km

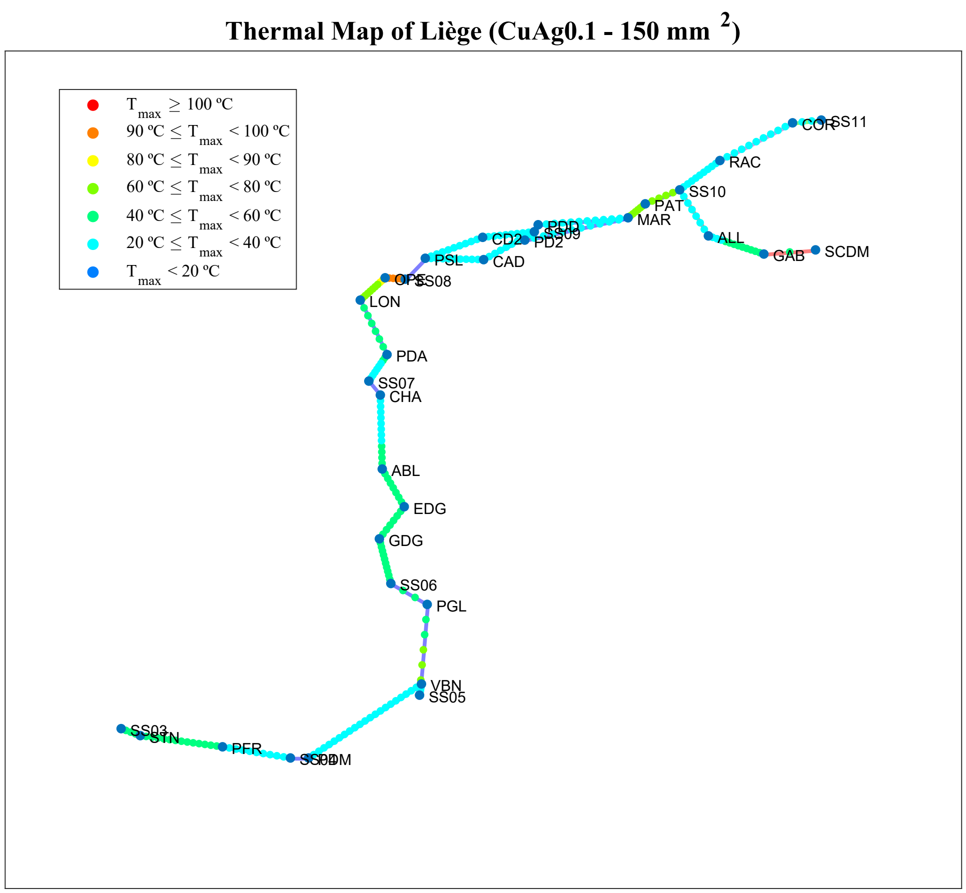

Figure 8-24: Thermal map of Liège's network with V_w=0 m/s and a CuAg0.1 conductor with a section of 150 mm^2

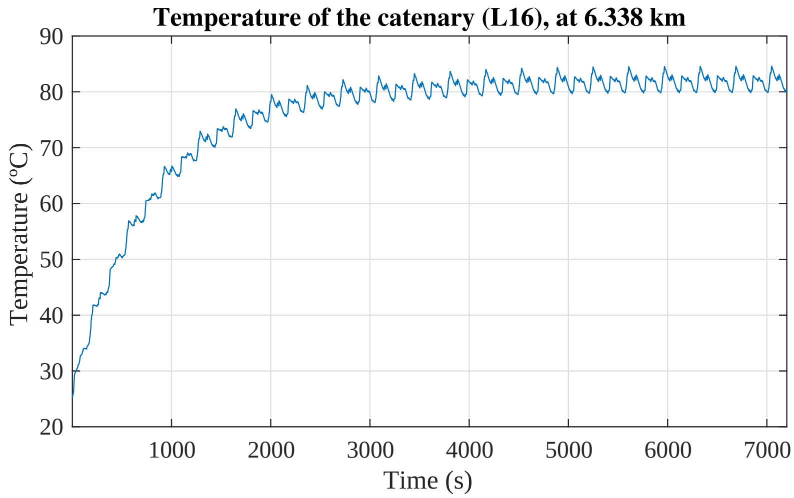

Figure 8-25: Evolution of temperature for line 16, at point 6.338 km

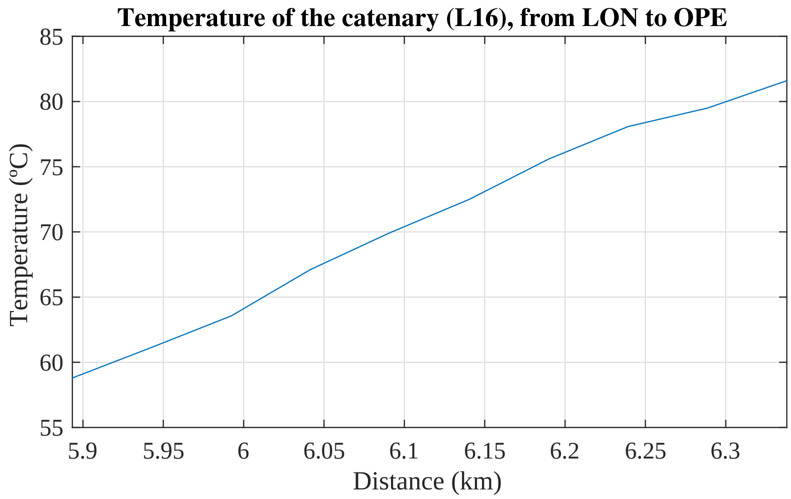

Figure 8-26: Average temperature of L16

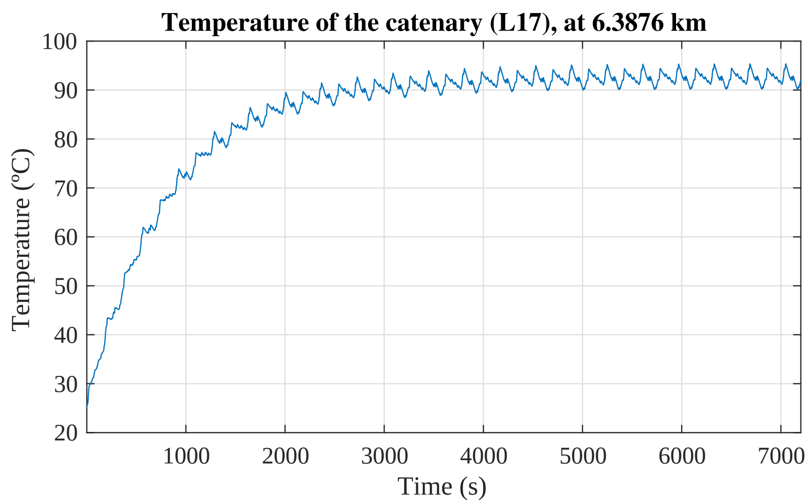

Figure 8-27: Evolution of temperature for line 17, at point 6.388 km

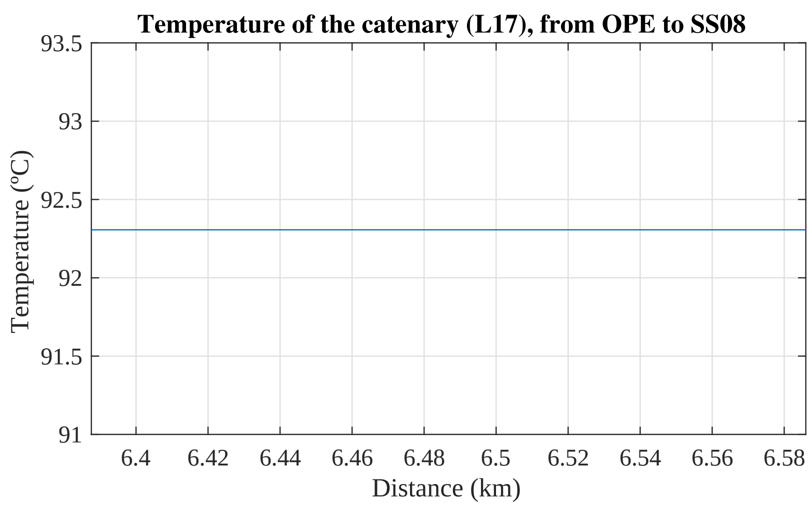

Figure 8-28: Average temperature of L17

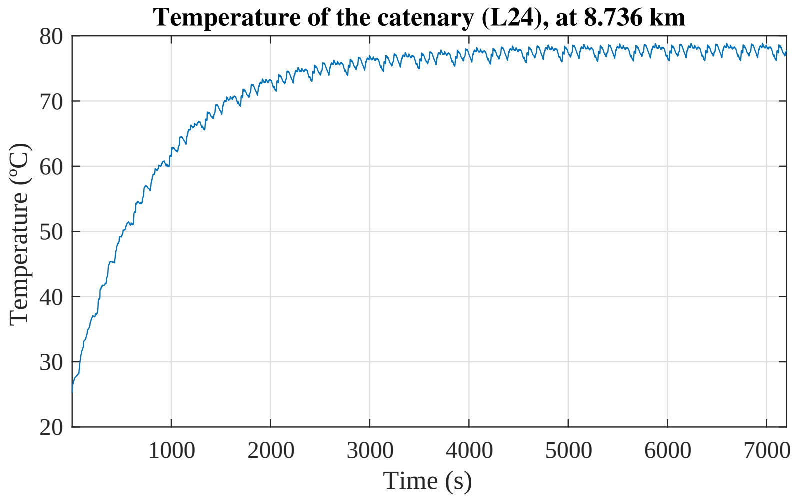

Figure 8-29: Evolution of temperature for line 24, at point 8.736 km

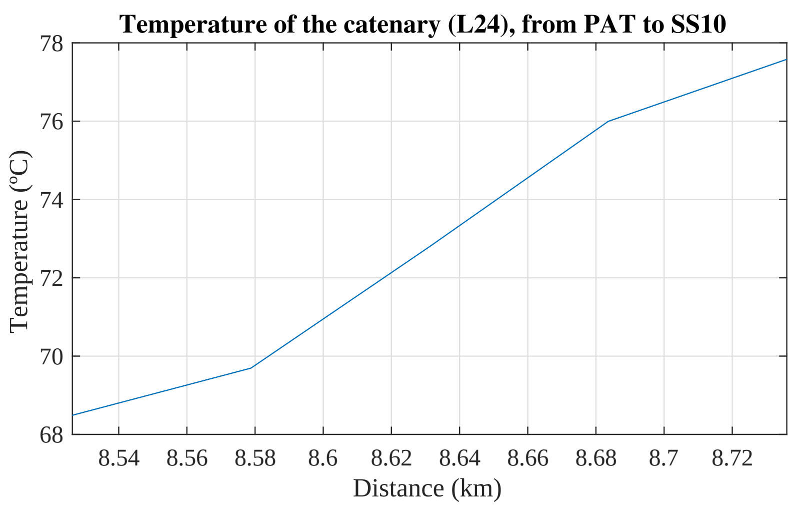

Figure 8-30: Average temperature of L24

Thus, with conductors of CuAg0.1 and sections of 150 mm^2, all points of the lines met the operation requirements. In case of Cu-ETP, all the lines except L16, L17 and L24 fulfilled the permanent operations limits. The most expensive solution is to use for all the lines, conductors of CuAg0.1. In order to reduce the CAPEX, conductors of CuAg0.1 can be installed for lines L16, L17 and L24, and for the rest of the lines, Cu-ETP conductors can be used.

Chapter 9

Conclusions

In this Master's Thesis, a tool, called RailThermal app, has been developed to calculate the thermal behavior of railway systems. The main idea was to create an intuitive software that was able to adjust to any type of grid, calculating the results in a reasonable time. The concept of line sectioning allows to adapt the simulation to the features of the grid, in terms of simulation time and precision.

An explanation of the software code and a description of the main functionalities of the interface have been presented, serving as guide to anyone external to the project. The idea is to describe how RailThermal app performs the calculations and how is the interaction with the software. Then, a verification process of the main parameters of the thermal calculation (q_c, q_r and q_s) have been carried out. The influence of the input parameters of these variables has been analyzed. Finally, a case study of Liège's network was presented. The network has been analyzed from the most unfavorable case and solutions have been provided.

The main goal of this thesis, is to create a base tool for thermal calculation and its subsequent implementation within RailNeos 2.0 software. For now, RailThermal app is an application developed with Matlab that allows to extract temperature results, plotting a thermal map of the lines and showing the temperature evolution over the time and average temperature of the conductors.

Appendix A

Code of RailThemal app

A.1 Thermal_study.m

```
function [OutLine, OutSect, nlines, OutNode, nnodes, Tcat_max] = ...
    Thermal_study(Ts0,Tinf, dbfile, matCat, sectCat, He, Vw, ...
    checkSHG, n_month, n_day, Lat,  L_sect)

%% IMPORT THE RES.DB FILE
connRes = sqlite(dbfile);      % create the connection

%% FUNCTIONS

Conduct_Types = contact_wire;

[Cp, Do, Spc_m, emi, absorp] = InitData(Conduct_Types, matCat, ...
    sectCat);

[OutLine, OutSect, nlines, nSteps, Paths, Lat, Start_Time, ...
    day_year, OutNode, nnodes, Tcat_max] = InputData(connRes, ...
    checkSHG, n_month, n_day, Lat, L_sect);

[Kangle,Coord_nodes] = K_angle(connRes);

```

```
OutSect = InputResults(OutSect, OutLine, Spc_m, Do, Cp, Ts0, ...
    Tinf, nSteps, Kangle, Vw, emi, nlines, Lat, Start_Time, ...
    day_year, He, absorp);

Tcat_max = T_max(OutSect, OutLine, nlines, Tcat_max);
```

A.2 contact_wire.m

```
function Conduct_Types = contact_wire

catnames={'CuAg01','CuETP','CuMg02','CuMg05','CuSn01','CuSn02'};  ...
    % catenary types
dbwire=fullfile(pwd,'Contact_wire.db');
conn_wire=sqlite(dbwire);

for n=1:length(catnames)
sqlcat=['select ', catnames{n},'.Sect, ', catnames{n},'.SpM from ...
    ', catnames{n}];
dataCat=cell2table(fetch(conn_wire,sqlcat));

Sect=double(dataCat{:,1});
SpM=double(dataCat{:,2});

Conduct_Types.(catnames{n}).info=['Sect: Section in mm^2 ...
                ';
'SpM: Specific mass in 10^3 kg/m3'];
Conduct_Types.(catnames{n}).data=[Sect, SpM];
end

% Note: this function import the data of the different types of wires
```

A.3 InitData

```
function [Cp, Do, Spc_m, emi, absorp] = InitData(Conduct_Types, ...
    matCat, sectCat)
%% Initial temperatures

% Note: if Ts is lower than the Tinf the result
% of the convective heat loss will become a complex
% number so it is necessary to avoid that

% Specific heat of typical conductor metal wire
CP.info='Cp in J/(kg C)';
CP.Al=955;             % Aluminum
CP.Cu=423;             % Copper
CP.St=476;             % Steel
CP.Al_clad_st=534;     % Aluminum clad steel

CP.CuETP = 386;
CP.CuAg01 = 386;
CP.CuMg02 = 320;
CP.CuMg05 = 320;
CP.CuSn05 = 377;
CP.CuSn02 = 377;

Cp = CP.(matCat);       % Specific heat of the conductor used
sectCat = str2double(sectCat);

if sectCat == 80
nsectCat = 1;
elseif sectCat == 100
nsectCat = 2;
elseif sectCat == 107
nsectCat = 3;
elseif sectCat == 120
nsectCat = 4;
```

```
elseif sectCat == 150
nsectCat = 5;
end

Do = sqrt(Conduct_Types.(matCat).data(nsectCat,1)*4/pi)/1000;      ...
    % Diameter of the catenary in m
Spc_m = Conduct_Types.(matCat).data(nsectCat,2)*1000;               ...
    % Specific mass in kg/m^3

%% Variables for the radiated heat loss
emi = 0.3;           % emissivity
%% Variables for the solar heat gain
absorp = 0.3;        % absorptivity
```

A.4 InputData.m

```
function [OutLine, OutSect, nlines, nSteps, Paths, Lat, ...
    Start_Time, day_year, OutNode, nnodes, Tcat_max] = ...
    InputData(connRes, checkSHG, n_month, n_day, Lat, L_sect)

%% ---- LOSSES OF THE CATENARY ----
sqlOutLine = 'select OUT_Line.Stp, OUT_Line.Line ,OUT_Line.P_Los, ...
    OUT_Line.Length from OUT_Line, Line_t where ...
    OUT_Line.Line=Line_t.ID';

% Import all the data of the catenay in a table
OutLine_table = cell2table(fetch(connRes,sqlOutLine));
% Change the name of the variables
OutLine_table.Properties.VariableNames = {'Step' 'Line' 'PLosses' ...
    'Length'};
% Vector with the value of the steps
Step = double(OutLine_table.Step(:,1));
% Vector with the number of the lines
Line = double(OutLine_table.Line(:,1));
% Vector with the value of the PLosses
PLosses = double(OutLine_table.PLosses(:,1));
Length = double(OutLine_table.Length(:,1));

% Vector that contents all the lines at the first instant
nlines = double(OutLine_table.Line((find(OutLine_table.Step==1))));
% Select only the unique values, removing lines that appear twice
% (lines in which there is at least one train circulating)
nlines = unique(nlines);
% Each step
nSteps = unique(Step);

for n = 1:length(nlines)
    mline = nlines(n);
    linename = ['L',num2str(mline)];
```

```
    OutLine.data.(linename) = [ Step(Line==mline),...
    PLosses(Line==mline), Length(Line==mline)];
end

% Note: it carries out the importation of the data of
% the catenary in the struct OutLine.
% OutLine is composed by n fields (L1,L2,...Ln) and
% insight each field there is an array of three
% colums: | Step (s) | PLosses (kW) | Length (km)

%% ---- VARIABLES FOR SOLAR HEAT GAIN ----
if checkSHG == 1
    n_year = 2019;
    date = [n_year n_month n_day];
    date = datetime(date);     % Date in the format: 02-Jan-2012
    day_year = day(date,'dayofyear');    % Day of the year
else
    Lat = 0;
    day_year = 0;
end
%% ---- LINE SECTIONING ----

sqlOutLine = 'select Line_t.ID, Line_t.Length_p, Line_t.Src, ...
    Line_t.Dst from Line_t';

Line_table = cell2table(fetch(connRes,sqlOutLine));
% Table: | ID of line | Length (km) | Src | Dst
Line_table.Properties.VariableNames = {'Line' 'Length' 'Src' 'Dst'};

Line_ID = double(Line_table.Line(:,1));          % Vector with the ...
    ID of the line
Length_Line = double(Line_table.Length(:,1));    % Vector with the ...
    length of the line
Src = double(Line_table.Src(:,1));               % Vector with the ...
    source node
```

```
Dst = double(Line_table.Dst(:,1));                % Vector with the ...
    destination node
%L_sect = 0.100;                                  % Length of each ...
    section in km
nnodes = [];                                      % Index of the nodes

for n = 1:length(nlines)
    linename = ['L',num2str(nlines(n))];
    L_line = round(Length_Line(Line_ID==nlines(n)),3);  % Length ...
        of the line in km
    OutLine.LengthReal.(linename) = L_line;             % Length ...
        of the lines
    OutLine.SrcDst.(linename) = [Src(Line_ID==nlines(n)), ...
        Dst(Line_ID==nlines(n))];        % Source node and ...
        destination node
    nodesSrcDst = [Src(Line_ID==nlines(n)), Dst(Line_ID==nlines(n))];
    nnodes = [nnodes , nodesSrcDst];

    if round(L_line,3) ≥ round(L_sect,3)
        No_sect = round(L_line/L_sect);                     % The ...
            number of the section must be an integer
        L_sect_real = L_line/No_sect;                       % ...
            Real Length of each section in km
        coords = linspace(L_sect_real,L_line,No_sect)';     % ...
            Vector with the position of each point
        OutLine.Sect.(linename) = coords ;
    else
        OutLine.Sect.(linename) = L_line;
    end
end
% Note: this function add to Outline a field called Sect in which the
% division points for each line are defined

% Introduce the source and destination nodes
nnodes = unique(nnodes)';                                   % ...
    only nodes of connected lines have been considered
```

```
%% REMOVE PARALLEL LINES (WHEN THEY HAVE SAME SOURCE AND ... DESTINATION SOURCE)

Linefn = fieldnames(OutLine.SrcDst);                           % ... Fields of Outline.SrcDst
nlines_aux = nlines;
L_SrcDst_num_all = [];

for n = 1:length(nlines_aux)
    linename = ['L',num2str(nlines_aux(n))];
    SrcDst_Aux = OutLine.SrcDst.(linename);
    tf = cellfun(@(x) isequal(OutLine.SrcDst.(x), SrcDst_Aux), ... Linefn);
    if sum(tf)≥2
        L_SrcDst_num_aux = nlines(tf);
        L_SrcDst_num = L_SrcDst_num_aux(end,1);
        L_SrcDst_num_all = [L_SrcDst_num_all; L_SrcDst_num];
    end
end

if isempty(L_SrcDst_num_all)== 0
    L_SrcDst_num_all = unique(L_SrcDst_num_all);
    if length(L_SrcDst_num_all)>1
        for n = 1:length(L_SrcDst_num_all)
            linename = ['L',num2str(L_eqSrcDst_num_all(n))];
            OutLine.data = rmfield(OutLine.data, linename);
            OutLine.SrcDst = rmfield(OutLine.SrcDst, linename);
            OutLine.Sect = rmfield(OutLine.Sect, linename);
            OutLine.LengthReal = rmfield(OutLine.LengthReal, ... linename);
            nlines(strcmp(Linefn,L_eqSrcDst_all))=[];
        end
    else
        linename = ['L',num2str(L_SrcDst_num_all)];
        OutLine.data = rmfield(OutLine.data, linename);
```

```
        OutLine.SrcDst = rmfield(OutLine.SrcDst, linename);
        OutLine.Sect = rmfield(OutLine.Sect, linename);
        OutLine.LengthReal = rmfield(OutLine.LengthReal, linename);
        nlines(nlines(:,1)==L_SrcDst_num_all)=[];
    end
end

%% ---- PATH LINES ----
sqlnpaths='select Path_Line_t.Path , Path_Line_t.Line, ...
    Path_Line_t.Dir from Path_Line_t order by Path_Line_t.Path';

Paths_table=cell2table(fetch(connRes,sqlnpaths));
Paths_table.Properties.VariableNames = {'Path' 'Line' 'Dir'};

Path=double(Paths_table.Path(:,1));
Line=double(Paths_table.Line(:,1));
Dir=double(Paths_table.Dir(:,1));

Line_unique=unique(Line);

nPath=unique(Path);                         % number of the paths

Lines_disc=setdiff(Line_unique, nlines);    % Line or lines ...
    disconnected. It is
% compared the lines that appear in
% nlines and all the lines that compose
% the paths. The aim is to remove the
% Paths with lines that are disconneted

Path_rmvd = [];                             % Paths that are ...
    removed due to some line of this path has been disconnected

for n = 1:length(nPath)
    pathname = ['P',num2str(nPath(n))];
    Paths.(pathname) = [Line(Path==nPath(n)), Dir(Path==nPath(n))];
    if any(ismember(Paths.(pathname)(:,1),Lines_disc))==1
```

```
        Paths = rmfield(Paths, pathname);
        Path_rmvd = [Path_rmvd; nPath(n)];
    end
end

Path_names = fieldnames(Paths);       % Number of paths that are ...
    connected

% Note I: function ismember defines a logical array. 1 in the ...
    case, the value of the
% line disconnected was the same that the line of the path. By ...
    using any,
% the logical array is reduced to one figure. If this value is ...
    equal to
% 1, the field of the path is removed

% Note II: Import the data of the paths in the variable ...
    Path_Line_t. Path_Line_t
% consists of fields (called Path n), where the number of the ...
    lines that compose
% the path are stored


%% ---- PLOT GRAPH NODES + DECLARE TCAT_MAX.SRC VARIABLES ----

sqlOutNode = 'select Node.ID, Node.Pos_X, Node.Pos_Y , Node.Name ...
    from Node';
% Import all the data of the catenay in a table
OutNode_table = cell2table(fetch(connRes,sqlOutNode));
% Change the name of the variables
OutNode_table.Properties.VariableNames = {'Node' 'Pos_X' 'Pos_Y' ...
    'Name'};

Node = double(OutNode_table.Node(:,1));          % Number of the node
Pos_X = double(OutNode_table.Pos_X(:,1));        % Pos X of the node
Pos_Y = double(OutNode_table.Pos_Y(:,1));        % Pos Y of the node
```

```
Name = OutNode_table.Name(:,1);                    % Name of the node

for n = 1:length(nnodes)
    nodename = ['N',num2str(nnodes(n))];
    % ---- OutNode.Pos_XY & OutNode.Name ----------------
    % Coordinate X of the node
    OutNode.Pos_XY.(nodename) = [Pos_X(Node==nnodes(n)) ...
        Pos_Y(Node==nnodes(n))];
    % Name of the node
    OutNode.Name.(nodename) = Name(Node==nnodes(n));

    % ---- Tcat_max.SrcDst ------------------------------
    Tcat_max.SrcDst.(nodename) = [0,0];         % Init the ...
        variable Tcat_max
end

%% COLOR OF THE PATHS + LENGTH OF THE LINES SCALED TO THE GRAPH +
%% REAL PK + INIT TCAT_MAX FOR THE DIFFERENT INTERVALS
% Introduce the colors of each path
sqlColors = 'select Line_t.ID, Path_Line_t.Path from Line_t,...
Path_Line_t where Line_t.ID = Path_Line_t.Line ';

Line_Path = cell2table(fetch(connRes,sqlColors));
Line_Path.Properties.VariableNames = {'Line' 'Paths'};

LinePath = double(Line_Path.Line(:,1));     % number of lines
numPath = double(Line_Path.Paths(:,1));     % number of the path

% Remove paths that containt a disconnected line
for n = 1:length(Path_rmvd)
    LinePath(numPath(:,1)==Path_rmvd(n,1))= [];
    numPath(numPath(:,1)==Path_rmvd(n,1)) = [];
end

% variable with path and its color. First column is for the number
% of the path and the 2:end columns are the color code
```

```
Path_clr = zeros(1,4);

colorsP = [0 0 1; 0 1 0; 1 0 0; 0 1 1; 1 0 1; 1 1 0];
ncolorP = 0;

for n = 1:length(nlines)
    linename = ['L', num2str(nlines(n))];
    % ---- Length of the lines scaled to the graph-------------

    nodenameSrc = ['N', num2str(OutLine.SrcDst.(linename)(1,1))];
    nodenameDst = ['N', num2str(OutLine.SrcDst.(linename)(1,2))];

    OutLine.LengthGraph.(linename) = ...
       pdist([OutNode.Pos_XY.(nodenameSrc);...
    OutNode.Pos_XY.(nodenameDst)],'euclidean');

    %----------------------------------------
    % ---- Color of the Paths --------------
     % Num of the path. Note that was removed such lines that are ...
        disconnected
    Path_num_aux = numPath(LinePath(:,1)==nlines(n),1);

    % I) Use a generic color for the first 6 paths
        if ncolorP < 6
            % I.1) If the line is not associated to any path
            if isempty(Path_num_aux)==1
                Path_num = 0;
                % Color of the line
                OutLine.Colors.(linename) = rand(1,3);
                % Path of the line selected. (0 means that the ...
                    line hasn't path )
                OutLine.PathSlctd.(linename) = Path_num;
                % I.2) If the line is associated to some path
            else
                % Only one path is selected, the others are neglected
                Path_num = Path_num_aux(1,1);
```

```
                % Path of the line selected
                OutLine.PathSlctd.(linename) = Path_num;
                if any(Path_clr(:,1)==Path_num)==1
                    OutLine.Colors.(linename) = ...
                       Path_clr(Path_clr(:,1)==Path_num,2:end);
                else
                    ncolorP = ncolorP+1;
                    OutLine.Colors.(linename) = colorsP(ncolorP,:);
                    Path_clr = [Path_clr; Path_num, ...
                       OutLine.Colors.(linename)];
            end
        end
    % II) Use a random color if there would be more than 6 paths
    else
        % II.1) If the line is not associated to any path
        if isempty(Path_num_aux)==1
            Path_num = 0;
            OutLine.Colors.(linename) = rand(1,3);
            OutLine.PathSlctd.(linename) = Path_num; ...
                                    % Path of the line ...
               selected. (0 means that the line hasn't assigned ...
               path )
        % II.2) If the line is associated to some path
        else
            Path_num = Path_num_aux(1,1); ...
                                       % Only one path ...
               is selected, the others are neglected
            OutLine.PathSlctd.(linename) = Path_num; ...
                               % Path of the line selected
            if any(Path_clr(:,1)==Path_num)==1
                OutLine.Colors.(linename) = ...
                   Path_clr(Path_clr(:,1)==Path_num,2:end);
            else
                OutLine.Colors.(linename) = rand(1,3);
                Path_clr = [Path_clr; Path_num, ...
                   OutLine.Colors.(linename)];
```

```
            end
        end
    end
    %--------------------------------------
    % ---- Real Start PK ------------------
    k = 1; ...
                                                                                   ...
       % Counter for calculatinf the Real Start PK
    i_PK = 0; ...
                                                                                ...
       % PK Counter of the path
    if Path_num≠0
        pathname = ['P',num2str(Path_num)];
        Lines_Path_aux = Paths.(pathname); ...
                                               % Lines that belong ...
           to the path

        while Lines_Path_aux(k,1) ≠ nlines(n)
            linename_aux = ['L', num2str(Lines_Path_aux(k,1))];
            i_PK = i_PK + OutLine.LengthReal.(linename_aux);
            k = k+1;
            if k > length(Lines_Path_aux)
                continue
            end
        end

        OutLine.Real_PK.(linename) = i_PK + ...
           OutLine.Sect.(linename);         % Real PK of the lines
        else
        OutLine.Real_PK.(linename) = OutLine.Sect.(linename); ...
                            % if there is not a path join to the ...
           line the initial PK of the line is 0

    end
    %---------------------------------------------------------
    % ---- Init Tcat_max for the different intervals --------
```

```
    Tcat_max.Higher100.(linename) = [];
    Tcat_max.fr90to100.(linename) = [];
    Tcat_max.fr80to90.(linename) = [];
    Tcat_max.fr60to80.(linename) = [];
    Tcat_max.fr40to60.(linename) = [];
    Tcat_max.fr20to40.(linename) = [];
    Tcat_max.Lower20.(linename) = [];
end



%% ---- OUTSECT -> LINE SECTIONING + LOSSES OF THE CATENARY ----

for n = 1:length(nlines)
    % Init variables
    linename = ['L',num2str(nlines(n))];
    OutSect.(linename)=[];
    Sect_Length = OutLine.Sect.(linename)(1,1)-0; ...
                                         % Length of the sections for ...
       each line in km
    num_sect = length(OutLine.Sect.(linename)(:,1)); ...
                                      % Total number of sections in Ln
    Length_Step = ones(num_sect,1); ...
                                                      % Vector of ...
       ones with the length of the total number of sections

    for m = 1:length(nSteps)
        % Init variables
        PLosses_OutSect = [];
        % I) There isn't any train in the line
        if length(OutLine.data.(linename)...
        (OutLine.data.(linename)(:,1)==m,1))==1...
        OutSect.(linename)= [ OutSect.(linename);
        m*Length_Step, OutLine.Sect.(linename)(:,1), ...
           OutLine.data.(linename)...
```

```
(OutLine.data.(linename)(:,1)==m,2)*Length_Step/num_sect];

% II) At least one train is circulating along the line
else
data_Sec_Aux = [OutLine.data.(linename)...
(OutLine.data.(linename)(:,1)==m,2),...
OutLine.data.(linename)(OutLine.data.(linename)(:,1)==m,3)];
 % Aux variable: | PLosses (kW) | Length (km) |, for the ...
     dynamic line in a instant
data_Sec_Aux(:,3) = data_Sec_Aux(:,1)./data_Sec_Aux(:,2); ...
           % Plosses in kW/km
start_seg = 0;
mid_point =0;

for k = 1:length(data_Sec_Aux(:,1))

% FOR THE FIRST SEGMENT OF THE LINE
if k==1
if round(Sect_Length,3)<round(data_Sec_Aux(1,2),3)
% Case 1) A train is circulating in the first section:
% |--o--|-----|-----|
mid_point = data_Sec_Aux(k,2)+mid_point;
ones_seg_0 = round(OutLine.Sect.(linename)...
(:,1),3)>round(start_seg,3) &...
round(OutLine.Sect.(linename)(:,1),3)<...
round(mid_point,3);
% vector of ones with sections lower than the mid point ...
    and higher than the start seg point

end_seg = max(OutLine.Sect.(linename)(ones_seg_0)); ...
                         % Coordinates of the last ...
    section with a length lower than the midpoint

PLoss_seg = ...
    ones(sum(ones_seg_0),1)*Sect_Length*data_Sec_Aux(k,3); ...
             % Plosses of each section of segment 1 in kW
```

```
% Mid point 1
ones_seg_1 = round(OutLine.Sect.(linename)...
(:,1),3)≥round(mid_point,3);                    % ... Vector of ones with sections higher than the midpoint
start_seg = min(OutLine.Sect.(linename)(ones_seg_1)); ...
                            % Start seg point of the next ... iteration

start_seg_aux = min(start_seg, ...
mid_point+data_Sec_Aux(k+1,2));               % Min of the ... star_seg and the next mid_point for the case that the ... next midpoint will be in the same segmente than the ... previous midpoint

PLoss_midpoint = (mid_point-end_seg)*data_Sec_Aux(k,3)+...
(start_seg_aux-mid_point)*data_Sec_Aux(k+1,3);

% Join join the result of the segment and the middle
% segment
PLosses_OutSect_Aux = [PLoss_seg; PLoss_midpoint];

elseif round(Sect_Length,3)≥round(data_Sec_Aux(1,2),3)
mid_point = data_Sec_Aux(k,2);
ones_seg_1 = round(OutLine.Sect.(linename)(:,1),3)...
≥round(mid_point,3);                          % Vector of ... ones with sections higher than the midpoint
end_seg = 0;
start_seg = min(OutLine.Sect.(linename)(ones_seg_1)); ...
                            % Start seg point of the next ... iteration


PLoss_midpoint = (mid_point-0)*data_Sec_Aux(k,3)...
+(start_seg-mid_point)...
*data_Sec_Aux(k+1,3);
```

```
        PLosses_OutSect_Aux = PLoss_midpoint;
        end
        % FOR THE INTERMEDIATE SEGMENTS
        elseif k≠length(data_Sec_Aux(:,1)) && k≠1

        % Update
        mid_point_1 = data_Sec_Aux(k,2)+mid_point; ...
                                                                      ...
           % second midpoint of the group
        limit_seg = min(OutLine.Sect.(linename)...
        (round(OutLine.Sect.(linename)...
        (:,1),3)≥round(mid_point_1,3)));          % final point ...
           for the next group of segments

        % Case 1) Two trains are not in consecutive sections:
        % |--o--|-----|--o--|
        if length(OutLine.Sect.(linename)...
        (round(OutLine.Sect.(linename)(:,1),3)...
        ≥round(mid_point,3) & round(OutLine.Sect.(linename)...
        (:,1),3)≤(round(limit_seg,3))))>2
        % Segment of the line
        mid_point = mid_point_1;
        ones_seg_0 = round(OutLine.Sect.(linename)(:,1),3)...
        >round(start_seg,3) & round(OutLine.Sect.(linename)...
        (:,1),3)<round(mid_point,3);              % vector of ...
           ones with sections lower than the mid point and higher ...
           than the start seg point

        end_seg = max(OutLine.Sect.(linename)(ones_seg_0)); ...
                                   % Coordinates of the last ...
           section with a length lower than the midpoint

        PLoss_seg = ones(sum(ones_seg_0),1)*Sect_Length*...
        data_Sec_Aux(k,3);          % Plosses of each section of ...
           segment 1 in kW

```

```
% Mid point 1
ones_seg_1 = round(OutLine.Sect.(linename)(:,1),3)...
≥round(mid_point,3);                          % Vector of ...
   ones with sections higher than the midpoint
start_seg = min(OutLine.Sect.(linename)(ones_seg_1)); ...
                         % Start seg point of the next ...
   iteration

start_seg_aux = min(start_seg, mid_point+...
data_Sec_Aux(k+1,2));                % Min of the star_seg ...
   and the next mid_point for the case that the next ...
   midpoint will be in the same segmente than the ...
   previous midpoint

PLoss_midpoint = (mid_point-end_seg)*data_Sec_Aux...
(k,3)+(start_seg_aux-mid_point)*data_Sec_Aux(k+1,3);

% Join join the result of the segment and the middle
% segment
PLosses_OutSect_Aux = [PLoss_seg; PLoss_midpoint];

% Case 2) Two trains are in consecutive sections:
% |--o--|--o--|
elseif length(OutLine.Sect.(linename)(round...
(OutLine.Sect.(linename)(:,1),3)≥round(mid_point,3) &...
round(OutLine.Sect.(linename)(:,1),3)≤...
(round(limit_seg,3))))==2
mid_point = mid_point_1; ...
                                                      % ...
   midpoint for the next iteration
ones_seg_1 = round(OutLine.Sect.(linename)(:,1),3)≥...
round(mid_point,3);
% Vector of ones with sections higher than the midpoint
start_seg = min(OutLine.Sect.(linename)(ones_seg_1)); ...
                         % Start seg point of the next ...
   iteration
```

```

        end_seg = end_seg + Sect_Length;

        s1 = mid_point-end_seg; ...
                                                        % ...
            Length of the subsegment 3 (km)
        s2 = start_seg-mid_point; ...
                                                        % ...
            Length of the subsegment 4 (km)

        % Join the result of the s1 and s2
        PLosses_OutSect_Aux = s1*(data_Sec_Aux(k,3))+s2*...
        (data_Sec_Aux(k+1,3));  % Plosses of the s1 and s2 (kW

        % Case 3) Two train are in the same section:
        % |-o-o-|
        elseif length(OutLine.Sect.(linename)(round...
        (OutLine.Sect.(linename)(:,1),3)≥...
        round(mid_point,3) &...
        round(OutLine.Sect.(linename)(:,1),3)≤...
        (round(limit_seg,3))))==1
        mid_point = mid_point_1;
        ones_seg_1 = round(OutLine.Sect.(linename)(:,1),3)...
        ≥round(mid_point,3);     % Vector of ones with sections ...
            higher than the midpoint
        start_seg = min(OutLine.Sect.(linename)(ones_seg_1)); ...
                                    % Start seg point of the next ...
            iteration

        PLosses_OutSect(end,1) = PLosses_OutSect_Aux(end,end)+...
        (start_seg-mid_point)*data_Sec_Aux(k+1,3); % The PLosses ...
            are added to the previous segment
        PLosses_OutSect_Aux = [];
        end

        % FOR THE LAST SEGMENT
```

```
        else
        % Segment of the line
        end_point = sum(data_Sec_Aux(:,2));
        ones_seg_0 = OutLine.Sect.(linename)(:,1)>start_seg &...
         OutLine.Sect.(linename)(:,1)≤(end_point+1);  % vector of ...
             ones with sections lower than the mid point and ...
             higher than the start seg point

        PLoss_seg = ...
            ones(sum(ones_seg_0),1)*Sect_Length*data_Sec_Aux(k,3); ...
                                               % Plosses of each ...
            section of segment 1 in kW
        PLosses_OutSect_Aux = PLoss_seg;
        end

        PLosses_OutSect = [PLosses_OutSect; PLosses_OutSect_Aux];

        end
        % Update the data
        OutSect.(linename) = [OutSect.(linename); Length_Step*m,...
         OutLine.Sect.(linename), PLosses_OutSect];
        end

    end
end

%% ---- START TIME ----

sqlCfg = 'select Cfg.Start_Time from Cfg';

Cfg_table = cell2table(fetch(connRes,sqlCfg)); ...
                                   % Import the time in which the ...
    simulation has started
Cfg_table.Properties.VariableNames = {'Star_Time'}; ...
                              % Change the name of the variables

```

```
Start_Time = double(Cfg_table.Star_Time(:,1)); ...
                                    % Value of the start time in seconds
```

A.5 K_angle.m

```
function  [Kangle,Coord_nodes] = K_angle(connRes)

sqlCoord_Src = 'select Line_t.ID, Line_t.Src, Node.Pos_X, ...
    Node.Pos_Y from Node, Line_t where Node.ID = Line_t.Src';
sqlCoord_Dst = 'select Line_t.ID, Line_t.Dst, Node.Pos_X, ...
    Node.Pos_Y from Node, Line_t where Node.ID = Line_t.Dst';

Coord_table_Src = cell2table(fetch(connRes, sqlCoord_Src));
Coord_table_Dst = cell2table(fetch(connRes, sqlCoord_Dst));

Coord_table_Src.Properties.VariableNames = {'Line' 'Src' 'Pos_X' ...
    'Pos_Y'};
Coord_table_Dst.Properties.VariableNames = {'Line' 'Dst' 'Pos_X' ...
    'Pos_Y'};

Line = double(Coord_table_Src.Line(:,1)); ...
                              % Line ID
Src = double(Coord_table_Src.Src(:,1)); ...
                                % Source node ID
Pos_X_Src = double(Coord_table_Src.Pos_X(:,1)); ...
                        % Pos X of the source node
Pos_Y_Src = double(Coord_table_Src.Pos_Y(:,1)); ...
                        % Pos Y of the source node
Dst = double(Coord_table_Dst.Dst(:,1)); ...
                                % Destination node ID
Pos_X_Dst = double(Coord_table_Dst.Pos_X(:,1)); ...
                        % Pos X of the destination node
Pos_Y_Dst = double(Coord_table_Dst.Pos_Y(:,1)); ...
                        % Pos Y of the destination node
m = atan((Pos_Y_Dst-Pos_Y_Src)./(Pos_X_Dst-Pos_X_Src)).*180/pi; ...
        % Calculate the direction of the lines in rad
normalizeDeg = @(x)(-mod(-x+180,360)+180); ...
                             % Function that normalize the angles ...
```

```
    to [-180,180] range. Note the function mode calculate the ...
    rest, where the first value is the dividen and the second the ...
    divisor
phi_L = normalizeDeg(m); ...
                                                    % angle direction ...
    normalized between -180 and 180

Coord_nodes = [Line, Src, Pos_X_Src, Pos_Y_Src, Dst, Pos_X_Dst, ...
    Pos_Y_Dst, phi_L];

% Note: Coord_nodes is a variable with the following structure:
% | Line | Src | Pos_X_Src | Pos_Y_Src | Dst | Pos_X_Dst | ...
    Pos_Y_Dst | direction |

phi_WL = 0:30*pi/180:90*pi/180; ...
                                                    % angle between ...
    the direction of the wind and the direction of the line


phi_W=[];               % Wind angle:

K_angle_funct = @(phi) ...
    1.194-cos(phi)+0.194*cos(2*phi)+0.368*sin(2*phi);    % ...
    function to calculate the wind direction factor. Remember that ...
    this value is between 0 and 1

for n = 1:length(phi_L)
    m = phi_L(n,1)*pi/180*ones(length(phi_WL),1);
    phi_W =[phi_W ,  phi_WL'+m];
end

K_angle_array = K_angle_funct(phi_WL);

    for n = 1:length(Line) ...
                                                              % ...
        Loop to ensure that the value of K_angle of a specific ...
```

```
        line is saved correctly
    m = Line(n,1); ...
                                                                         % ...
        Index of the line
    linename = ['L', num2str(m)];
    Kangle.(linename) = phi_W(:,n);
end

Kangle.phi_WL = K_angle_array;

% Note: the K_angle structure has fields with name L1, L2 ... ...
    where the
% value of the wind direction is saved. And in the field phi_WL
% the value of the K_angle going from phi_WL equal to zero to ...
    phi_WL equal
% to 90 is calculated

% Example in degrees, Variable phi_W: the wind angle is calculted ...
    summing,
% phi_L+phi_WL.
```

A.6 InputResults.m

```
function OutSect = InputResults(OutSect, OutLine, Spc_m, Do, ...
    Cp, Ts0, Tinf, nSteps, Kangle, Vw, emi, nlines, Lat, ...
    Start_Time, day_year, He, absorp)

for n = 1:length(nlines)
    % Init variables
    Ts_Tot = [];            % Vector with the temperatures of ...
        each iteration
    qc_Tot = [];            % Vector with the heat convection ...
        loss in W
    qr_Tot = [];            % Vector with the radiated heat ...
        loss in W
    qs_Tot = [];            % Vector with the solar heat gain ...
        in W
    count = 1;              % Hour index, for each hour the ...
        hour angle must be increased in 15 degrees

    % Used variables
    linename = ['L',num2str(nlines(n))];
    Sect_L = OutLine.Sect.(linename)(1,1)*1000;
    % number of section for the line
    nSect = ...
        length(OutSect.(linename)((OutSect.(linename)(:,1)==1),3));
    Ts= Ts0*ones(nSect,1); ...
                                                            % ...
        Vector with T0

    % Calculate: 1/(m*Cp)
    K_mCp = 1/(Spc_m*Sect_L*pi*Do^2/4*Cp); ...
                                               % Factor 1/(mCp) in ...
        C/J or C/(Ws)

    for m = 1:length(nSteps)
```

```
        % Convective heat loss in kW
        qc = Convect_Heat_Loss(Kangle,Ts,Tinf,Do,Vw,He)*Sect_L;

        % Radiated heat loss in kW
        qr = Radiated_Heat_Loss(Ts,Tinf,Do,emi)*Sect_L;

        % Solar heat gain in kW
        qs = Solar_Heat_Gain(Lat, Start_Time, day_year, He,...
        absorp, Do, nlines, nSect, count)*Sect_L;

        % Sum = (Plosses + qs - qc - qr) in W
        Sum = (OutSect.(linename)((OutSect.(linename)...
        (:,1)==m),3)-qc-qr+qs)*1000;

        % Variation of the temperature for interval
        dT_dt = K_mCp*Sum;

        qc_Tot = [qc_Tot;qc];
        qr_Tot = [qr_Tot;qr];
        qs_Tot = [qs_Tot;qs];
        % Vector with the final temperature of this instant
        Ts_1 = Ts+dT_dt;
        % Vector with the initial temperature of the next instant
        Ts = Ts_1;
        % Vector with the temperatures of each iteration
        Ts_Tot = [Ts_Tot;Ts_1];

        % Update the counter for the hour angle, each hour of ...
            simulation
        if m > count*3600
            count = count + 1;
        end
    end

    % Add the column of the temperatures
```

```
        OutSect.(linename)(:,4) = qc_Tot;
        OutSect.(linename)(:,5) = qr_Tot;
        OutSect.(linename)(:,6) = qs_Tot;
        OutSect.(linename)(:,7) = Ts_Tot;


    end
```

A.7 T_max.m

```
function Tcat_max = T_max(OutSect, OutLine, nlines, Tcat_max)

for n=1:length(nlines)
    linename = ['L',num2str(nlines(n))];

    % Define the maximum temperature of the line
    for m = 1:length(OutLine.Sect.(linename))
        Tmax = max(OutSect.(linename)(OutSect.(linename)(:,2) == ...
            OutLine.Sect.(linename)(m,1),7));
        Instant = OutSect.(linename)(OutSect.(linename)(:,7) == ...
            Tmax(1,1) & OutSect.(linename)(:,2) == ...
            OutLine.Sect.(linename)(m,1),1);
        Instant = Instant(1,1); % this was made due to in some ...
            situations, there were several instants with the max ...
            temp for the same section
        Position = OutLine.Sect.(linename)(m,1);
        T_H120 = ...
            length(OutSect.(linename)(OutSect.(linename)(:,7) > ...
            120 & OutSect.(linename)(:,2) == ...
            OutLine.Sect.(linename)(m,1),1));
        Position_PK_Real = OutLine.Real_PK.(linename)(m,1);

        if Tmax(1,1)≥100
            Tcat_max.Higher100.(linename) = ...
                [Tcat_max.Higher100.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<100 && Tmax(1,1)≥90
            Tcat_max.fr90to100.(linename) = ...
                [Tcat_max.fr90to100.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<90 && Tmax(1,1)≥80
            Tcat_max.fr80to90.(linename) = ...
                [Tcat_max.fr80to90.(linename); Instant Position ...
```

```
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<80 && Tmax(1,1)≥60
            Tcat_max.fr60to80.(linename) = ...
                [Tcat_max.fr60to80.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<60 && Tmax(1,1)≥40
            Tcat_max.fr40to60.(linename) = ...
                [Tcat_max.fr40to60.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<40 && Tmax(1,1)≥20
            Tcat_max.fr20to40.(linename) = ...
                [Tcat_max.fr20to40.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        elseif Tmax(1,1)<20
            Tcat_max.Lower20.(linename) = ...
                [Tcat_max.Lower20.(linename); Instant Position ...
                Tmax T_H120 Position_PK_Real];
        end

    end

    % Define the maximum temperature of the source and ...
        destination nodes
    nodeSrc = OutLine.SrcDst.(linename)(1,1);
    nodeDst = OutLine.SrcDst.(linename)(1,2);

    nodenameSrc = ['N',num2str(nodeSrc)];
    nodenameDst = ['N',num2str(nodeDst)];

    Tcat_max_Src_aux = ...
        max(OutSect.(linename)(OutSect.(linename)(:,2)== ...
        min(OutSect.(linename)(:,2)),7));
    T_H120_Src = ...
        length(OutSect.(linename)(OutSect.(linename)(:,7) > 120 & ...
        OutSect.(linename)(:,2) == min(OutSect.(linename)(:,2)),1));

```

```
    Tcat_max_Dst_aux = ...
        max(OutSect.(linename)(OutSect.(linename)(:,2)== ...
        max(OutSect.(linename)(:,2)),7));
    T_H120_Dst = ...
        length(OutSect.(linename)(OutSect.(linename)(:,7) > 120 & ...
        OutSect.(linename)(:,2) == max(OutSect.(linename)(:,2)),1));

    Tcat_max.SrcDst.(nodenameSrc) = [ ...
        max(Tcat_max.SrcDst.(nodenameSrc)(1,1), ...
        Tcat_max_Src_aux(1,1)) ...
        max(Tcat_max.SrcDst.(nodenameSrc)(1,2), T_H120_Src(1,1))];
    Tcat_max.SrcDst.(nodenameDst) = [ ...
        max(Tcat_max.SrcDst.(nodenameDst)(1,1), ...
        Tcat_max_Dst_aux(1,1)) ...
        max(Tcat_max.SrcDst.(nodenameDst)(1,2), T_H120_Dst(1,1))];

end
```

A.8 Code implemented in the interface

```matlab

% Update the information box
app.info.Value = 'Selecting the case...';

% Open the foulder and select the file
[file, path] = uigetfile('*.db');
dbfile = fullfile(path,file);

% Update the information box
app.info.Value = 'Simulating...';

% Define the title for the thermal map
titlefile = strsplit(path,'\');
filename_title = titlefile(end-1);

if contains(filename_title,'_')==1
filename_title_aux = char(filename_title);
filename_title_aux = strsplit(filename_title_aux,'_');
filename_title_aux = strjoin(filename_title_aux,' ');
filename_title{1,1} = filename_title_aux;
end


tic
% Define the values of initial temperature
Ts0 = app.Tcat0.Value;        % Temperature of the conductor ...
    (initial value) in C
Tinf = app.Tair.Value;        % Temperature of the air in C

% Define the InitData for the catenary
matCat = app.Material.Value;          % material of the catenary
sectCat = app.Sectionmm2.Value;       % Section of the catenary in ...
    mm^2
```

```

% Define the weather conditions
Vw = app.speedWnd.Value;                % speed of the wind in m/s
He = app.seaLvl.Value;                  % sea level in m

% Solar heat gain
checkSHG = app.SolarHeatGainCheckBox.Value;  % solar heat gain on/off
n_month = app.MonthSim.Value;                % Month of the ...
    simulation
n_day = app.DaySim.Value;                    % Day of the simulation
Lat = app.Latitude.Value;                    % Latitude of the ...
    location in degrees
Lat = Lat*pi/180;                            % Latitude of the ...
    location in rad

% Length of the section
L_sect = app.LengthS.Value;                  % Length of the ...
    sections in km

% Select graphs depending on temperature range
T_H100 = app.TH100.Value;
T_90_100 = app.T90_100.Value;
T_80_90 = app.T80_90.Value;
T_60_80 = app.T60_80.Value;
T_40_60 = app.T40_60.Value;
T_20_40 = app.T20_40.Value;
T_20 = app.T20.Value;

[OutLine, OutSect, nlines, OutNode, nnodes, Tcat_max] = ...
    Thermal_study...
(Ts0,Tinf, dbfile, matCat, sectCat, He, Vw, checkSHG, n_month, ...
    n_day, Lat, L_sect);                 % Run the main file

% Update the information box
app.info.Value = 'Plotting the results...';

```

```
%% GRAPHS
% Init variables
nSrc = [];
nDst = [];
Xpos = [];
Ypos = [];
nNames = [];
nColor = [];
addedNodes100 = [];
addedNodes90_100 = [];
addedNodes80_90 = [];
addedNodes60_80 = [];
addedNodes40_60 = [];
addedNodes20_40 = [];
addedNodes20 = [];
nodeTmax = [];
nodeSize = [];
L_T_H100 = 0;

% Create the variables for the arrays
for n = 1:length(nlines)
linename = ['L', num2str(nlines(n))]; ...
                                            % Names of the lines
nSrc = [nSrc ; OutLine.SrcDst.(linename)(1,1)]; ...
                                  % Source node index
nDst = [nDst ; OutLine.SrcDst.(linename)(1,2)]; ...
                                  % Destination node index
nColor = [nColor; OutLine.Colors.(linename)]; ...
                                    % Color of the line
end

for n = 1:length(nnodes)
nodename = ['N', num2str(nnodes(n))]; ...
                                            % Names of the nodes in number
Xpos = [Xpos ; OutNode.Pos_XY.(nodename)(1,1)]; ...
                                  % X position of the nodes
```

```
Ypos = [Ypos ; OutNode.Pos_XY.(nodename)(1,2)]; ...
                                % Y position of the nodes
nNames = [nNames; OutNode.Name.(nodename)]; ...
                                    % Names of the nodes
nodeTmax = [nodeTmax; Tcat_max.SrcDst.(nodename)(:,1)...
 round(Tcat_max.SrcDst.(nodename)(:,2)/60,1)]; ...
                              % Maximum temperatures of the nodes in C
end

nodeSize = 4*ones(length(nnodes),1);

% Add new nodes
namesLimT = fieldnames(Tcat_max);
L_T_all = {};
% 1) T > 100 C
if all( structfun(@isempty, Tcat_max.Higher100) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.Higher100);
tf = cellfun(@(c) isempty(Tcat_max.Higher100.(c)), fn);
Tcat_max.Higher100 = rmfield(Tcat_max.Higher100, fn(tf));

L_T_H100 = fieldnames(Tcat_max.Higher100); ...
                                              % Lines with temperatures ...
    higher than 80C
ind_new_node = length(nnodes); ...
                                                        % Index of the ...
    new nodes



for n = 1:length(L_T_H100)

SrcNode = OutLine.SrcDst.(L_T_H100{n,1})(1,1); ...
                                  % Source node of the line
DstNode = OutLine.SrcDst.(L_T_H100{n,1})(1,2); ...
                                  % Destination node of the line
```

```
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                        % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                        % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                              % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                              % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

L_line_Real = OutLine.LengthReal.(L_T_H100{n,1}); ...
                            % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_H100{n,1}); ...
                          % Graphical length of the line
L_Real = Tcat_max.Higher100.(L_T_H100{n,1})(:,2); ...
                             % Distance between the source node and ...
    the point with temperature higher than 80C
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.Higher100.(L_T_H100{n,1})(:,3:4); ...
                      % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)(1,1)+...
(L_Graph.*cos(alpha));    % X position of the added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)(1,2)+...
(L_Graph.*sin(alpha));    % Y position of the added point
```

```

Xpos = [Xpos; X_added_point]; ...
                                        % Update the X ...
    coordenates
Ypos = [Ypos; Y_added_point]; ...
                                        % Updata the Y ...
    coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                    % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                     % Update the names of the ...
    nodes
addedNodes100 = [addedNodes100 ; ind_new_node]; ...
                           % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 5]; ...
                                        % Size of highlited ...
    nodes
nodeTmax = [nodeTmax; T_added_point(m,1) ...
    round(T_added_point(m,2)/60,1)];                    % ...
    Update the temperature of the added nodes
end

if T_H100 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_100 = ...
    OutSect.(L_T_H100{n,1})(OutSect.(L_T_H100{n,1})(:,1)==1,2); ...
          % Length of the Sections
GraphL_100 = [];

% Variable to save the lines that have been already plotted
L_T_all = L_T_H100;

for m = 1:length(SectL_100)
```

```
Temp_100 = OutSect.(L_T_H100{n,1})(OutSect.(L_T_H100{n,1})...
(:,2)==SectL_100(m),7);   % Select all the values of the temperatures
Temp_100 = Temp_100(round(length(Temp_100)/2):end,1); ...
                                   % Try to select only the ...
    temperatures in permanent regimen
GraphL_100 = [GraphL_100; SectL_100(m), mean(Temp_100)]; ...
                                 % Average temperatures
end
Sect_Tmax_100 = Tcat_max.Higher100.(L_T_H100{n,1})...
(Tcat_max.Higher100.(L_T_H100{n,1})...
(:,3)==max(Tcat_max.Higher100.(L_T_H100{n,1})(:,3)),2);
Sect_Tmax_100 = Sect_Tmax_100(1,1);           % Sect with the ...
    highest temperature
Pk_Tmax_100 = Tcat_max.Higher100.(L_T_H100{n,1})...
(Tcat_max.Higher100.(L_T_H100{n,1})...
(:,3)==max(Tcat_max.Higher100.(L_T_H100{n,1})(:,3)),5);
Pk_Tmax_100 = Pk_Tmax_100(1,1);
Temp_100 = OutSect.(L_T_H100{n,1})...
(OutSect.(L_T_H100{n,1})(:,2)==Sect_Tmax_100,7);
time_100 = OutSect.(L_T_H100{n,1})...
(OutSect.(L_T_H100{n,1})(:,2)==Sect_Tmax_100,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_100(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_H100{n,1}),GraphL_100(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_H100{n,1} '), ...
from ' nodenameSrc_text ' to ' nodenameDst_text]);
```

```
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_H100{n,1})(1,1) ...
OutLine.Real_PK.(L_T_H100{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_100,Temp_100)
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_H100{n,1} '), at ' ...
    num2str(Pk_Tmax_100) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_100(1,1) time_100(end,1)]);
grid on;
end
end
end
end
% 2) 100 > T > 90
if all( structfun(@isempty, Tcat_max.fr90to100) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.fr90to100);
tf = cellfun(@(c) isempty(Tcat_max.fr90to100.(c)), fn);
Tcat_max.fr90to100 = rmfield(Tcat_max.fr90to100, fn(tf));

L_T_90_100 = fieldnames(Tcat_max.fr90to100); ...
                                        % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100); ...
                                % Index of the new nodes
for n = 1:length(L_T_90_100)

```

```
SrcNode = OutLine.SrcDst.(L_T_90_100{n,1})(1,1); ...
                              % Source node of the line
DstNode = OutLine.SrcDst.(L_T_90_100{n,1})(1,2); ...
                              % Destination node of the line
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                       % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                       % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                               % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                               % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

L_line_Real = OutLine.LengthReal.(L_T_90_100{n,1}); ...
                           % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_90_100{n,1}); ...
                         % Graphical length of the line
L_Real = Tcat_max.fr90to100.(L_T_90_100{n,1})(:,2); ...
                            % Distance between the source node and ...
    added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                 % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.fr90to100.(L_T_90_100{n,1})(:,3:4); ...
                      % Maximum temperatures of the added nodes

```

```
X_added_point = ...
    OutNode.Pos_XY.(nodenameSrc)(1,1)+(L_Graph.*cos(alpha));   % X ...
    position of added point
Y_added_point = ...
    OutNode.Pos_XY.(nodenameSrc)(1,2)+(L_Graph.*sin(alpha));   % Y ...
    position of added point

Xpos = [Xpos; X_added_point]; ...
                                                     % Update the X ...
    coordenates
Ypos = [Ypos; Y_added_point]; ...
                                                     % Updata the Y ...
    coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                              % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                                % Update the names of the ...
    nodes
addedNodes90_100 = [addedNodes90_100 ; ind_new_node]; ...
                  % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                                     % Size of highlited ...
    nodes
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                             % Update the temperature of the ...
    added nodes
end
if T_90_100 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_90_100 = ...
    OutSect.(L_T_90_100{n,1})(OutSect.(L_T_90_100{n,1})(:,1)==1,2); ...
          % Length of the Sections
GraphL_90_100 = [];
```

```

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_90_100{n,1},L_T_all); ...
                              % line that must be removed

% Condition to not plot again the same line that already appear in
% other range of temperatures
if all(ind_line_rmvd==0)
L_T_all = [L_T_all;L_T_90_100{n,1}];
% Update the values of the lines for average
%temperature and evolution of temperuta with time
for m = 1:length(SectL_90_100)
Temp_90_100 = OutSect.(L_T_90_100{n,1})(OutSect.(L_T_90_100{n,1})...
(:,2)==SectL_90_100(m),7);     % Select all the values of the ...
    temperatures
% Try to select only the temperatures in permanent regimen
Temp_90_100 = Temp_90_100(round(length(Temp_90_100)/2):end,1);
% Average temperatures

GraphL_90_100 = [GraphL_90_100; SectL_90_100(m), mean(Temp_90_100)];
end
Sect_Tmax_90_100 = Tcat_max.fr90to100.(L_T_90_100{n,1})...
(Tcat_max.fr90to100.(L_T_90_100{n,1})...
(:,3)==max(Tcat_max.fr90to100.(L_T_90_100{n,1})(:,3)),2);
Sect_Tmax_90_100 = Sect_Tmax_90_100(1,1);          % Sect with ...
    the highest temperature
Pk_Tmax_90_100 = Tcat_max.fr90to100.(L_T_90_100{n,1})...
(Tcat_max.fr90to100.(L_T_90_100{n,1})...
(:,3)==max(Tcat_max.fr90to100.(L_T_90_100{n,1})(:,3)),5);
Pk_Tmax_90_100 = Pk_Tmax_90_100(1,1);
Temp_90_100 = OutSect.(L_T_90_100{n,1})(OutSect.(L_T_90_100{n,1})...
(:,2)==Sect_Tmax_90_100,7);
time_90_100 = OutSect.(L_T_90_100{n,1})(OutSect.(L_T_90_100{n,1})...
(:,2)==Sect_Tmax_90_100,1);

% Graphs of the evolution of the temperature with time and the
```

```
% average temperature for the simaltion
if length(GraphL_90_100(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_90_100{n,1}),GraphL_90_100(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_90_100{n,1} '),...
 from ' nodenameSrc_text ' to ' nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_90_100{n,1})(1,1)...
 OutLine.Real_PK.(L_T_90_100{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_90_100,Temp_90_100)
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_90_100{n,1} '),...
 at ' num2str(Pk_Tmax_90_100) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_90_100(1,1) time_90_100(end,1)]);
grid on;
end
end
end
end
end
% 3) 80 > T > 90
if all( structfun(@isempty, Tcat_max.fr80to90) )==0
```

```
% Remove the fields that are empty
fn = fieldnames(Tcat_max.fr80to90);
tf = cellfun(@(c) isempty(Tcat_max.fr80to90.(c)), fn);
Tcat_max.fr80to90 = rmfield(Tcat_max.fr80to90, fn(tf));

L_T_80_90 = fieldnames(Tcat_max.fr80to90); ...
                                        % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100)...
+length(addedNodes90_100);                          % Index of ...
    the new nodes
for n = 1:length(L_T_80_90)

SrcNode = OutLine.SrcDst.(L_T_80_90{n,1})(1,1); ...
                              % Source node of the line
DstNode = OutLine.SrcDst.(L_T_80_90{n,1})(1,2); ...
                              % Destination node of the line
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                        % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                        % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                               % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                               % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);
```

```
L_line_Real = OutLine.LengthReal.(L_T_80_90{n,1}); ...
                           % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_80_90{n,1}); ...
                         % Graphical length of the line
L_Real = Tcat_max.fr80to90.(L_T_80_90{n,1})(:,2); ...
                            % Distance between the source node and ...
    added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                 % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.fr80to90.(L_T_80_90{n,1})(:,3:4); ...
                     % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)(1,1)+...
(L_Graph.*cos(alpha));    % X position of added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)(1,2)+...
(L_Graph.*sin(alpha));    % Y position of added point

Xpos = [Xpos; X_added_point]; ...
                                                % Update the X ...
    coordenates
Ypos = [Ypos; Y_added_point]; ...
                                                % Updata the Y ...
    coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                         % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                           % Update the names of the ...
    nodes
addedNodes80_90 = [addedNodes80_90 ; ind_new_node]; ...
                      % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                                % Size of highlited ...
```

```
    nodes
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                        % Update the temperature of the ...
    added nodes
end

if T_80_90 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_80_90 = OutSect.(L_T_80_90{n,1})...
(OutSect.(L_T_80_90{n,1})(:,1)==1,2);        % Length of the Sections
GraphL_80_90 = [];

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_80_90{n,1},L_T_all); ...
                                   % line that must be removed

% Condition to not plot again the same line that already appear in
% other range of temperatures
if all(ind_line_rmvd==0)
% Update the values of the lines for average
%temperature and evolution of temperuta with time
L_T_all = [L_T_all;L_T_80_90{n,1}];

for m = 1:length(SectL_80_90)
Temp_80_90 = OutSect.(L_T_80_90{n,1})...
(OutSect.(L_T_80_90{n,1})(:,2)==SectL_80_90(m),7);      % Select ...
    all the values of the temperatures
Temp_80_90 = Temp_80_90(round(length(Temp_80_90)/2):end,1); ...
                                          % Try to select only the ...
    temperatures in permanent regimen
GraphL_80_90 = [GraphL_80_90; SectL_80_90(m), mean(Temp_80_90)]; ...
                                     % Average temperatures
end

```

```
Sect_Tmax_80_90 = ...
    Tcat_max.fr80to90.(L_T_80_90{n,1})(Tcat_max.fr80to90.(L_T_80_90{n,1})...
(:,3)==max(Tcat_max.fr80to90.(L_T_80_90{n,1})(:,3)),2);
Sect_Tmax_80_90 = Sect_Tmax_80_90(1,1);          % Sect with the ...
    highest temperature
Pk_Tmax_80_90 = Tcat_max.fr80to90.(L_T_80_90{n,1})...
(Tcat_max.fr80to90.(L_T_80_90{n,1})...
(:,3)==max(Tcat_max.fr80to90.(L_T_80_90{n,1})(:,3)),5);
Pk_Tmax_80_90 = Pk_Tmax_80_90(1,1);
Temp_80_90 = OutSect.(L_T_80_90{n,1})(OutSect.(L_T_80_90{n,1})...
(:,2)==Sect_Tmax_80_90,7);
time_80_90 = OutSect.(L_T_80_90{n,1})(OutSect.(L_T_80_90{n,1})...
(:,2)==Sect_Tmax_80_90,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_80_90(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_80_90{n,1}),GraphL_80_90(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_80_90{n,1} '),...
 from ' nodenameSrc_text ' to ' nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_80_90{n,1})(1,1) ...
OutLine.Real_PK.(L_T_80_90{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_80_90,Temp_80_90)
set(gca, 'FontName', 'Times New Roman');
```

```
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_80_90{n,1} '),...
 at ' num2str(Pk_Tmax_80_90) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_80_90(1,1) time_80_90(end,1)]);
grid on;
end
end
end
end
end

% 4) 60 > T > 80
if all( structfun(@isempty, Tcat_max.fr60to80) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.fr60to80);
tf = cellfun(@(c) isempty(Tcat_max.fr60to80.(c)), fn);
Tcat_max.fr60to80 = rmfield(Tcat_max.fr60to80, fn(tf));

L_T_60_80 = fieldnames(Tcat_max.fr60to80); ...
                                        % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100)+....
length(addedNodes90_100)+length(addedNodes80_90); ...
                                 % Index of the new nodes
for n = 1:length(L_T_60_80)

SrcNode = OutLine.SrcDst.(L_T_60_80{n,1})(1,1); ...
                               % Source node of the line
DstNode = OutLine.SrcDst.(L_T_60_80{n,1})(1,2); ...
                               % Destination node of the line
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                        % Name of the source node
```

```
nodenameDst = ['N', num2str(DstNode)]; ...
                                        % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                                % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                                % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1),...
 OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

L_line_Real = OutLine.LengthReal.(L_T_60_80{n,1}); ...
                            % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_60_80{n,1}); ...
                          % Graphical length of the line
L_Real = Tcat_max.fr60to80.(L_T_60_80{n,1})(:,2); ...
                             % Distance between the source node and ...
    added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                  % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.fr60to80.(L_T_60_80{n,1})(:,3:4); ...
                      % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)(1,1)+...
(L_Graph.*cos(alpha));    % X position of added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)(1,2)+...
(L_Graph.*sin(alpha));    % Y position of added point

```

```
Xpos = [Xpos; X_added_point]; ...
                                                    % Update the X ...
    coordenates
Ypos = [Ypos; Y_added_point]; ...
                                                    % Updata the Y ...
    coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                             % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                               % Update the names of the ...
    nodes
addedNodes60_80 = [addedNodes60_80 ; ind_new_node]; ...
                       % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                                    % Size of highlited ...
    nodes
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                          % Update the temperature of the ...
    added nodes
end

if T_60_80 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_60_80 = OutSect.(L_T_60_80{n,1})...
(OutSect.(L_T_60_80{n,1})(:,1)==1,2);       % Length of the Sections
GraphL_60_80 = [];

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_60_80{n,1},L_T_all); ...
                                   % line that must be removed

% Condition to not plot again the same line that already appear in
% other range of temperatures
```

```
if all(ind_line_rmvd==0)
L_T_all = [L_T_all;L_T_60_80{n,1}];                    % Update the ...
    values of the lines for average temperature and evolution of ...
    temperuta with time
for m = 1:length(SectL_60_80)
Temp_60_80 = OutSect.(L_T_60_80{n,1})...
(OutSect.(L_T_60_80{n,1})(:,2)==SectL_60_80(m),7);     % Select ...
    all the values of the temperatures
Temp_60_80 = Temp_60_80(round(length(Temp_60_80)/2):end,1); ...
                                    % Try to select only the ...
    temperatures in permanent regimen
GraphL_60_80 = [GraphL_60_80; SectL_60_80(m), mean(Temp_60_80)]; ...
                              % Average temperatures
end

Sect_Tmax_60_80 = Tcat_max.fr60to80.(L_T_60_80{n,1})...
(Tcat_max.fr60to80.(L_T_60_80{n,1})...
(:,3)==max(Tcat_max.fr60to80.(L_T_60_80{n,1})(:,3)),2);
Sect_Tmax_60_80 = Sect_Tmax_60_80(1,1);          % Sect with the ...
    highest temperature
Pk_Tmax_60_80 = Tcat_max.fr60to80.(L_T_60_80{n,1})...
(Tcat_max.fr60to80.(L_T_60_80{n,1})...
(:,3)==max(Tcat_max.fr60to80.(L_T_60_80{n,1})(:,3)),5);
Pk_Tmax_60_80 = Pk_Tmax_60_80(1,1);
Temp_60_80 = OutSect.(L_T_60_80{n,1})...
(OutSect.(L_T_60_80{n,1})(:,2)==Sect_Tmax_60_80,7);
time_60_80 = OutSect.(L_T_60_80{n,1})...
(OutSect.(L_T_60_80{n,1})(:,2)==Sect_Tmax_60_80,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_60_80(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_60_80{n,1}),GraphL_60_80(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
```

```
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary ...
(' L_T_60_80{n,1} '), from ' nodenameSrc_text ' to ' ...
    nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_60_80{n,1})...
(1,1) OutLine.Real_PK.(L_T_60_80{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_60_80,Temp_60_80)
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',...
[x0,y0,width,height])
title(['Temperature of the catenary ...
(' L_T_60_80{n,1} '), at ' num2str(Pk_Tmax_60_80) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_60_80(1,1) time_60_80(end,1)]);
grid on;
end
end
end
end
end

% 5) 40 > T > 60
if all( structfun(@isempty, Tcat_max.fr40to60) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.fr40to60);
tf = cellfun(@(c) isempty(Tcat_max.fr40to60.(c)), fn);
```

```
Tcat_max.fr40to60 = rmfield(Tcat_max.fr40to60, fn(tf));

L_T_40_60 = fieldnames(Tcat_max.fr40to60); ...
                                          % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100)+...
length(addedNodes90_100)...
+length(addedNodes80_90)+length(addedNodes60_80); ...
                                   % Index of the new nodes
for n = 1:length(L_T_40_60)

SrcNode = OutLine.SrcDst.(L_T_40_60{n,1})(1,1); ...
                                 % Source node of the line
DstNode = OutLine.SrcDst.(L_T_40_60{n,1})(1,2); ...
                                 % Destination node of the line
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                          % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                          % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                                  % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                                  % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

```

```
L_line_Real = OutLine.LengthReal.(L_T_40_60{n,1}); ...
                        % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_40_60{n,1}); ...
                      % Graphical length of the line
L_Real = Tcat_max.fr40to60.(L_T_40_60{n,1})(:,2); ...
                         % Distance between the source node and ...
   added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                              % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.fr40to60.(L_T_40_60{n,1})(:,3:4); ...
                  % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)(1,1)...
+(L_Graph.*cos(alpha));   % X position of added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)(1,2)+...
(L_Graph.*sin(alpha));   % Y position of added point

Xpos = [Xpos; X_added_point]; ...
                                            % Update the X ...
   coordenates
Ypos = [Ypos; Y_added_point]; ...
                                            % Updata the Y ...
   coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                       % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                         % Update the names of the ...
   nodes
addedNodes40_60 = [addedNodes40_60 ; ind_new_node]; ...
                    % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                            % Size of highlited ...
   nodes
```

```
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                        % Update the temperature of the ...
    added nodes
end

if T_40_60 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_40_60 = OutSect.(L_T_40_60{n,1})(OutSect.(L_T_40_60...
{n,1})(:,1)==1,2);        % Length of the Sections
GraphL_40_60 = [];

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_40_60{n,1},L_T_all); ...
                                  % line that must be removed

% Condition to not plot again the same line that already appear in
% other range of temperatures
if all(ind_line_rmvd==0)
L_T_all = [L_T_all;L_T_40_60{n,1}];                  % Update the ...
    values of the lines for average temperature and evolution of ...
    temperuta with time
for m = 1:length(SectL_40_60)
Temp_40_60 = OutSect.(L_T_40_60{n,1})(OutSect.(L_T_40_60{n,1})...
(:,2)==SectL_40_60(m),7);      % Select all the values of the ...
    temperatures
Temp_40_60 = Temp_40_60(round(length(Temp_40_60)/2):end,1); ...
                                           % Try to select only the ...
    temperatures in permanent regimen
GraphL_40_60 = [GraphL_40_60; SectL_40_60(m), mean(Temp_40_60)]; ...
                                      % Average temperatures
end

Sect_Tmax_40_60 = Tcat_max.fr40to60.(L_T_40_60{n,1})...
(Tcat_max.fr40to60.(L_T_40_60{n,1})...
(:,3)==max(Tcat_max.fr40to60.(L_T_40_60{n,1})(:,3)),2);
```

```
Sect_Tmax_40_60 = Sect_Tmax_40_60(1,1);          % Sect with the ... highest temperature
Pk_Tmax_40_60 = Tcat_max.fr40to60.(L_T_40_60{n,1})...
(Tcat_max.fr40to60.(L_T_40_60{n,1})(:,3)...
==max(Tcat_max.fr40to60.(L_T_40_60{n,1})(:,3)),5);
Pk_Tmax_40_60 = Pk_Tmax_40_60(1,1);
Temp_40_60 = OutSect.(L_T_40_60{n,1})...
(OutSect.(L_T_40_60{n,1})...
(:,2)==Sect_Tmax_40_60,7);
time_40_60 = OutSect.(L_T_40_60{n,1})...
(OutSect.(L_T_40_60{n,1})...
(:,2)==Sect_Tmax_40_60,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_40_60(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_40_60{n,1}),GraphL_40_60(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',...
[x0,y0,width,height])
title(['Temperature of the catenary ...
(' L_T_40_60{n,1} '), from ' nodenameSrc_text ' to ' ... nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_40_60{n,1})...
(1,1) OutLine.Real_PK.(L_T_40_60{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_40_60,Temp_40_60)
```

```
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_40_60{n,1} '), at ' ...
    num2str(Pk_Tmax_40_60) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_40_60(1,1) time_40_60(end,1)]);
grid on;
end
end
end
end
end

% 6) 20 > T > 40
if all( structfun(@isempty, Tcat_max.fr20to40) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.fr20to40);
tf = cellfun(@(c) isempty(Tcat_max.fr20to40.(c)), fn);
Tcat_max.fr20to40 = rmfield(Tcat_max.fr20to40, fn(tf));

L_T_20_40 = fieldnames(Tcat_max.fr20to40); ...
                                        % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100)+length...
(addedNodes90_100)+length(addedNodes80_90)+length...
(addedNodes60_80)+length(addedNodes40_60); ...
                                   % Index of the new nodes
for n = 1:length(L_T_20_40)

SrcNode = OutLine.SrcDst.(L_T_20_40{n,1})(1,1); ...
                                 % Source node of the line
DstNode = OutLine.SrcDst.(L_T_20_40{n,1})(1,2); ...
                                 % Destination node of the line
```

```
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                        % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                        % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                                % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                                % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1), ...
OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

L_line_Real = OutLine.LengthReal.(L_T_20_40{n,1}); ...
                            % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_20_40{n,1}); ...
                          % Graphical length of the line
L_Real = Tcat_max.fr20to40.(L_T_20_40{n,1})(:,2); ...
                             % Distance between the source node and ...
    added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                  % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.fr20to40.(L_T_20_40{n,1})(:,3:4); ...
                      % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)...
(1,1)+(L_Graph.*cos(alpha));   % X position of added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)...
(1,2)+(L_Graph.*sin(alpha));   % Y position of added point
```

```
Xpos = [Xpos; X_added_point]; ...
                                                  % Update the X ...
    coordenates
Ypos = [Ypos; Y_added_point]; ...
                                                  % Updata the Y ...
    coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                           % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                             % Update the names of the ...
    nodes
addedNodes20_40 = [addedNodes20_40 ; ind_new_node]; ...
                        % Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                                  % Size of highlited ...
    nodes
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                        % Update the temperature of the ...
    added nodes
end

if T_20_40 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_20_40 = OutSect.(L_T_20_40{n,1})...
(OutSect.(L_T_20_40{n,1})(:,1)==1,2);       % Length of the Sections
GraphL_20_40 = [];

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_20_40{n,1},L_T_all); ...
                                  % line that must be removed

% Condition to not plot again the same line that already appear in
```

```
% other range of temperatures
if all(ind_line_rmvd==0)
L_T_all = [L_T_all;L_T_20_40{n,1}];                   % Update the ...
    values of the lines for average temperature and evolution of ...
    temperuta with time
for m = 1:length(SectL_20_40)
Temp_20_40 = OutSect.(L_T_20_40{n,1})...
(OutSect.(L_T_20_40{n,1})(:,2)==SectL_20_40(m),7);      % Select ...
    all the values of the temperatures
Temp_20_40 = Temp_20_40(round(length(Temp_20_40)/2):end,1); ...
                                         % Try to select only the ...
    temperatures in permanent regimen
GraphL_20_40 = [GraphL_20_40; SectL_20_40(m), mean(Temp_20_40)]; ...
                                    % Average temperatures
end

Sect_Tmax_20_40 = ...
    Tcat_max.fr20to40.(L_T_20_40{n,1})(Tcat_max.fr20to40.(L_T_20_40{n,1})...
(:,3)==max(Tcat_max.fr20to40.(L_T_20_40{n,1})(:,3)),2);
Sect_Tmax_20_40 = Sect_Tmax_20_40(1,1);           % Sect with the ...
    highest temperature
Pk_Tmax_20_40 = Tcat_max.fr20to40.(L_T_20_40{n,1})...
(Tcat_max.fr20to40.(L_T_20_40{n,1})(:,3)...
==max(Tcat_max.fr20to40.(L_T_20_40{n,1})(:,3)),5);
Pk_Tmax_20_40 = Pk_Tmax_20_40(1,1);
Temp_20_40 = OutSect.(L_T_20_40{n,1})(OutSect.(L_T_20_40{n,1})...
(:,2)==Sect_Tmax_20_40,7);
time_20_40 = OutSect.(L_T_20_40{n,1})(OutSect.(L_T_20_40{n,1})...
(:,2)==Sect_Tmax_20_40,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_20_40(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_20_40{n,1}),GraphL_20_40(:,2));
set(gca, 'FontName', 'Times New Roman');
```

```
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_20_40{n,1} '),...
 from ' nodenameSrc_text ' to ' nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_20_40{n,1})(1,1) ...
OutLine.Real_PK.(L_T_20_40{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_20_40,Temp_20_40)
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_20_40{n,1} '), at ' ...
    num2str(Pk_Tmax_20_40) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_20_40(1,1) time_20_40(end,1)]);
grid on;
end
end
end
end
end

% 7) T < 20
if all( structfun(@isempty, Tcat_max.Lower20) )==0
% Remove the fields that are empty
fn = fieldnames(Tcat_max.Lower20);
tf = cellfun(@(c) isempty(Tcat_max.Lower20.(c)), fn);
Tcat_max.Lower20 = rmfield(Tcat_max.Lower20, fn(tf));
```

```
L_T_20 = fieldnames(Tcat_max.Lower20); ...
                                        % Lines with temperatures ...
    higher than 80C

ind_new_node = length(nnodes)+length(addedNodes100)+...
length(addedNodes90_100)...
+length(addedNodes80_90)+length(addedNodes60_80)+...
length(addedNodes40_60)+length(addedNodes20_40); ...
                              % Index of the new nodes
for n = 1:length(L_T_20)

SrcNode = OutLine.SrcDst.(L_T_20{n,1})(1,1); ...
                            % Source node of the line
DstNode = OutLine.SrcDst.(L_T_20{n,1})(1,2); ...
                            % Destination node of the line
nodenameSrc = ['N', num2str(SrcNode)]; ...
                                        % Name of the source node
nodenameDst = ['N', num2str(DstNode)]; ...
                                        % Name of the destination node

nodenameSrc_text = OutNode.Name.(nodenameSrc){1,1}; ...
                             % Name of the Source node (used in ...
    T_avg Vs PK)
nodenameDst_text = OutNode.Name.(nodenameDst){1,1}; ...
                             % Name of the Destination node (used ...
    in T_avg Vs PK)

SrcPoint = [OutNode.Pos_XY.(nodenameSrc)(1,1),...
 OutNode.Pos_XY.(nodenameSrc)(1,2)];
DstPoint = [OutNode.Pos_XY.(nodenameDst)(1,1), ...
OutNode.Pos_XY.(nodenameDst)(1,2)];

alpha = getAngle(SrcPoint,DstPoint);

```

```
L_line_Real = OutLine.LengthReal.(L_T_20{n,1}); ...
                           % Real length of the line
L_line_Graph = OutLine.LengthGraph.(L_T_20{n,1}); ...
                         % Graphical length of the line
L_Real = Tcat_max.Lower20.(L_T_20{n,1})(:,2); ...
                            % Distance between the source node and ...
   added point
L_Graph = L_line_Graph.*L_Real./L_line_Real; ...
                                  % L_Tmax_Real scaled to the graph

T_added_point = Tcat_max.Lower20.(L_T_20{n,1})(:,3:4); ...
                      % Maximum temperatures of the added nodes

X_added_point = OutNode.Pos_XY.(nodenameSrc)...
(1,1)+(L_Graph.*cos(alpha));   % X position of added point
Y_added_point = OutNode.Pos_XY.(nodenameSrc)...
(1,2)+(L_Graph.*sin(alpha));   % Y position of added point

Xpos = [Xpos; X_added_point]; ...
                                                 % Update the X ...
   coordenates
Ypos = [Ypos; Y_added_point]; ...
                                                 % Updata the Y ...
   coordenates

for m = 1:length(L_Real)
ind_new_node = ind_new_node + 1; ...
                                          % Update the index of the node
nNames = vertcat(nNames, ' '); ...
                                            % Update the names of the ...
   nodes
addedNodes20 = [addedNodes20 ; ind_new_node];                % ...
   Add the new nodes to the array addedNodes
nodeSize = [nodeSize; 3]; ...
                                                 % Size of highlited ...
   nodes
```

```
nodeTmax = [nodeTmax; T_added_point(m,:)]; ...
                                        % Update the temperature of the ...
    added nodes
end

if T_20 == 1
% Graphs averaged temperature against length & evolution of the
% temperature with time for the section with highest Temp
SectL_20 = ...
    OutSect.(L_T_20{n,1})(OutSect.(L_T_20{n,1})(:,1)==1,2);         ...
    % Length of the Sections
GraphL_20 = [];

% Variable to save the lines that have been already plotted
ind_line_rmvd = strcmp(L_T_20{n,1},L_T_all); ...
                                 % line that must be removed

% Condition to not plot again the same line that already appear in
% other range of temperatures
if all(ind_line_rmvd==0)
L_T_all = [L_T_all;L_T_20{n,1}];
% Update the values of the lines for average ...
temperature and evolution of temperuta with time
for m = 1:length(SectL_20)
Temp_20 = OutSect.(L_T_20{n,1})(OutSect.(L_T_20{n,1})...
(:,2)==SectL_20(m),7);      % Select all the values of the ...
    temperatures
Temp_20 = Temp_20(round(length(Temp_20)/2):end,1); ...
                                           % Try to select only the ...
    temperatures in permanent regimen
GraphL_20 = [GraphL_20; SectL_20(m), mean(Temp_20)]; ...
                                  % Average temperatures
end

Sect_Tmax_20 = ...
    Tcat_max.Lower20.(L_T_20{n,1})(Tcat_max.Lower20.(L_T_20{n,1})...
```

```
(:,3)==max(Tcat_max.Lower20.(L_T_20{n,1})(:,3)),2);
Sect_Tmax_20 = Sect_Tmax_20(1,1);          % Sect with the ...
    highest temperature
Pk_Tmax_20 = ...
    Tcat_max.Lower20.(L_T_20{n,1})(Tcat_max.Lower20.(L_T_20{n,1})(:,3)...
==max(Tcat_max.Lower20.(L_T_20{n,1})(:,3)),5);
Pk_Tmax_20 = Pk_Tmax_20(1,1);
Temp_20 = ...
    OutSect.(L_T_20{n,1})(OutSect.(L_T_20{n,1})(:,2)==Sect_Tmax_20,7);
time_20 = ...
    OutSect.(L_T_20{n,1})(OutSect.(L_T_20{n,1})(:,2)==Sect_Tmax_20,1);

% Graphs of the evolution of the temperature with time and the
% average temperature for the simaltion
if length(GraphL_20(:,1))≠1
figure('Name','Average temperature along the line');
plot(OutLine.Real_PK.(L_T_20{n,1}),GraphL_20(:,2));
set(gca, 'FontName', 'Times New Roman');
x0=340;
y0=200;
width=400;
height=200;
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
title(['Temperature of the catenary (' L_T_20{n,1} '),...
 from ' nodenameSrc_text ' to ' nodenameDst_text]);
xlabel('Distance (km)');
ylabel('Temperature (C)');
xlim([OutLine.Real_PK.(L_T_20{n,1})(1,1)...
 OutLine.Real_PK.(L_T_20{n,1})(end,1)]);
grid on;
figure('Name','Evolution of the temperature');
plot(time_20,Temp_20)
set(gca, 'FontName', 'Times New Roman');
set(gcf,'position',[x0,y0,width,height]);
set(gcf,'units','points','position',[x0,y0,width,height])
```

```
title(['Temperature of the catenary (' L_T_20{n,1} '), at...
 ' num2str(Pk_Tmax_20) ' km ']);
xlabel('Time (s)');
ylabel('Temperature (C)');
xlim([time_20(1,1) time_20(end,1)]);
grid on;
end
end
end
end
end

% Reorganize the number of the nodes in order to have a ...
    sequential numbering
while nnodes(1,1) ≠ 1
nnodes = nnodes - 1;
nSrc = nSrc - 1;
nDst = nDst - 1;
end

while  any(diff(nnodes) ≠ 1)

node_not_seg = nnodes((diff(nnodes)≠1));
nnodes_to_change = nnodes(nnodes(:,1)>node_not_seg(1,1),1);

for n = 1:length(nnodes_to_change)
nSrc(nSrc(:,1) == nnodes_to_change(n,1)) = nSrc(nSrc(:,1)...
 == nnodes_to_change(n,1),1)-1;
nDst(nDst(:,1) == nnodes_to_change(n,1)) = nDst(nDst(:,1) ...
== nnodes_to_change(n,1),1)-1;

end
nnodes = [nnodes(1:node_not_seg(1,1),1); nnodes...
(node_not_seg(1,1)+1:end,1)-1];
end

```

```

%% Create the grid graph
gridGraph = graph(nSrc(:,1), nDst(:,1));
num_added_nodes = ind_new_node - length(nnodes);
gridGraph = addnode(gridGraph, num_added_nodes);

% Create the figure
figure('Name','Thermal Map');
H = plot(gridGraph, 'XData', Xpos, 'YData', Ypos, 'NodeLabel', ...
nNames, 'EdgeColor', nColor, 'LineWidth', 2, 'MarkerSize', nodeSize);
set(get(get(H(1),'Annotation'),'LegendInformation')...
,'IconDisplayStyle','off');
title(['Thermal Map of ',filename_title{1,1}], 'FontSize', 14);

hold on
scatter([],[], 5, 'r','filled');
scatter([],[], 3, [1 0.5 0],'filled');
scatter([],[], 3, [1 1 0],'filled');
scatter([],[], 3, [0.5 1 0],'filled');
scatter([],[], 3, [0 1 0.5],'filled');
scatter([],[], 3, [0 1 1],'filled');
scatter([],[], 3, [0 0.5 1],'filled');


legend('T_{max} \geq 100 C', '90 C \leq T_{max} < 100 C','80 C ...
\leq T_{max} < 90 C','60 C \leq T_{max} < 80 C','40 C \leq T_{max}...
 < 60 C', '20 C \leq T_{max} < 40 C','T_{max} < 20 C');

highlight(H,addedNodes20,'NodeColor',[0 0.5 1])
highlight(H,addedNodes20_40,'NodeColor',[0 1 1])
highlight(H,addedNodes40_60,'NodeColor',[0 1 0.5])
highlight(H,addedNodes60_80,'NodeColor',[0.5 1 0])
highlight(H,addedNodes80_90,'NodeColor',[1 1 0])
highlight(H,addedNodes90_100,'NodeColor', [1 0.5 0])
highlight(H,addedNodes100,'NodeColor','r')
set(gca, 'XTick', [], 'YTick', [] );
```

```
set(gca, 'FontName', 'Times New Roman');
hold off

% Update the information box
app.info.Value = 'Simulation has finished succesfully';

% Data Cursor Callback
hdt = datacursormode;
hdt.UpdateFcn = @(obj,event_obj) ...
    GraphCursorCallback(obj,event_obj,nodeTmax);
```

Bibliography

[1] Sight reduction tables for air navigators. ho pub. no. 249, vols. ii and iii,. US Navy Hydrographic Office.

[2] Transmission conductors thermal ratings paper 68-tap-28. Report by Transmission Advisory Panel, East Central Area Reliability Coordination Agreement.

[3] Determination of bare overhead conductor ratings. Conductor Rating Task Force, PA, NJ, and MD Interconnection, May 1973.

[4] Ieee draft standard for calculating the current-temperature relationship of bare overhead conductors. *IEEE Std 738-2012 Draft 10 (Revision of IEEE Std 738-2006)*, pages 1–67, Sep. 2012.

[5] Gonzalo Abad. *Power Electronics and Electric Drives for Traction Applications*. Mondragon University, Spain, 2017.

[6] CIGRE Working Group B2.12. Thermal behaviour of overhead conductors. Technical Brochure 207, August 2002.

[7] Bush R. A. Black, W. Z. Conductor temperature research. EPRI Report EL 5707, May 1988.

[8] R. L. Black, W. Z. Rehberg. Simplified model for steady state and real-time ampacity of overhead conductors. IEEE Transactions on Power Apparatus and Systems, vol. 104, pp. 29–42, October 1985.

[9] W. R. Black, W. Z. Byrd. Real-time ampacity model for overhead lines. IEEE Transactions on Power Apparatus and Systems, vol. PAS-102, No. 7, pp. 2289-2293, July 1983.

[10] Energy department CAF TE. Traction systems in europe. 2016.

[11] Yunus A Çengel and Michael A Boles. *Thermodynamics: An Engineering Approach,-PDF*. McGraw-Hill, 2008.

[12] House H. E. Tuttle P. D. Current carrying capacity of acsr. IEEE Transactions on Power Apparatus and Systems,pp. 1169-1178, February 1958.

[13] Donoho T. E. Landrieu P. R. H. Mcelhaney R. T. Saeger J. H.I Davidson, G. A. Short-time thermal ratings for bare overhead conductors. EEE Transactions on Power Apparatus and Systems, vol. PAS-88, No. 3, March 1969.

[14] M. W Davis. A new thermal rating approach: The real time thermal rating system for strategic overhead conductor transmission lines, part ii. IEEE Transactions on Power Apparatus and Systems, vol. PAS-97, pp. 810–825,, April 1978.

[15] Lin S. H. Fernandez R. A. Foss, S. D. Dynamic thermal line ratings,part 1,dynamic ampacity rating algorithm. IEEE Transactions on Power Apparatus and Systems, vol. PAS-102, No. 6, pp 1858-1864, June 1983.

[16] Sheilah Frey. *Railway electrification systems & engineering.* Delhi: White Word Publications., 2012.

[17] J Guo, AW Glisson, and D Kajfez. Skin-effect resistance of conductors with a trapezoidal cross section. *Microwave and Optical Technology Letters*, 18(6):387–389, 1998.

[18] Rigdon W. S. Grosh R. J. Cottingham W. B. House, H. E. Emissivity of weathered conductors after service in rural and industrial environments. AIEE Transactions, pp. 891–896, February 1963.

[19] Jorge LLaviana Juan. *Diseño de las subestaciones eléctricas de tracción y centros de autotransformación asociados de una línea ferroviaria de alta velocidad.* PhD thesis, Escola Técnica Superior d'Enginyeria Industrial de Barcelona, 2010.

[20] David R Lide. *CRC handbook of chemistry and physics*, volume 85. CRC press, 2004.

[21] Philip C Magnusson, Vijai K Tripathi, Gerald C Alexander, and Andreas Weisshaar. *Transmission lines and wave propagation.* CRC press, 2000.

[22] Bernard Stanford Massey and Alfred John Ward-Smith. *Mechanics of Fluids: Solutions Manual.* Taylor & Francis, 2006.

[23] W. H. McAdams. Heat transmission, 3rd ed. new york. McGraw-Hill, 1954.

[24] Bassam Mohamed. *Power Flow Algorithms for Special Electric Networks Including Devices with Non-Linear and Non-Smooth Characteristics.* PhD thesis, Universidad de Oviedo, 2018.

[25] V. T. Morgan. The current carrying capacities of overhead line conductors paper a75 575-3,. IEEE PES Summer Meeting, Los Angeles, CA,, 1978.

[26] Dario Zaninelli Morris Brenna, Federica Foiadelli. *Electrical Railway Transportation Systems.* IEEE Press, 2018.

[27] Serm Murmson. What is solar altitude? *Sciencing*, May 2018.

[28] G. A.Alcan Mussen. The calculation of current carrying capacity of overhead conductors. Research and Development Limited, November 1966.

[29] B Umesh Rai. *Handbook of research on emerging innovations in rail transportation engineering*. IGI Global, 2016.

[30] F Schmid, CJ Goodman, and C Watson. Overview of electric railway systems. 2015.

[31] C. U. Schurig, O. R. Frick. Heating and current carrying capacity of bare conductor for outdoor service. General Electric Review, vol. 33, no. 3, pp. 141-157,, March 1930.

[32] House H. E. Taylor, C. S. missivity and its effects on the current carrying capacity of stranded aluminium conductors. AIEE Transactions, vol. 75, pt. III, pp. 970–976, October 1956.

[33] CAF TE. Generalidades sobre electrificación de líneas ferroviarias. Internal report CAF TE.

[34] CAF TE. Thermal calculation of the catenary temperature. Internal document.

[35] CER & UIC. Rail transport and enviroment: Facts & figures. *CER/UIC 2014*, 2015.

[36] Yi Yang, Ronald G Harley, Deepak Divan, and Thomas G Habetler. Thermal modeling and real time overload capacity prediction of overhead power lines. In *2009 IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives*, pages 1–7. IEEE, 2009.