University of Southern Queensland

Faculty of Health, Engineering and Sciences

System Dependent IDMT Settings: Direct Buried Underground Cable

A dissertation submitted by

Gregory Nagel

in fulfilment of the requirements of

Courses ENG 4111 and 4112 Research Project

Towards the degree of

Bachelor of Engineering - Electrical and Electronics

Submitted: October, 2014

ABSTRACT

As modern day power protection devices become smarter, more configurable and more accurate, they offer users the ability to configure the exact protection elements and settings required for a specific network section. This combined with the increased number of power systems going underground provides the basis for this research project. As we look to optimise our power networks and the protection of our power networks, we must ask, how can we ensure that our protection relays have the optimal settings for our underground network?

Presently, when determining the time-dependant over-current protection settings for underground cables, all worst-case scenarios are considered, resulting in excessive safety margins and over-protective configurations. Whilst these excessive safety assumptions ensure adequate protection for the electrical asset, they also potentially work to increase the possibility of false fault detection. Such an error leads to unnecessary supply isolation and consequently, costly downtime.

Given these limitations to current methods for determining protection settings, this research project develops and implements a simulation model using finite element analysis that analyses specific underground cable systems based on operating and environmental conditions. By determining the steady-state thermal profile of the underground cable system as a result of the load current, the simulation continues to analyse the effect of fault current on the system to determine the most suitable protection settings for the underground cable system.

The results presented herein outline the effect that environmental conditions have on the required protection settings of underground cable systems, and when used in conjunction with the simulation software, provide valuable information to assist design engineers making decisions on a system's setting values for numerical protection relays.

UNIVERSITY OF SOUTHERN QUEENSLAND
FACULTY OF HEALTH, ENGINEERING AND SCIENCES
ENG4111/ENG4112 RESEARCH PROJECT

LIMITATIONS OF USE

UNIVERSITY OF SOUTHERN QUEENSLAND
FACULTY OF HEALTH, ENGINEERING AND SCIENCES
ENG4111/ENG4112 RESEARCH PROJECT

CERTIFICATION OF DISSERTATION

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Gregory Dirk Nagel

0061025127

Signed

Date

ACKNOWLEDGEMENTS

Thank you to Associate Professor Tony Ahfock for allowing me to execute this project with autonomy and at my own pace. Your timely feedback was instrumental to producing this document.

Thank you to the teaching staff at USQ, especially Mr Chris Snook, for assisting with course related issues and for your understanding when extensions were required due to work commitments. The flexibility offered by USQ makes it an ideal institute for external study.

Thank you to my colleagues in the power industry for engaging in technical discussions and furthering my understanding of real world underground cable applications and challenges. I would also like to thank the two companies I have worked for throughout this undergraduate program, offering flexibility and support to ensure I met my commitments at USQ.

To my family and friends, thank you for sticking by me throughout my endeavours at USQ. Your support and sympathy helped me through the late nights and long weekends required to complete the coursework at the level I aspired to.

To my lovely Anna, words cannot express the support I have received from you throughout this journey. You have radiated success ever since I met you providing a constant source of motivation which has helped me to not only reach but exceed my goals. Thank you for your love, care and inspiration during the tough times. I owe you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

NOMENCLATURE

Short form	**Long form**
CT	Current Transformer
DTS	Distributed Temperature Sensing
F.E.	Finite Element
IDMT	Inverse Definite Minimum Time
IEC	International Electrotechnical Commission
IED	Intelligent Electrical Devices
IEEE	Institute of Electrical and Electronics Engineers
kV	Kilovolts
PE	Polyethylene
Ph-E	Phase to Earth
Ph-Ph	Phase to Phase
PVC	Polyvinyl Chloride
RMS	Root Mean Squared
TMS	Time Multiplier Setting
USQ	University of Southern Queensland
XLPE	Cross-Linked Polyethylene

1) INTRODUCTION

Today, underground cables are being used more frequently, particularly in the likes of new suburban developments and industrial installations such as coal seam gas plants (Bascom, 2011). This shift brings aesthetic benefits and reduces the likely impact of disturbances caused by bad weather (Navrud & Ready, 2008). Moreover, a report commissioned by the Australian government in 1997 found that the principle benefits of burying cables were reduced maintenance costs, the avoidance of tree trimming expenses, and the removal of the cost associated with motor vehicle accidents with power poles (Janick, 2000). Despite these benefits the report identified that only seven per cent of homes in Australia were currently served by underground power and that it would cost $50 billion to bury all existing cables underground. Despite the high cost associated with the replacement of overhead lines with underground cables, Janick (2000, p. 20) argues that these figures should be interpreted with care given this move would be 'replacing aged infrastructure with new, modern, energy-efficient systems'. Furthermore, he makes the point that the maintenance costs to these new systems should be very minimal for a number of years. However, if something were to go wrong, finding and restoring faults in an underground cable can be a significant challenge.

Cigre (2009) observes that locating a fault within an underground network can be time consuming and requires specialised equipment. First a fault must be located, then the cable must be excavated using vacuum trucks to avoid further damage to plant equipment, such as communication, power or gas lines buried in the vicinity. Once the cable has been exposed, the cable is then repaired using specialised cable jointing kits. Historical data suggests that rectification of an underground XLPE cable takes an average of 20 days (Cigre, 2009). Thus the procedure for repairing a cable can be long, onerous and costly for the owner, however, these expenses are often insignificant in comparison to the cost associated with the of loss of production or supply to consumers. For this reason, it is critical to determine the correct protection settings for the over-current devices protecting underground cables.

In the presence of fault current, ohmic heating due to the current flow through the resistance of the conductor material generates a temperature rise within the conductor and ultimately the insulating material. The protection settings must be configured to trip the upstream circuit breaker before the conductor temperature exceeds the maximum permissible temperature of the insulation. Often, to ensure this is achieved in all possible situations, significant safety margins are used to ensure the protection relay will operate the circuit breaker and clear the fault before the cable temperature becomes destructive. To achieve this, many general assumptions are made on the system to cover all situations and designers have no option other than to consider the worst case scenario based on the potential extremes of the environment and operating conditions (Naskar, 2013).

Considering all the worst case factors and configuring the protection system accordingly presents a new risk, the selectivity[1] of the network. Making assumptions and increasing safety margins also increases the risk of unnecessarily tripping and isolating sections of a network during the event of peak loading, overloading or transient faults resulting in the loss of production or supply to consumers. Furthermore, from my experience in the industry, a maintenance team must now be deployed to investigate the cause of the protection trip and verify the cables are fit for supply restoration. For example, the permissible fault levels may vary significantly depending on the conditions of the network prior to a fault existing, the seasonal variation of soil moisture content and temperature, or the condition of a network as it ages.

With this in mind, this research project will endeavour to create a computer model that will determine the thermal profile of a cable system during normal operation, and from this point determine the protection settings that would provide adequate protection to the cable system. This model will then be used to simulate a variety of cable systems with variations of one parameter to determine the effect this would have on the required protection settings and hence, the capacity of the cable system. These methods could also be used to dynamically configure the IDMT protection setting values based on real-time network loading and environmental conditions.

---

[1] The selectivity of a network is the ability to correctly determine that the component it is protecting is faulty and to isolate only that component from the rest of the power system (USQ ELE3804, 2013).

2) LITERATURE REVIEW

2.1) Overview and the need for further research

2.1.1) Product datasheets

Information on how and what products are used within the industry is critical to creating and verifying a realistic simulation model that will accurately analyse the thermal profile of a cable system. Product datasheets and application notes provide the basis for material properties and cable dimensions, whilst industry standards provide overarching information about cable systems and the areas in which they operate. Many standards are written to cover a wide range of industry products and applications which will provide a good basis for validating the simulation model, however, product datasheets often contain more specific information which, if used, will improve the accuracy of the simulation model. For this reason the following product datasheets have been selected and will form the basis of this research project:

- Gemscab – The right connection – HT-XLPE Cables.
- NKT Cables – High Voltage Cable Systems – Cables and accessories up to 550 kV.
- Tyco Electrics – Installation Instruction – Raychem Joint for Polymeric Insulated Cables.

2.1.2) AS and IEC standards

AS 3000:2007 – Wiring rules

The information from this standard was used to understand the legal constraints on underground cable installations to ensure the model reflects common applications within the industry.

AS 3008.1.1:2009 – Electrical Installations – Selection of cables

The information from this standard was used to validate the accuracy of the simulation model.

IEC 60255 – Measuring relays and protection equipment Part 1: Common requirements

This standard was used in conjunction with manufacturer information to determine standard protection curves and the equations associated with the curves.

2.1.3) Key articles influencing the simulation model

Statistical life data analysis for cable joints

Mehairjan's (2010) paper examines a 10kV underground power network with specific focus on faults that occur within the cables and cable joints. The investigation covers a variety of different cable joints including a detailed analysis of failure modes and failure rates. This publishing provides the basis for the statistical analysis of cable joints used in this research project.

Method for using finite elements to calculate temperature diffusion

Chapter 2 in Nikishkov's book outlines the method for using finite elements to calculate temperature diffusion within a 2-dimensional system with the inclusion of internal heating. The mathematics published will form the basis of the heat transfer model that will be adapted into a computer simulation program and used to determine the thermal properties of cable systems.

Calculating temperature rise and load capability of cable systems

The publication by Neher & McGrath (1957) provided a method for estimating the steady-state temperature of electrical power cables. The method is limited to generic cable configurations and uses a complex string of calculations to estimate the thermal conditions of a conductor. This method formed the basis for many ampacity de-rating tables and is a reoccurring reference in many works published around the analysis of underground power cables.

2.1.4) Distributed temperature sensing using fibre optics

A technology that should not be overlooked when determining how to best protect underground HV assets is the use of fibre optics to determine the temperature profile along a cable run. To achieve distributed temperature sensing (DTS), a fibre optic cable is embedded within the HV cable, or next to the cable in a trench. Light is then pulsed along the fibre and the effect of light scattering causes a small portion of the light to reflect back as it travels down the fibre. The amount of light scattered, and therefore reflected, is dependent on the temperature of the medium through which it is travelling (Peck & Seebacher, 2000). This technology offers real-time monitoring of a cable system and can help to detect the onset of hotspots within the cable system. Williams (1999), states that often cable systems are loaded with a 10% safety margin to account for the worst case conditions. It also outlines a cable installation that was retrofitted with DTS allowing the operation to increase the load by 8% using the real-time thermal monitoring. Figure 2.1 shows the magnitude of temperature variation along a cable system and highlights the criticality of understanding the environmental conditions throughout the entire cable run.

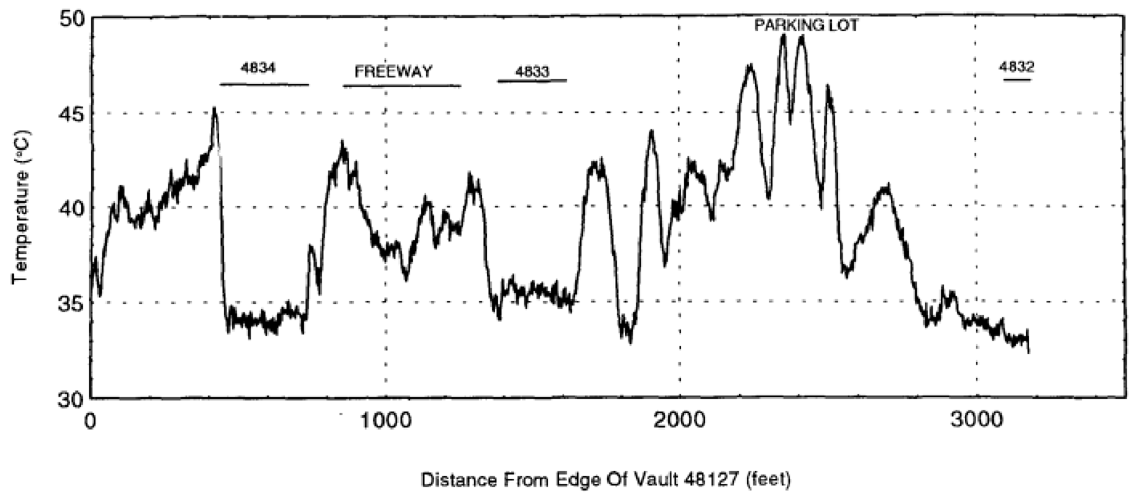


Figure 2.1 – DTS analysis of an underground cable system (Williams, 1999)

2.1.5) Similar published works

Nguyen (2010)

Nguyen uses a mesh approach to determine the temperature rise and ampacity of underground cables as shown in Figure 2.2. The author states that a mesh approach was adopted to maintain accuracy whilst reducing the processing time of the model. The model used here predetermines the power output from the cable and disperses this power evenly about the outer circumference of the cable system. While this method will provide valuable information on the thermal field surrounding the cable system, it does not analyse the temperature profile within the cable to accurately determining the maximum temperature of the conductor and therefore the potential for material damage within the cable.

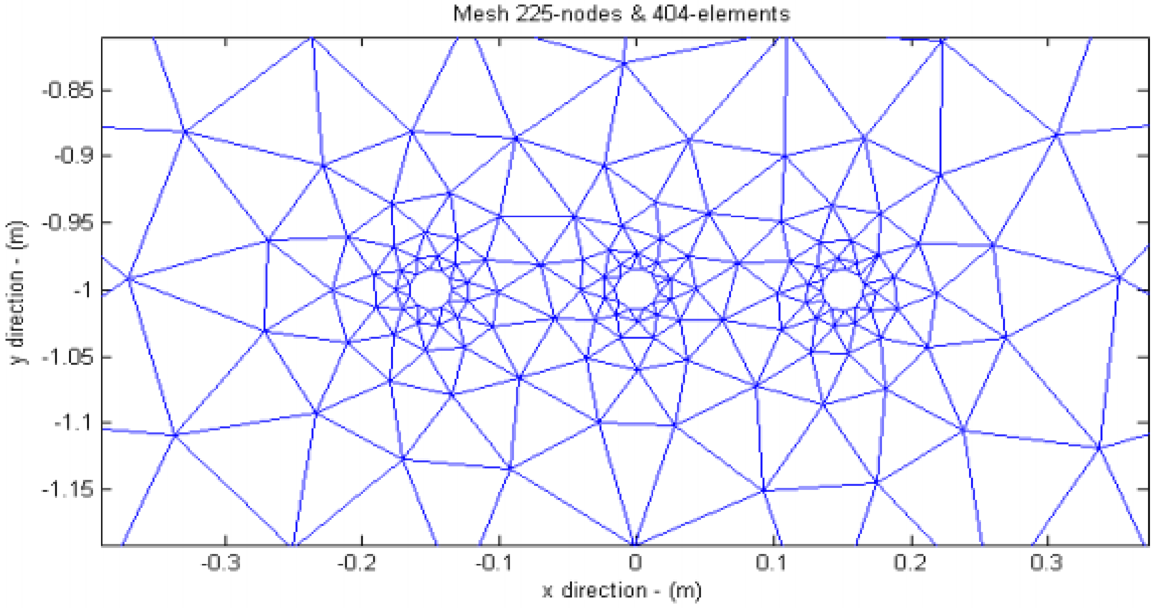


Figure 2.2 – Discrete field domain using mesh layout (Nguyen, 2010, p. 3)

Zang (2012)

Zang, uses triangular mesh nodes, similar to that used by Nguyen, to determine the current rating of cables as shown in Figure 2.3. The model created by Zang is used to complement temperature sensors placed near underground duct banks. By understanding the temperature profile of the system, a more accurate assumption could be made about the temperature of the cable using results from indirect temperature sensors.

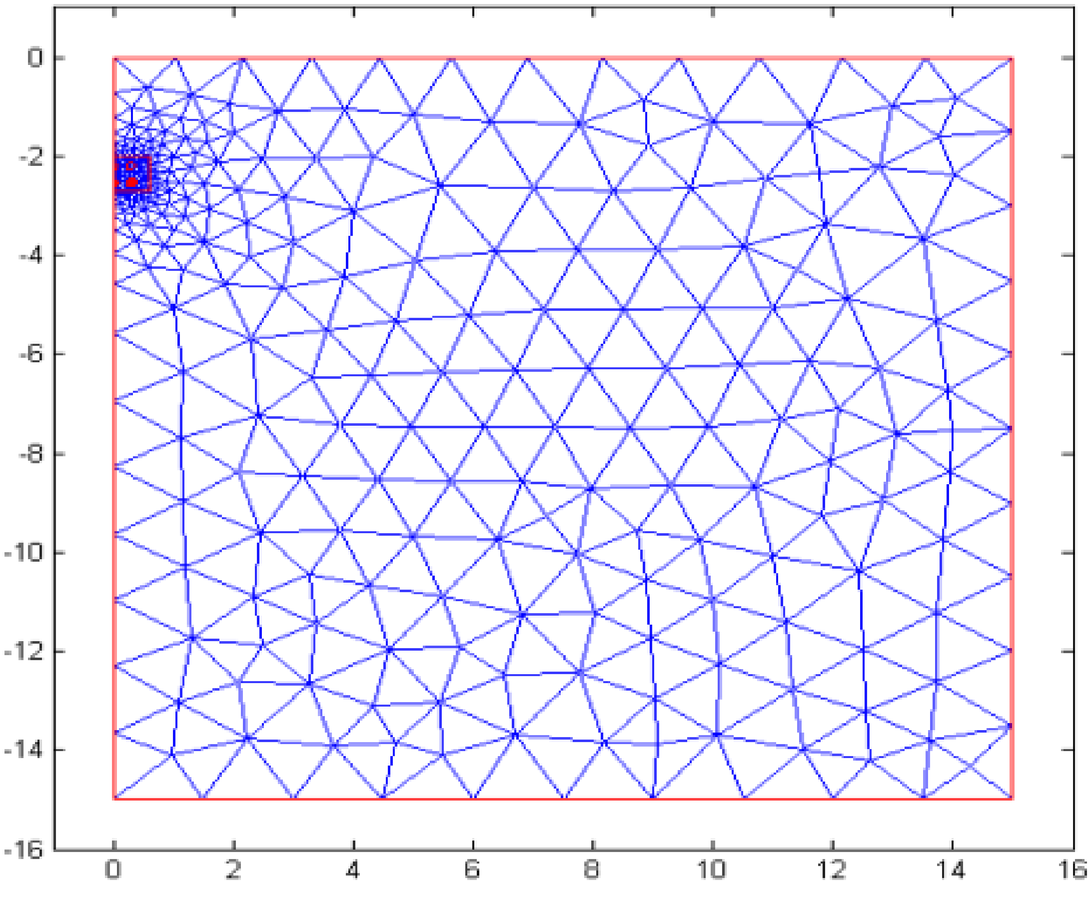


Figure 2.3 – Mesh outlines of finite elements in Zhang's model (Zang, 2012, p. 4)

Naskar (2013)

Naskar uses an arc approximation to determine the current rating of cable layouts, shown in Figure 2.4. Naskar acknowledges the fact that approximations and assumptions lead to inaccuracies in the calculations and often force cable engineers to use unnecessarily large safety factors leading to over-conservative designs. The model developed by Naskar uses finite elements to model a 6.6 kV 3-core underground cable. The symmetry of this model limits the applications as it could not be used to interact with the heat generated by other near-by cable systems.

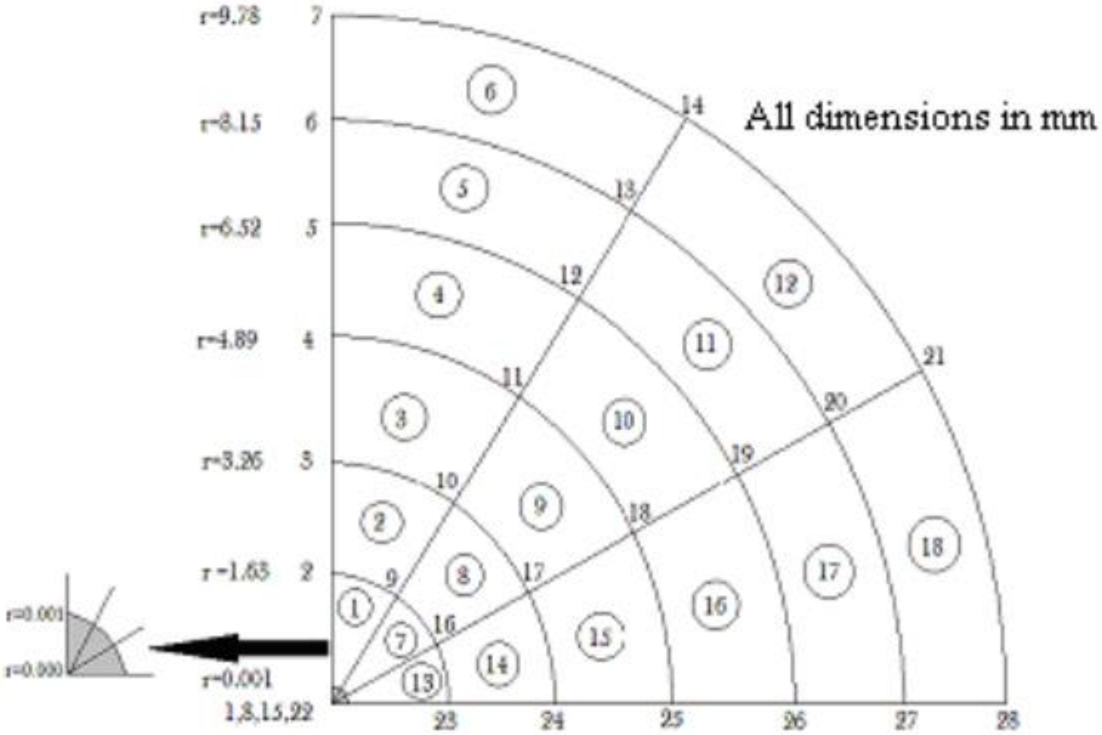


Figure 2.4 – Two-dimensional arc analysis (Naskar, 2013, p. 99)

2.1.6) Extension beyond these works

The ultimate goal of this research project is to develop a simulation program that can be used to determine the IDMT settings and the current capabilities of direct buried underground cable systems. The works mentioned above model the temperature profile of cable systems which is useful in determining the ampacity of the cables, however, they do not consider the requirements for protecting the cables and ensuring they operate within the specified operating limits at all times, especially during fault scenarios.

Cable protection will be the primary focus as this research project endeavours to extend the above works. Another dissimilarity to the above works is the use of a square matrix of finite elements which will increase the configurability for the user and allow the system to cover more cable system variations. As part of the research, simulations will be conducted on the cable variations to gain an understanding of how environmental and operation changes within the cable system affect the operational capacity and minimum protection settings of cable systems. This information may assist protection engineers as they undergo protection studies to determine the required protection settings of underground cable systems.

2.2) Fundamentals of underground cable systems

2.2.1) Underground cable construction

The construction of underground power cables vary depending on the intended application, however, the common components are best shown as a single core cable in Figure 2.5. Whilst some elements shown here may not be present in all cable designs, the critical components to provide basic function of a power cable are; a core conductor which will carry the load current, a screen which provides an electrical return path for any insulation failures, and an insulating medium to encompass the voltage potential of the core conductor. These components are extruded within the outer sheath, typically polyvinyl chloride (PVC). Important specifications of the cable are the voltage rating and current carrying capability which are determined, respectively, by the properties of the insulator and the core conductor.

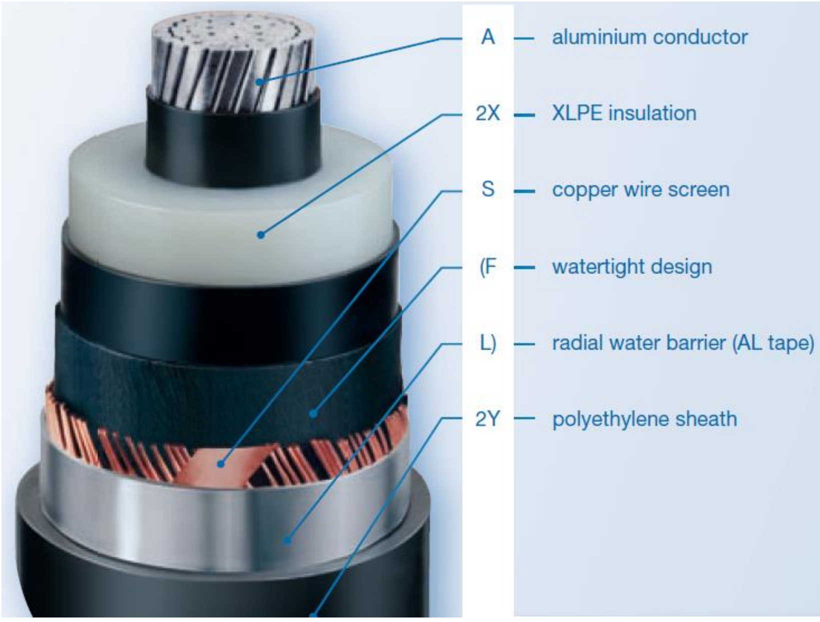


Figure 2.5 – Single Core XLPE Cable (NKT cables, 2009)

The core conductor, usually soft copper or pure aluminium, carries the load current to the downstream equipment. Due to the inherent properties of the conductor material, as current flows through the core conductor, heat will be generated within the metal. The conductor cross-section is a major factor determining the current rating of a cable as the resistance and therefore the power loss within the conductor will increase at a reduced cross-section due to the removal of available paths for electron drift.

The outer shield is bonded to potential earth at one or both ends of the cable run providing a critical electrical path for fault current to flow should the insulating medium break down or damage occur by an external source such as an excavator. The fault current is then detected by a protecting device monitoring the system and isolated from its source to ensure safe voltage potential around the cable system in the event of a failure.

A semiconductor layer exists at both the inside and outside of the insulating medium. This layer ensures a uniform electric field exists across the insulation. If the electric field within the insulation is not uniform, points of increased electric field can induce excessive stress on the insulation leading to early fatigue and failure.

The insulating material used and the thickness of the insulation determine the magnitude of the electric field that can be sustained between the conductor and the shield and therefore the voltage limitations of the conductor. The insulation also provides a means for heat to be drawn away from the core conductor and into the surroundings. The maximum permissible temperature of the insulation is much lower than that of the conductor material and therefore quantifies the maximum temperature at which core conductor can operate, ultimately defining the load and fault current levels of the cable. This research project will focus on cross-linked polyethylene (XLPE) insulated cables which were first developed in the 1930s. XLPE is commonly used in power cables as it has excellent dielectric properties making it useful for a large range of voltage applications from 600 V to 500 kV (Orton, 2013).

2.2.2) Impurities in cable construction

As with all manufacturing processes, cable manufacturing has a risk of defective products. One issue that can result from poor manufacturing is the presence of physical voids within the insulation. This can result in a reduced service life for the cable due to the reduction of insulating dielectric between the conductor and shield. The voids create sections where the electric permeability is reduced and the voltage gradient across the void is low in comparison to healthy insulation. This increases the voltage potential across the remaining insulation thus increasing the electrical stress on the insulation. Testing of the insulation at the manufacturing site is critical to ensure that the cable meets the required standards and is fit for purpose. However, it is never possible to manufacture the perfect cable and the presence of voids and insulation impurities are common sources for the initiation of breakdown in cables (Mehairjan, 2010).

2.2.3) Cable Joints

On-site cable jointing is often required as there are limitations to the maximum drum size that can be transported to installation sites. For example, a 10 km cable run using cables that have a drum length of 400m, will require 25 cable joints. Over time, these joints may deteriorate due to environmental and electrical conditions. This can lead to an increased resistance and above average temperatures at the cable joint, accelerating the deterioration of the cable’s insulation. Once the insulation is damaged, the electric field can no longer be contained between the conductor and the screen and the faulted section must be repaired. In my experience, this requires cutting out and replacing a 10m section of the cable and the addition of another joint and another potential point of failure.

According to Mehairjan (2010), cable joints are subject to more failures than the cable itself for the following reasons:

- they are subject to higher electrical, mechanical and thermal stress
- they are mounted in the field under non-ideal circumstances, particularly during outage situations
- they are not subjected to extensive reliability testing procedures like the cable itself
- the quality of installation of the accessories is reasonably sensitive to workmanship, experience and care of the involved employee.

Figure 2.6 shows data extracted from Mehairjan's (2010) statistical analysis of an underground power network which reveals that the majority of internal failures[2] occur at the location of cable joints.

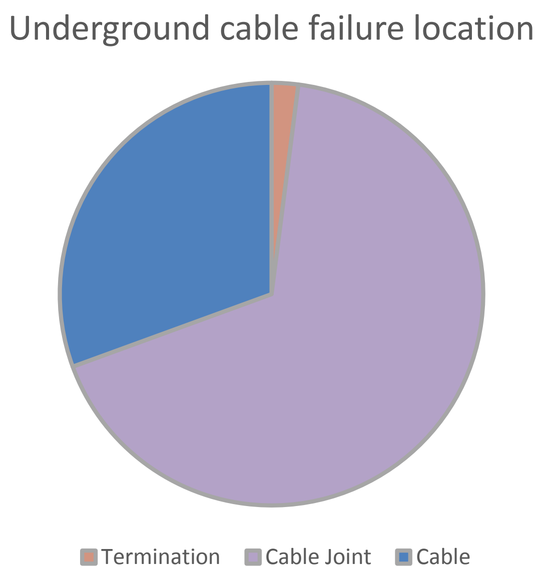


Figure 2.6 – Internal failures on an underground HV network (data: Mehairjan, 2010)

2 those failing under normal operation without external influences i.e. excavation error

Figure 2.7 shows a typical cable joint. As depicted, the shield of the cable is moved to one side of the joint to allow for the insulating heat-shrink to be placed over the conductive joint. This reduces the integrity of the electric-field distribution and can result in increased electrical stress on sections of the insulation. Another challenge that lies with cable joints of this nature is ensuring the joints are water tight as water ingress is a common cause of failures at these joints (Megger, 2003).

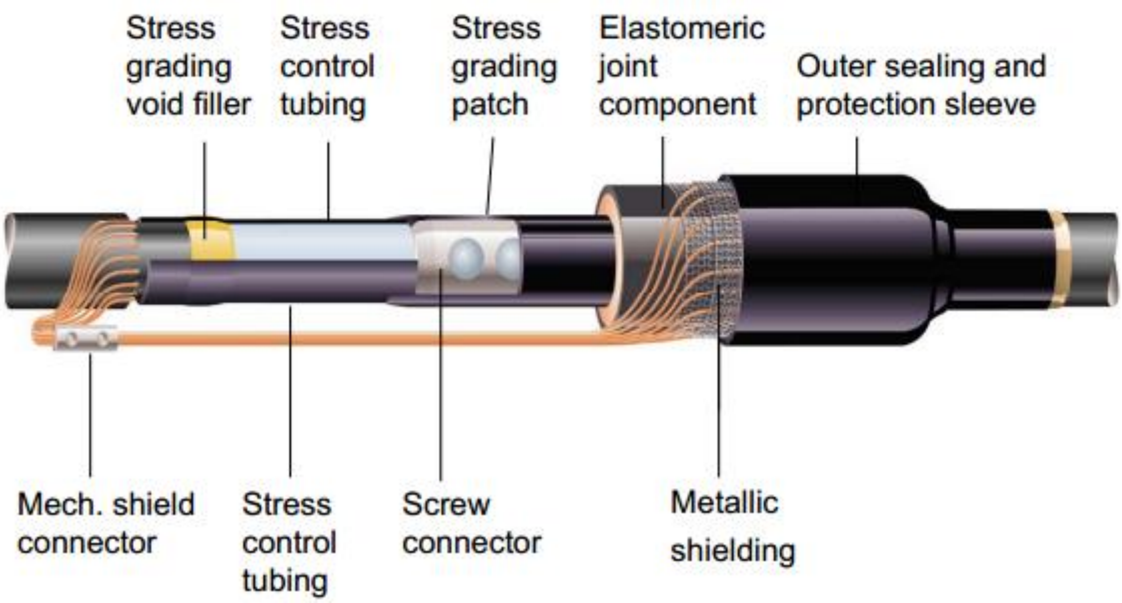


Figure 2.7 – Cut-away representation of a cable joint (Tyco Electronics, 2000, p. 11)

Quantifying the resistance of cable joints is difficult as there is no method for testing the joint resistance without destroying the cable. Fournier & Amyon, (2001) measured the resistance for a healthy electrical cable joint to be 15 μΩ while workmanship defects such as insufficient torque values or incorrect crimp settings increased the resistance to 48 μΩ.

An interesting phenomenon that exists within defective electrical joints is the effect of self-healing (Fournier, 1998). When a cable joint is degrading, hotspots form which lead to microscopic melting at the point of high impedance. This can resulting in welding, or self-healing, of the substandard cable joint. Fournier also noticed fluctuations in the contact resistance of the cable joint during this phase, resulting in resistance instabilities and unpredictable thermal profiles. Due to this effect, Fournier outlines the unreliability of results when periodically performing infra-red scanning of cable joints.

2.2.4) Operational and environmental stress

As the underground cable system ages, it is exposed to operational and environmental stress. These increase the likelihood of internal defects and ultimately cable faults. Figure 2.8 shows the increase in cable joint failures as the service life of the cable reaches 20-40 years.

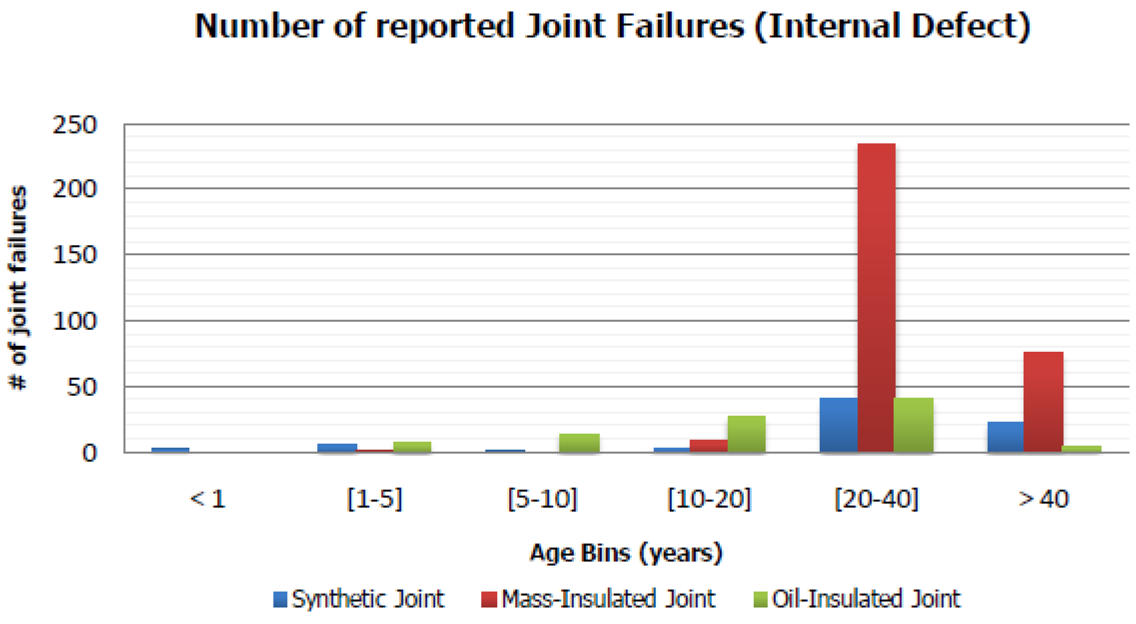


Figure 2.8 – Joint failures over time (Mehairjan, 2010, p. 66)

Environmental conditions, such as; ground humidity, ground pollution, thermal resistance of surrounding material, ambient temperature at the surface of the cable, can all contribute to the degradation of the cable system (Megger, 2003, p. 2). Moisture can penetrate the cable insulation decreasing the insulating properties, this is a major issue at cable joints as water can track under the additional joint insulation. In many cases, the location of the underground cable is determined by the layout of electrical plant or existing infrastructure and this defines the environmental conditions the cable will be exposed to. The effect of this can be reduced by using bedding sand with known and consistent thermal properties when backfilling the cable trench.

Throughout an average day, it is likely that the operating conditions of a cable will change. This is often due to daily load cycles and peak loading conditions which result in the fluctuation of the conductor's temperature. These thermal fluctuations cause cable materials to expand and contract imposing mechanical stress on the cable joints. Over time, the mechanical forces may lead to a reduction in contact area and an increase in resistance. An increased joint resistance will result in a 'hotspot' which is likely to accelerate the deterioration of insulation properties. Sudden temperature increases due to over-current conditions result in high temperatures within the cable, especially if this occurs with a high initial temperature following a period of heavy loading. As the conductor undergoes sudden temperature changes, movement can stress the XLPE material making the material more brittle and increasing the risk of void formation within the insulation.

This research project endeavours to utilise the operating and environmental conditions of a cable system to determine the required protection settings to avoid excessive operational stress that may lead to rapid degradation of the cable insulation.

2.2.5) Protection systems for underground cables

Modern numerical protection relays offer multiple protection elements in one device. Protection relays are connected to current and voltage transformers which provide linear conversions from the system level (i.e. 800 A, 11 kV) to levels measurable by sensitive analogue to digital converters within the device (i.e. 1 A, 110 V). The numerical protection relay uses these measured values in conjunction with protection algorithms to monitor power systems and determine when they are operating outside of permissible limits. Once abnormal operation is detected, the protection device will issue a trip command to the relevant circuit breaker isolating the supply to the faulted part of the network.

Time dependant over-current – 51

The most common protection applied to a feeder is over-current protection. This protection uses the measured current values to determine if the current flowing to the downstream equipment is acceptable or if the levels exceed normal operation. In this scheme, time delays are used to discriminate against faults that may exist within another protection device's primary protection zone as it is desirable for the protection device closest to the fault to isolate the fault and reduce the extent of the supply outage.

Instantaneous over-current – 50

Instantaneous over-current protection uses measured current values to determine when the current in the system has exceeded a specified threshold. If this occurs the relay will initiate a trip command instantly, i.e. without the use of a protection curve.

Earth fault – 51N

A significant unbalance in the 3-phase current vectors shows that current is leaking to earth and exiting the 3-phase system. Historically this was monitored by using a current transformer measuring the summated current flowing through all conductors, however, modern numerical relays are able to virtually summate the three individual current vectors and determine if an unacceptable amount of current is leaking to earth and initiate a protection trip accordingly.

Line differential protection – 87L

Line differential protection is a unit protection scheme with the ability to confidently detect any fault within the zone it is protecting whilst ignoring any faults outside the zone. A current transformer is placed at either end of the underground cable defining the protection zone. Current values measured by the upstream and downstream protection relays are communicated between the protection relays, usually by sending digital current values over a fibre-optic communication link. When the protection relay detects a significant discrepancy in current values, the protection will operate.

Distance protection – 21

Distance protection uses the measured current and voltage waveforms to calculate the impedance of the downstream system. The magnitude of the current and voltage are used to determine the value of the impedance whilst the phase shift between the current and voltage waveform is used to determine the ratio of resistance to reactance. These values are used to determine if a downstream fault is within the primary zone of the protection relay, initiating a trip command instantly, or if the fault exists within another device’s protection zone, allowing sufficient time for the downstream protection to clear the fault prior to initiating a trip command.

Underground cable protection summary

In underground cable systems, differential protection is the most reliable scheme for clearing faults that occur within the underground cable as the protection system can be certain fault current is escaping between the two current transformers. If a fault was to occur outside the differential protection zone, the differential scheme would still register $I_{in} \approx I_{out}$ and would not offer any protection to the underground cable. In this case, over-current protection would be required to determine if the upstream circuit breaker should trip. This is usually determined by an IDMT curve to allow for longer tripping times at lower fault levels as the fault may be transient or cleared by a downstream protection device. This would see the current values return to normal operating levels before damage occurs to the cable system, thus maintaining supply to the downstream system.

As discussed earlier, the point at which damage may occur to equipment depends on the condition of the equipment prior to the fault occurring. To be safe, protection engineers will often take the worst-case conditions to determine the protection settings of the numerical relay. This will ensure safe protection under all conditions, however in many situations, the system will be over-sensitive and may unnecessarily isolate supply to the downstream equipment. To overcome this, engineers require smarter, more sophisticated tools to better understand and analyse the protection levels required to accurately protect aspects of a power system.

2.2.6) Inverse Time Protection Curves

Figure 2.9 shows how fault current relates to the tripping time in an IDMT scheme. To interpret data from this graph, the reader should track up from the fault current level to the curve and across to find the protection operating time. This is the time the protection system will allow a fault of this magnitude to be present on the system before the protection will operate. It is important to note that the circuit will not isolate instantly as there is a mechanical delay for the circuit breaker mechanism to open the contacts enough to extinguish any arcing.

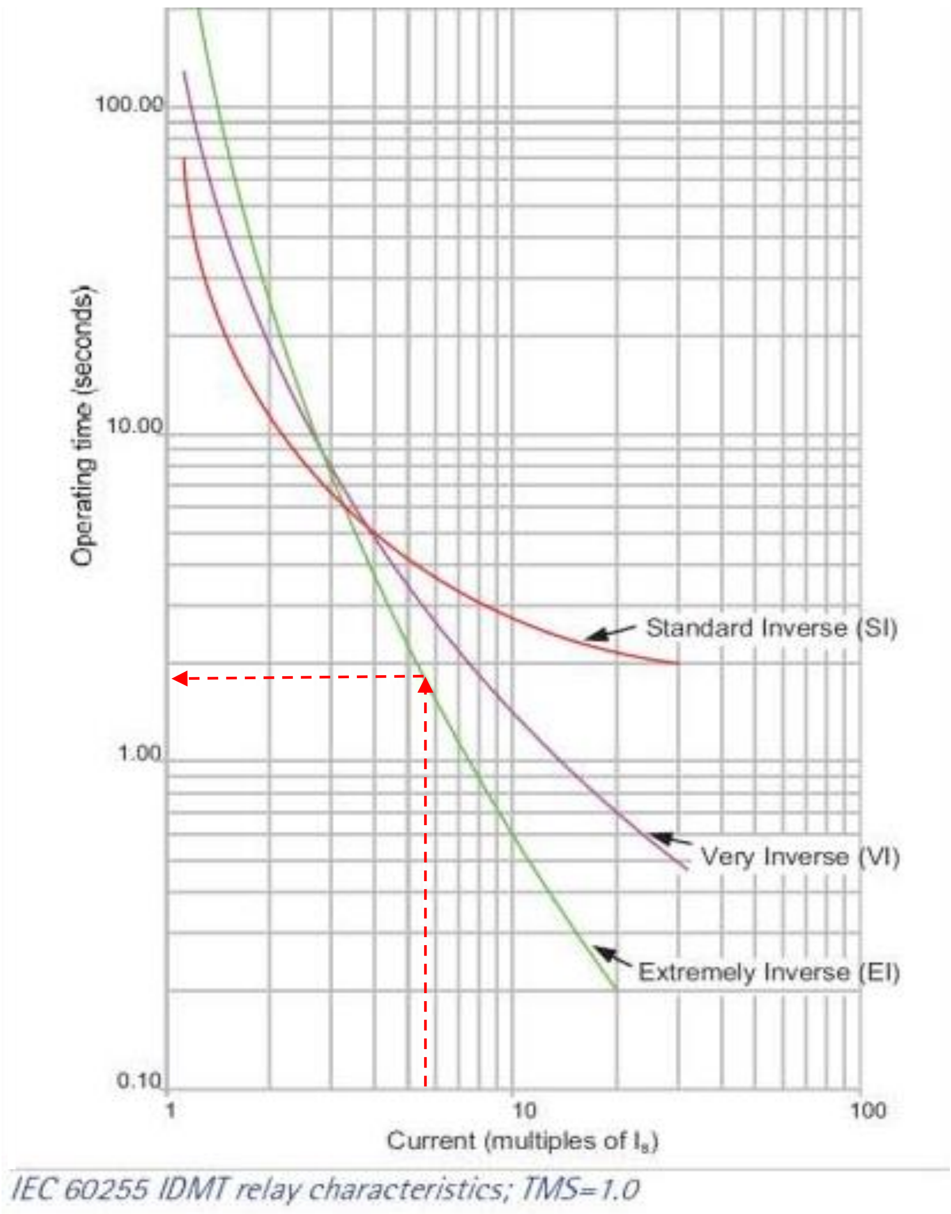


Figure 2.9 – IEC 60255 tripping curve characteristics (My Electrical, 2014)

Both the IEC and IEEE have standards outlining generic inverse definite minimum time (IDMT) tripping curves. These curves are typically drawn on a logarithmic scale plot with current along the horizontal axis and time on the vertical axis. Table 2.1 outlines the coefficients that vary the shape of the IEC protection curves shown in Figure 2.9. Once the curve shape has been determined, two variables, pick-up current and time multiplier setting (TMS), define where the curve will sit within the axis. The TMS shifts the curve upward to provide an additional time delay across all values whilst the pick-up current forms a vertical asymptote representing the maximum continuous current level shifting the curve left or right. The following equations and coefficient values shall form the basis of the IDMT curves that will be used within the simulation model.

$$td(I) = \frac{k}{\left(\frac{I}{Is}\right)^{\alpha} - 1} \times \frac{T}{\beta}$$

Equation 2.1 – IEC IDMT curve equation (Schneider Electric)

IEC curves			
Curve type	**Coefficient values**		
	k	α	β
Standard inverse / A	0.14	0.02	2.97
Very inverse / B	13.5	1	1.50
Long time inverse / B	120	1	13.33
Extremely inverse / C	80	2	0.808
Ultra inverse	315.2	2.5	1

Table 2.1 – IEC IDMT curve coefficient values (Schneider Electric)

$$td(I) = \left(\frac{A}{\left(\frac{I}{Is}\right)^p - 1} + B \right) \times \frac{T}{\beta}$$

Equation 2.2 – IEEE IDMT curve equation (Schneider Electric)

IEEE curves				
Curve type	**Coefficient values**			
	A	**B**	**p**	**β**
Moderately inverse	0.010	0.023	0.02	0.241
Very inverse	3.922	0.098	2	0.138
Extremely inverse	5.64	0.0243	2	0.081

Table 2.2 – IEEE IDMT curve coefficient values (Schneider Electric)

2.3) Physical properties of materials

2.3.1) Conductor

The most common materials used for underground conductors are copper and aluminium. Aluminium is often used as it offers a cheaper solution with a lower material and transportation cost compared to copper. The conductor cross-section and resistance values shown in Table 2.3 will form the basis for the resistance values used within the simulation model. A DC resistance value is used as the power loss within the conductor will come from the active current component.

Conductor cross-section (mm^2)	Conductor material	DC resistance Ω/km		Current Ratings (A)
		20 °C	90 °C	
185	Cu	0.0991	0.1270	368
	Al	0.1640	0.2110	289
240	Cu	0.0754	0.0973	420
	Al	0.1250	0.1610	332
300	Cu	0.0601	0.0781	469
	Al	0.1000	0.1290	371
400	Cu	0.0470	0.0618	525
	Al	0.0778	0.1010	420
500	Cu	0.0366	0.0492	586
	Al	0.0605	0.0791	474
630	Cu	0.0283	0.0393	649
	Al	0.0469	0.0622	533
800	Cu	0.0221	0.0326	706
	Al	0.0367	0.0500	591
1000	Cu	0.0176	0.0232	999
	Al	0.0291	0.0375	791
1200	Cu	0.0151	0.0201	1074
	Al	0.0247	0.0319	859
1400	Cu	0.0129	0.0175	1155
	Al	0.0212	0.0275	929
1600	Cu	0.0113	0.0156	1226
	Al	0.0186	0.0240	997
1800	Cu	0.0101	0.0142	1285
	Al	0.0165	0.0213	1058
2000	Cu	0.0090	0.0129	1346
	Al	0.0149	0.0193	1114

Table 2.3 – Standard design conductor properties (NKT cables, 2009, p. 9)

2.3.2) Insulating media

The insulating material surrounding the conductor of the cable dissipates the heat from the conductor core. The rated maximum conductor temperature of XLPE-insulated cables is 90°C (AS 3008, 2009). Cables insulated with XLPE also have a permissible conductor temperature, during a five second short-circuit fault, of 250°C (Orton, 2013). The thermal transfer through the insulating material is affected by the thickness of the insulation which is dependent on the voltage rating of the cable.

Voltage rating Ph-E/Ph-Ph (kV)	XLPE Thickness (mm)	Reference
1.9/3.3	2.2-3.0	(Gemscab, 2014) & (Nexans, 2010)
3.8/6.6	2.5-3.6	
6.35/11	3.4-5.5	
12.7/20	5.5-6.0	
19/33	8.0-8.8	
76/132	14-22	(NKT cables, 2009)
127/220	19-25	
230/400	26-33	
290/500	31-35	

Table 2.4 – XLPE insulation thickness

2.3.3) Outer sheath and fill material

The outer sheath of the cable is generally constructed with polyethylene (PE) or polyvinyl chloride (PVC). The thickness of the outer sheath is normally within the range of 1.8 to 4.0mm depending on the intended application (Gemscab, 2014). Due to the shape of 3-core cables, gaps exist between the inner cores, these are typically filled with PVC (Gemscab, 2014, p. 4).

2.3.4) Cable joint

The physical dimensions of the modelled cable joint are based on the Raychem joint for polymeric insulated cables with wire shields (Tyco Electronics, 2009). The joint part, MXSU-3341, is specifically designed for cables with a cross-section ranging from 185 - 400 mm^2.

This is an aluminium part with a connector diameter of 37mm and a length of 140mm. An example of this part is shown in Figure 2.10. The bolts used to secure the conductor ends are shear bolts and break away from the structure when the correct torque is reached.

Figure 2.10 – Typical conductive region of a cable joint (Tyco Electronics, 2000)

2.4) Thermal properties of materials

To calculate the steady state temperature of an underground conductor, a balance must be achieved between heat generated within the conductor and heat transferred through the insulation into the surrounding environment. The rate of thermal energy transfer is dependent on the thermal properties of the media through which it is diffusing. It is therefore important to ensure that the model accounts for the respective thermal properties of the different media and any material variation that may arise as temperature changes.

2.4.1) Thermal conductivity

The thermal conductivity, k, of a material is the rate at which heat will transfer throughout the material. Copper, followed closely by aluminium, has a high thermal conductivity allowing heat to quickly pass through it. This will cause heat to transfer and stabilise more quickly within the conductor material. Porous material, such as dry sand, has a very poor thermal conductivity as all of the pores are full with air. As the soil saturates, these voids are filled with water significantly changing the thermal conductivity of the material (TeKa, 2014, p. 1).

Thermal Conductivity, k (W/(m.K))				
Material	Temperature (°C)			Reference
	25	125	225	
Aluminium	205	215	250	(The Engineering Toolbox, 2014a)
Copper	401	400	398	
	Temperature (°C)			Reference
	20	55	90	
XLPE	0.223	0.267	0.280	(Lee, Yang, Choi, & Park, 2006, p. 806)
Semiconductor	0.552 - 0.587	0.631 - 0.673	0.631 - 0.673	
PVC	0.19			(The Engineering Toolbox, 2014a)
	Dry	Wet		Reference
Bedding	6.5	12.5		(TeKa, 2014)
Soil	0.03	0.6		
	Temperature (°C)			Reference
	20	40		
Air	0.0243	0.0271		(The Engineering Toolbox, 2014e)

Table 2.5 – Thermal conductivity of simulation materials

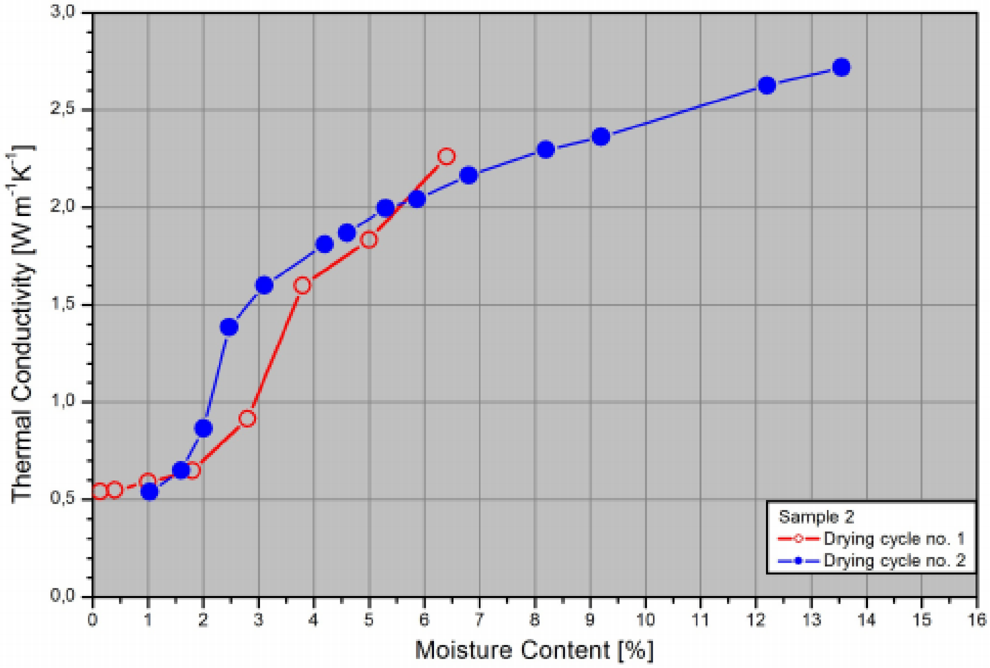


Figure 2.11 – Thermal conductivity of sand vs. moisture content (TeKa, 2014, p. 4)

2.4.2) Specific heat

The specific heat, $c_{p,}$ is the amount of heat energy, per unit mass, required to raise the temperature of an object by a one degree Celsius. In the case of this model the specific heat is important, especially in the conductor material, to determine the temperature rise with respect to the power loss within the conductor. The specific heat of the insulating material varies with respect to temperature, as shown in Figure 2.12. The initial increase is due to the volume expansion of the material, however, as the material heats closer to melting point, more thermal energy is required to change the physical state of the material (Lee, Yang, Choi, & Park, 2006, p. 808). To ensure the model uses accurate values with respect to temperature, the material's thermal properties are dynamically updated as the system changes temperature.

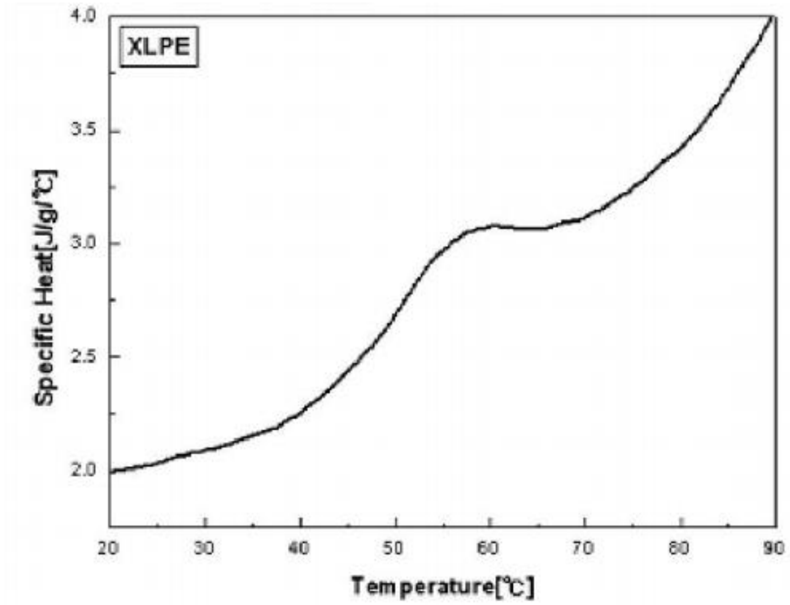


Figure 2.12 – Specific heat capacity of XLPE insulation (Lee, 2006, p. 806)

Specific heat, c_p (J/(g.K))				
Material	Temperature (°C)			Reference
	20	50	90	
Aluminium	0.897			(The Engineering Toolbox, 2014b)
Copper	0.385			
XLPE	2.034	2.976	4.049	(Lee, Yang, Choi, & Park, 2006, p. 806)
Semiconductor	1.6	2.39	-	
PVC	0.840 – 1.170			(The Engineering Toolbox, 2014b)
Bedding	1.480			
Soil	0.800 - 1.480 (moisture dependant)			
Air	1.005			

Table 2.6 – Specific heat of relevant materials

2.4.3) Volumetric mass density

The volumetric density, ρ, of the materials present in the simulation are required to calculate the thermal diffusivity of the material.

Material	Density, ρ (g/cm^3)	Reference
Aluminium	2.712	(The Engineering Toolbox, 2014d)
Copper	8.940	
XLPE	0.92 - 0.948	(Hampton, Hartlein, Lennartsson, Orton, & Ramachandran, 2012)
Semiconductor	1.4 - 1.5	
PVC	0.769 - 0.833	(The Engineering Toolbox, 2014c)
Bedding	1.522	(AgriInfo, 2011)
Soil	1.1 - 1.6	
Air	$(1.293 - 1.127)\text{x}10^{-3}$ (0°C to 40°C)	(The Engineering Toolbox, 2014e)

Table 2.7 – Thermal conductivity of relevant materials

2.4.4) Thermal Diffusivity

Thermal diffusivity, α, quantifies a material's ability to conduct thermal energy relative to its ability to store thermal energy and is governed by Equation 2.3 which combines the values listed above.

$$\alpha = \frac{k}{c_p \rho} \ (\mathrm{m}^2/\mathrm{s})$$

Equation 2.3 – Thermal diffusivity of a material

Using the worst-case values from the above tables, the values in Table 2.8 were calculated.

Thermal Diffusivity, ρ (10^{-6} m^2/s)		
Material	Minimum	Maximum
Aluminium	84.08	102.54
Copper	116.5	115.6
XLPE	0.058	0.150
Semiconductor	0.1540	0.3004
PVC	0.1949	0.2941
Bedding	2.886	5.549
Soil	0.0127	0.682
Air	0.0174	0.0239

Table 2.8 – Thermal diffusivity of relevant materials

2.5) Rating factors

Rating factors are used by engineers to extend beyond a standardised set of values and provide a higher level of accuracy when analysing a specific system. These factors help to minimise the assumptions and generalisations which lead to errors when calculating the capacity of underground cables (AS 3008, 2009). The rating factors include but are not limited to; the effects of air and ground temperature, the depth of cable lay, heating from neighbouring cables and variations between three-phase and single-phase cable construction (Gemscab, 2014, p. 8). The simulation model developed in this research project will provide the user with enough configurability to omit the need for applying rating factors as the simulation model will generate results based on the specific properties of the system.

2.6) Statistical representation of cable joint failures

Cable joints are subject to many external factors which can cause degradation and reduce service life. Due to the complex nature of these factors and variation between cable systems, a statistical rather than system dependant approach has been adopted in an attempt to predict the health of the cable joints within a cable system. Mehairjan's (2010) research into the failure rate of cable joints in a 10kV underground cable system will be utilised to provide information for a statistical model. The probability distribution shown in Figure 2.13 will provide the basis to determine the minimum life expectancy of a system containing synthetic cable joints.

This statistical analysis has many limitations as it is based on one set of data and many generalisations must be made to relate this data to all cable installations. Statistical information in the field of underground polymeric cable joint failures is very limited as it is a relatively new technology. More data will become available as cable installations age, however, it will be difficult to accurately apply this historical data to new cable installations as technology within the field of underground power cables is continually advancing through improved materials, manufacturing and installation techniques. Taking this into consideration, a statistical analysis of cable joints will be included in the simulation to provide users with a guideline for determining the health of a system by estimate the statistical worst-case joint condition. This information may be useful to evaluate protection setting adjustments or to determine preventative maintenance schedules.

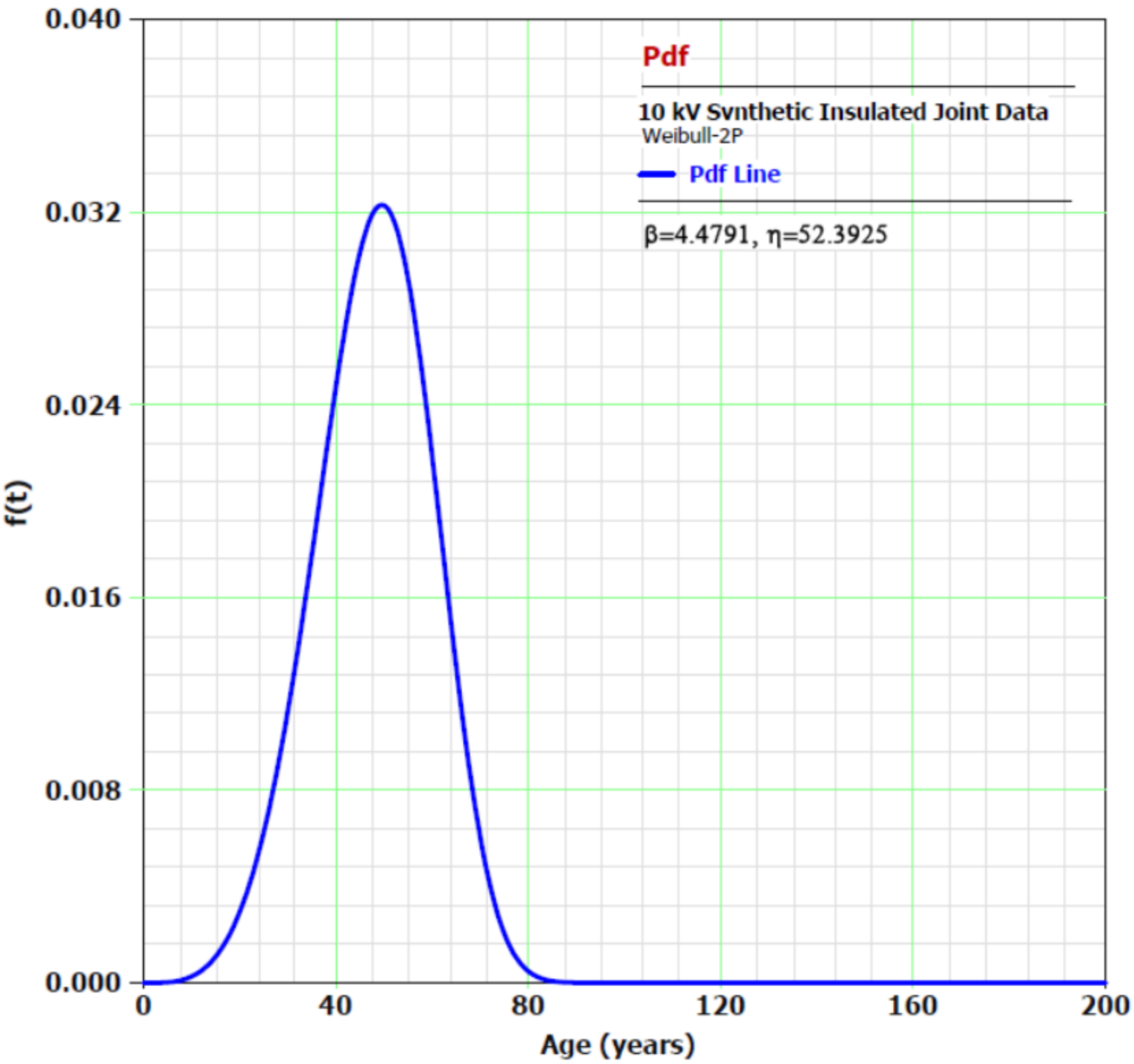


Figure 2.13 – Probability density function of synthetic cable joint failures

3) DESIGN AND METHODOLOGY

3.1) Thermal transfer model

3.1.1) Thermal diffusion using discrete finite elements

Heat transfer throughout a system, over time, can be modelled using a combination of Fourier's law of heat flow and a basic two dimensional equation of heat transfer (Nikishkov, 2010, p. 13).

$$-\left[\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}\right] + Q = \rho c_p \frac{\partial T}{\partial t}$$

Equation 3.1 – 2-D representation of a function of T in space and time

Where q_x and q_y are components of heat flow through the unit areas and Q is the rate of internal heat generation per unit volume. According to Fourier's law, the components of heat flow can be expressed as follows (Nikishkov, 2010, p. 13):

$$q_x = -k\frac{\partial T}{\partial x}, \qquad q_y = -k\frac{\partial T}{\partial y}$$

Equation 3.2 – Fourier's law of heat flow

Combining Equation 3.1 and Equation 3.2 yields:

$$\frac{k}{\rho c_p}\left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right] + Q = \frac{\partial T}{\partial t}$$

Equation 3.3 – Heat transfer equation

In order to simplify the system described by Equation 3.3, partial differential equations can be transformed into finite difference equations. For this transformation, the explicit approach will be used to solve for one unknown at a time. The central difference approximation outlined in Table 3.1 will be used to transform each of the partial differential equations in Equation 3.3.

f_{i-3}	f_{i-2}	f_{i-1}	f_i	f_{i+1}	f_{i+2}	f_{i+3}		Order of Error
0	0	-1	0	1	0	0	$2hf'(x_i)$	h^2
0	0	1	-2	1	0	0	$h^2f''(x_i)$	h^2
0	-1	2	0	-2	1	0	$2h^3f'''(x_i)$	h^2
0	1	-4	6	-4	1	0	$h^4f''''(x_i)$	h^2

Table 3.1 – Central difference approximation of derivatives (USQ ENG4104, 2013)

$$\frac{\partial^2 T}{\partial x^2} \rightarrow \frac{T_{m+1,n,o} - 2T_{m,n,o} + T_{m-1,n,o}}{(\Delta x)^2}$$

$$\frac{\partial^2 T}{\partial y^2} \rightarrow \frac{T_{m,n+1,o} - 2T_{m,n,o} + T_{m,n-1,o}}{(\Delta y)^2}$$

$$\frac{\partial T}{\partial t} \rightarrow \frac{T_{m,n,o+1} - T_{m,n,o}}{\Delta t}$$

$$\alpha\left[\frac{T_{m+1,n,o} - 2T_{m,n,o} + T_{m-1,n,o}}{(\Delta x)^2} + \frac{T_{m,n+1,o} - 2T_{m,n,o} + T_{m,n-1,o}}{(\Delta y)^2}\right] + \dot{q} = \frac{T_{m,n,o+1} - T_{m,n,o}}{\Delta t}$$

Equation 3.4 – Transformation into central difference equation

Where;

$\dot{q}$, is the thermal rate of change due to heating within a finite element

α, is the thermal diffusivity of the material as outlined in Section 2.4.4)

$$\alpha = \frac{k}{c_p \rho} \; (\mathrm{m^2/s})$$

Equation 3.5 – Thermal diffusivity of a material

Figure 3.1 shows how the future point of $T_{(m,n,o)}$, $T_{(m,n,o+1)}$, can be calculated from the know value of $T_{(m,n,o)}$ and the value of T at the surrounding discrete elements. This is repeated for all values of m and n to determine the future temperature distribution of the complete system.

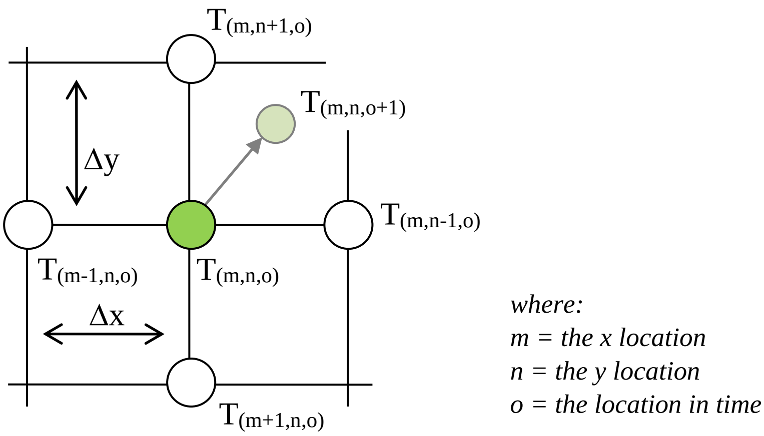


Figure 3.1 – Matrix formation for 2-D steady state temperature

Re-arranging Equation 3.4 and by ensuring that Δx = Δy, the following equation is produced:

$$\frac{\alpha}{(\Delta x)^2}\left[T_{m+1,n,o} - 2T_{m,n,o} + T_{m-1,n,o} + T_{m,n+1,o} - 2T_{m,n,o} + T_{m,n-1,o}\right] + \dot{q} = \frac{T_{m,n,o+1} - T_{m,n,o}}{\Delta t}$$

Which is equal to:

$$\frac{\alpha\Delta t}{(\Delta x)^2}\left[T_{m+1,n,o} - 2T_{m,n,o} + T_{m-1,n,o} + T_{m,n+1,o} - 2T_{m,n,o} + T_{m,n-1,o}\right] + \dot{q}\Delta t = T_{m,n,o+1} - T_{m,n,o}$$

Letting:

$$\lambda = \frac{\alpha\Delta t}{(\Delta x)^2}$$

Equation 3.6 – Simplified diffusivity constant

Which can be simplified to:

$$\lambda\left[T_{m+1,n,o} + T_{m-1,n,o} + T_{m,n+1,o} + T_{m,n-1,o} - 4T_{m,n,o}\right] + \dot{q}\Delta t = T_{m,n,o+1} - T_{m,n,o}$$

Rearranging to solve for the only unknown, $T_{(m,n,o+1)}$, yields the following equation:

$$T_{m,n,o+1} = \lambda\left[T_{m+1,n,o} + T_{m-1,n,o} + T_{m,n+1,o} + T_{m,n-1,o} - 4T_{m,n,o}\right] + T_{m,n,o} + \dot{q}\Delta t$$

Which can be simplified to:

$$T_{m,n,o+1} = \lambda T_{m+1,n,o} + \lambda T_{m-1,n,o} + \lambda T_{m,n+1,o} + \lambda T_{m,n-1,o} + (1 - 4\lambda)T_{m,n,o} + \dot{q}\Delta t$$

Equation 3.7 – Difference equation with one unknown

The central difference equation reduces to Equation 3.7 which can be used to solve the unknown future temperature of one finite element. This is effectively a summation of the previous temperature of the node, the heat received or lost to the four surrounding nodes and any internal heat generation at the node.

3.1.2) Heat generation due to current flow within the conductor

As current flows through the conductive material there is an inevitable power loss due to the voltage drop across the resistance of the material. The amount of heat generated by this power is dependent on the resistance of the conductor, the magnitude of the current and the specific heat capacity of the conducting material. The power received by the system is represented by the formula for Joule heating (Wiki, 2014a) using the magnitude of the current and the resistance of the conductor. The resistance per metre is available from manufacturer datasheets and in Table 2.3.

$$P = I^2 R \text{ (W or J/s)}$$

Equation 3.8 – Joule heating/Ohms law

The power acting on each finite element, p, can be solved by multiplying the total power loss, P, by the ratio of finite element area to the conductor cross-section. It should be noted that by using a length of 1 metre, the volume can be simplified to area.

$$p = P\frac{V_{FE}}{V_C} = P\frac{\Delta x.\Delta y}{A} \text{ (J/s)}$$

Equation 3.9 – Joule heating within one finite element

The rate of temperature change within the F.E. due to the power loss, $\dot{q}$, can be found using the specific heat of the material, c_p, which is the amount of energy required to heat a per unit mass by one degree. This rate of temperature change is required in Section 3.1.1) to cater for the additional thermal energy from internal heating (QueensU, 2014, p. 11).

Where, m is the mass of the finite element, relative to the density and volume of the F.E.

$$m = \rho . \Delta x . \Delta y . l$$

$$\dot{q} = \frac{p}{c_p . m} = \frac{p}{c_p . \rho . \Delta x . \Delta y . l} \text{ (K/s)}$$

Equation 3.10 – Rate of temperature change of F.E. from internal heating

3.2) Model conditions

3.2.1) Boundary conditions

Boundary conditions are required when using an explicit finite element approximation. The boundary conditions that will be used for this model are the temperature values of the outer finite elements. To ensure the boundary conditions have minimal impact to the system, the system will need to be big enough to ensure the cable system can heat without the boundary elements acting as a heat sink.

3.2.2) Conditions at time = 0

At the initial point of the simulation, the complete system will be set to the ambient temperature of the ground and air (if depth is less than half the system height). This will allow the simulation to analyse the temperature rise of the system from a no-load condition to steady-state.

3.2.3) Simulation time

The simulation shall continue to run until the maximum temperature within the system stabilises. This is dependant of the size of the system and it was found that the temperature change within the system was negligible after a simulation period of five days.

3.3) Fault current temperature rise

Once the load current has been used to determine the steady-state operating temperature of the cable, various over-current values shall be imposed on the system to determine the time it takes for the conductor to reach the operational temperature limit. By repeating this across a range of current values, a break curve can be generated and an industry standard IDMT curve can be fitted to the data points.

3.4) Statistical cable joint health

The values found by Mehairjan (2010, p. 73) for the Weibull distribution will be used to determine the failure rate function which will outline the probability that a joint failure will occur with respect to the age of the system.

The shape of the system, β = 4.48, and the scale parameter, η = 52.40, outline the probability distribution function, f(t), of the two parameter Weibull distribution equation:

$$f(t) = \frac{\beta r^{\beta-1}}{\eta^{\beta}} e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

Equation 3.11 – Equation for 2-parameter Weibull distribution

The reliability function is defined as follows (New Mexico Tech):

$$R(t) = e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

Equation 3.12 – Equation for reliability function

The failure rate distribution, F(t), is determined by dividing the probability distribution function by the reliability function (New Mexico Tech):

$$F(t) = \frac{f(t)}{R(t)}$$

Equation 3.13 – Equation for failure rate distribution

Solving F(t) for t = age of system (years) gives the probability of one cable joint failing.

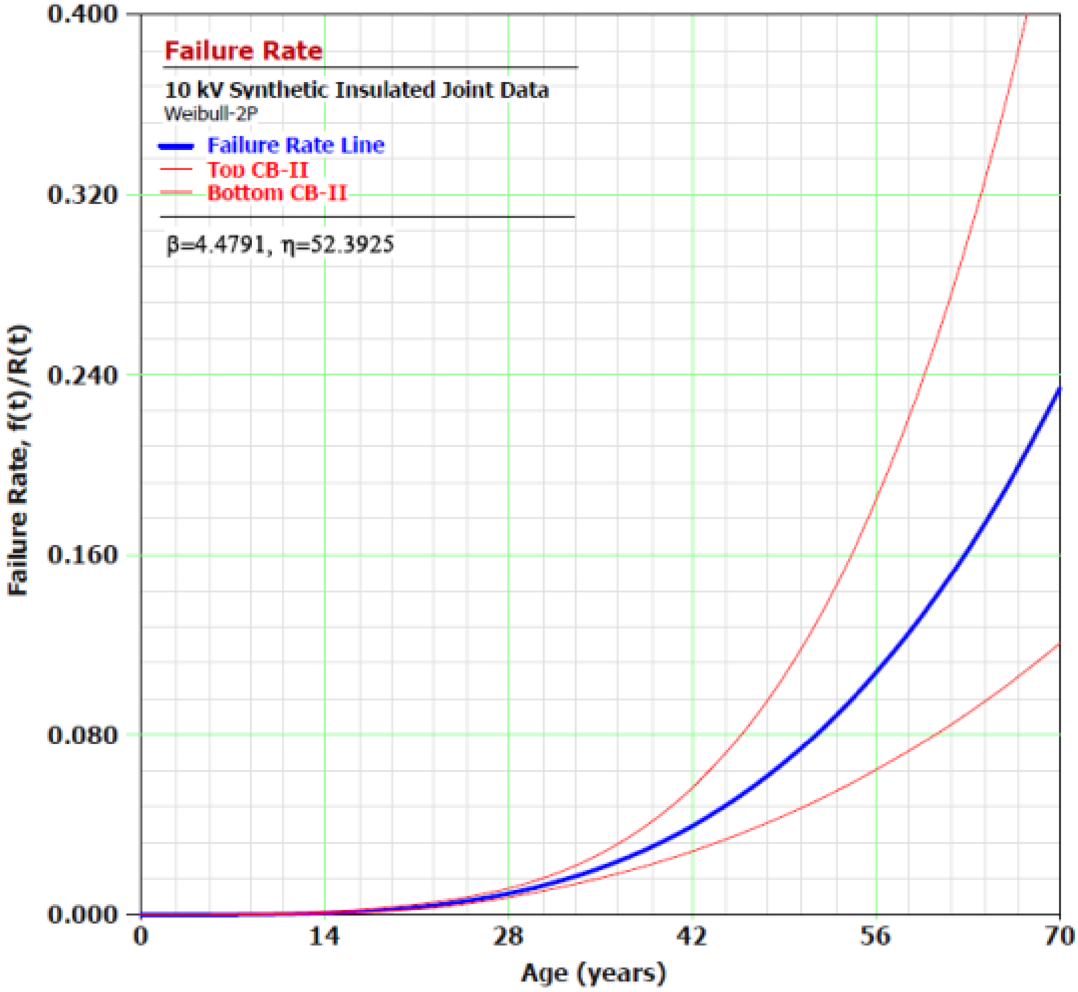


Figure 3.2 – Failure rate of a single cable joint (Mehairjan, 2010, p. 73)

The probability of one joint failing, Figure 3.2, is then multiplied by the total number of joints within the system to determine the probability any one joint will fail at the specified age of the system. This probability will then be linearly interpolated with the probability of 0 returning the impedance value of a healthy joint, 15 μΩ, and the probability of 1 returning the impedance of a poor joint, 48 μΩ (Fournier & Amyon, 2001). This impedance value can then be simulated to analyse how some of the joints within the system may behave.

Note: this is based on an assumption that if a joint has failed, it has been replaced and the replacement is the same age as the system.

4) IMPLEMENTATION INTO MATLAB

4.1) Overview

MATLAB software provides a platform to modify and manipulate data contained in matrices. This makes it perfect for manipulating and solving a matrix of finite elements. This chapter outlines how the mathematics discussed in Chapter 3 will be implemented into MATLAB and used to solve the thermal analysis of an underground cable system. Appendix C - MATLAB code structure, outlines the interaction between the MATLAB files with the complete code outlined in Appendix D - MATLAB code.

4.2) Finite elements of the system

The size of the finite element matrix depends on the user specification of the system resolution. The cross-section of the simulated cable system is configured as a square with an equal number of finite elements across the horizontal and vertical planes. Increasing the resolution of the system significantly increases the time the simulation will take to solve. This is due to the extra finite elements and a further requirement to decrease the simulation time step to ensure the system remains stable. Running higher resolution simulations may be feasible when solving a system requiring a higher level of accuracy. The following table outlines the three different resolution settings available to the user.

Resolution	System size	Matrix size	No. of F.E.	Stage 1 iterations	Runtime	Comments
Low (10 mm)	1m x 1m	101 x 101	10,201	2,160,000	20 mins*	This resolution will provide the user with a good approximation of the system within a reasonable timeframe. It is suggested to use the 'Mid' resolution to when system is confirmed. This resolution will not work for cable systems with conductor cross-section area of less than 500mm^2.
Mid (4 mm)	0.8m x 0.8m	201 x 201	40,401	14,400,000	6 hours*	This resolution will provide accurate results with a trade-off of simulation runtime. This should be used to simulate mid-large cross-sections and will return well defined images throughout the simulation.
High (2 mm)	0.6m x 0.6m	301 x 301	90,601	43,200,000	18 hours*	This resolution should be used only to simulate smaller cable systems as the cross-sectional area is smaller than the systems above. The 2mm step size improves the ability for the square based F.E. system to represent the curved shape of the cable cross-section.

Table 4.1 – Finite element resolution configuration

* Simulation time will depend on computer's performance

4.2.1) Layout matrix

A matrix representing the total number of finite elements shown in Table 4.1 is generated with each matrix entry represented by an integer that maps to the material most present at the finite element location. This can be shown by comparing the low and mid resolution of a single cable in Figure 4.1. These images are displayed to the user with a relevant colour mapped to the number of each element of the layout matrix during the configuration of the cable system. It can be seen that the materials of the low resolution setting are in the correct location and take the form of the dominant material within the finite element.

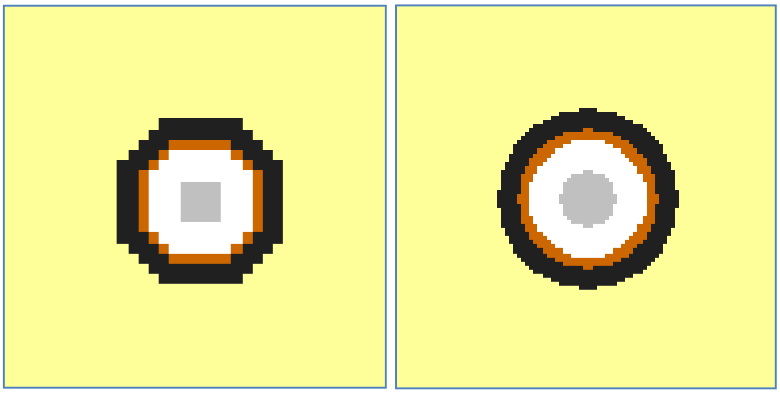


Figure 4.1 – Comparison of the layout representation of low and mid resolution

The layout matrix is created using Pythagoras' theorem to calculate the distance from each finite element to the centre of each conductor using the number of rows and columns. This distance is used to determine which material would fall within the finite element based on the user defined material thickness.

When simulating a system without cable joints, the layout matrix will be defined using the material thicknesses configured by the user. If a system contains cable joints (only applicable to conductors of cross-section of 400mm^2 or less) the thickness of the materials will be defined by the part specification of the Tyco jointing kit, MXSU (Tyco Electronics, 2009).

The joint dimensions are based on the MXSU-3341 as this joint is appropriate for cables with a cross-section of 185mm^2 to 400mm^2.

The layout matrix contains integers from 1-7 and map accordingly to Table 4.2.

Integer representation	Material representation	Colour representation
1	Conductor	Grey (aluminium) Copper (gold)
2	XLPE	White
3	Shield	Gold
4	PVC	Black
5	Bedding sand	Tan
6	Soil	Brown
7	Air	Blue

Table 4.2 – Layout matrix integer representation

Note: air is only shown if the depth of lay is less than half the height of the system, as defined above in Table 4.1. Australian Standard AS3000 (2007) specifies cables to be buried at a depth greater than 0.6 m thus this simulation would not typically be required, however, this feature has been included to simulate cables rising to be terminated above ground.

Establishing the above layout matrix simplifies the association of material properties with specific finite elements, as a MATLAB ‘if’ statement can be used to manipulate one material type. For example, heat generated within the finite elements will only occur within conductor materials represented by the integer ‘1’ in the layout matrix.

4.3) Thermal matrix computation

As the simulation progresses through time, the temperatures throughout the system will vary. As the temperatures within the system change, the material properties will also change. It is therefore required that the material properties be updated as the thermal profile of the system changes to enhance the accuracy of the simulation.

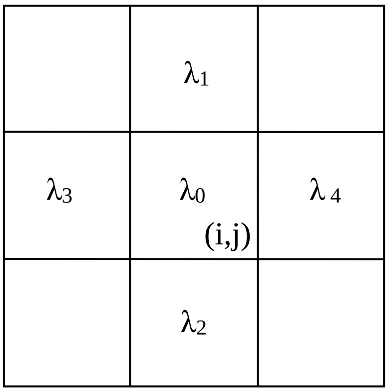
4.3.1) Material property variation

The properties of the materials, as discussed in Section 2.3) vary with respect to temperature. The individual diffusivity coefficient, λ, as outlined in Equation 3.6 is dependent on; the material's thermal diffusivity, α, the simulation time step, Δt, and the area of the finite element, $(\Delta x)^2$. As the timestep and area remain constant across the complete simulation space, the value of alpha becomes the variable of lambda for each finite element.

$$\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \quad \text{where,} \quad \alpha = \frac{k}{c_p \rho}$$

Equation 4.1 – Diffusivity constant revisited

The λ value of each finite element is calculated and maintained in the lambda matrix, L. This is dynamically updated as the temperature changes within the system to account for variation in the material properties. Due to these physical variations and the use of different materials throughout the system, neighbouring elements will have different lambda values. To overcome this, the L matrix is used to determine the average lambda value at each edge of all finite elements. These values are represented as four matrices, (Lu, Ld, Lr, Ll Figure 2.3), containing the lambda values at each boundary of every finite element. Further use of the L matrices is outlined in section 4.3.3).



Lambda matrix values of the finite element located at (i,j):

$Lu_{(i,j)} = (\lambda_0 + \lambda_1) / 2$
$Ld_{(i,j)} = (\lambda_0 + \lambda_2) / 2$
$Ll_{(i,j)} = (\lambda_0 + \lambda_3) / 2$
$Lr_{(i,j)} = (\lambda_0 + \lambda_4) / 2$

Figure 4.2 – Method to determine the Lambda values interacting with each F.E.

4.3.2) Qdot matrix

Another consideration that must not be overlooked is the variation of the conductor resistance with respect to temperature. As outlined in Equation 3.8, the power generated within the conductor is proportional to the resistance of the conductor, it is therefore important that the rate of heat generation within each conductor finite element be dynamically recalculated as the conductor temperature changes.

This matrix Qdot contains the value of $\dot{q}$ at every conductor finite element which is calculated using the steps outlined in section 3.1.2). By maintaining the data in matrix form, the influence of $\dot{q}$ can be applied across the complete system using matrix addition of Qdot during the creation of the future thermal matrix as outlined in section 4.3.3).

4.3.3) T matrix

The thermal matrix, T, contains the temperature of each finite element within the system. This makes it considerably large with up to 90,000 entries, depending on the resolution. Executing the steps outlined in Section 3.1.1) to calculate the future temperature for each of the entries would take a very long time if executed as a ‘for’ loop which would not be acceptable. To optimise the run time, future temperature values will be found using matrix mathematics. This is where MATLAB’s ability to manipulate matrices becomes a critical tool for this research project.

As discussed previously, they system has boundary conditions fixing all outside elements to a known temperature. Therefore, the internal matrix values need to be solved based on the previous temperature of the finite elements within the system. To achieve this, five new matrices are created from the existing thermal matrix and then used to solve the new thermal matrix with matrix operations. Each of these matrices are 2 x 2 smaller than the T matrix and are created by first copying T and then shedding unnecessary rows and columns from the T matrix as shown in Figure 4.3.

This matrix manipulation allows for the following single line of code to apply Equation 3.7 to all the internal values and generate the future internal matrix for the thermal profile of the cable system:

```
T_Int = Lu.*T1 + Ld.*T2 + Ll.*T3 + Lr.*T4 + (1-(Lu+Ld+Ll+Lr)).*T0 + Qmat
```

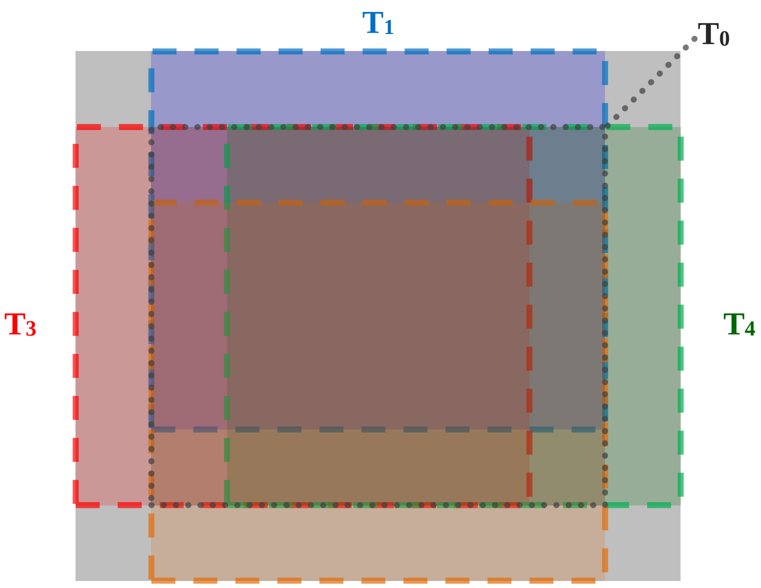


Figure 4.3 – Creation of T' matrices for simulation optimisation

4.4) Pick-up value

The pick-up value serves two purposes in this simulation; to show the users the maximum pick-up value that could be used on the cable system, and as a reference point for the first break curve value. To solve for the pick-up value, the simulation must find the current that will cause the system to heat to 90 °C at the timer limit of the protection relay. This could be solved by brute force, however, the following method has been used to reduce the time to solve the simulation.

The value cannot be solved directly from the load current temperature change due to the non-linearity of the system with respect to temperature. However, this information can be used to estimate the pick-up current. The maximum time for this simulation is restricted by the maximum counter time of the protection relay, t_{max} (10,000 seconds). The simulation is run with the estimated pick-up current and the resultant temperature rise is used to further estimate the pick-up current. Once the maximum temperature stabilises within 0.5°C of the maximum allowable temperature, 90°C, the pick-up current value is accepted and the simulation advances to the next step; determining the system break points.

4.4.1) Method for pick-up current estimation

As, ΔT is proportional to q, q is proportional to P, and P is equal to I^2 the following equation can be used as an approximation. However, as the system is non-linear due to the variation of material properties with respect to temperature, this is used as a guide only.

$$\Delta T \propto \mathrm{I}^2\mathrm{R}$$

Although the target curve is the red curve outlined in Figure 4.4, starting from the steady-state thermal profile, it can be seen that solving for the no load pick-up current value, green, will yield only a small error to the desired 'red' curve. This error will later be removed as the simulation converges on 90 °C.

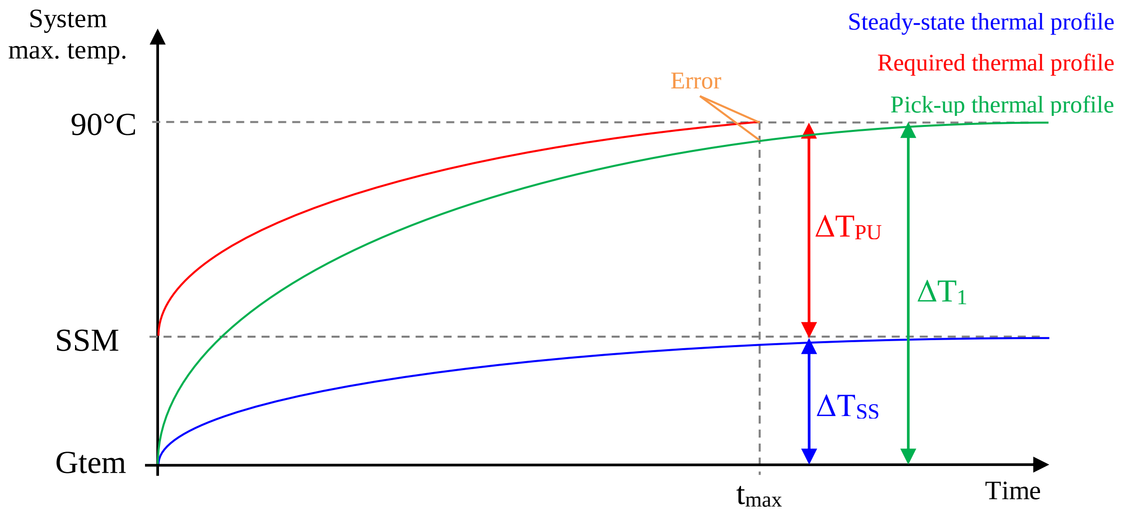


Figure 4.4 – Solving for the system's pick-up current

Using the following proportionality equations:

$$\Delta T_{SS} \propto I_L{}^2 R$$

$$\Delta T_1 \propto I_1{}^2 R$$

Where resistance, R, is assumed constant (this is not true due to material property changes):

$$\frac{\Delta T_{SS}}{I_L{}^2} \propto R$$

$$\frac{\Delta T_1}{I_1{}^2} \propto R$$

The following equation can be derived:

$$\frac{\Delta T_{SS}}{\mathrm{I_L}^2} = \frac{\Delta T_1}{\mathrm{I_1}^2}$$

Rearranging to isolate the unknown:

$$\mathrm{I_1}^2 = \mathrm{I_L}^2 \frac{\Delta T_1}{\Delta T_{SS}}$$

Solving for I_1:

$$\mathrm{I_1} = \mathrm{I_L} \sqrt{\frac{\Delta T_1}{\Delta T_{SS}}}$$

4.4.2) Pick-up current finalisation

The above method is used to determine an estimate of the pick-up current. Due to the non-linearity of the system, this value is always above the target of 90°C. This is predominantly due to the increase in resistance of the conductor and therefore a higher power output as temperature increases.

By repeating the above method, the system can converge on the actual pick-up current of the system. Again, due to the non-linearity of the system, the second attempt would overshoot so the average between the calculated value and the previous estimate is used. This is repeated until the resultant thermal curve reaches $89.5 < T < 90.5$ °C at $t = t_{max}$. The maximum attempts is locked at five to ensure the system does not get into an endless loop should the result not diverge, however, the pick-up current value is generally found in less than three attempts, as shown from the output data below.

Example progress report from the MATLAB command window:

```
At iteration 1, current used 1399A, max temp. 135.27
At iteration 2, current used 1228A, max temp. 93.17
At iteration 3, current used 1212A, max temp. 90.08
```

4.5) Solving for protection settings

4.5.1) Break curve

Once the pick-up current is known, the simulation then continues to solve for the break points of the cable system. To achieve this, fault current values at logarithmic intervals above that of the pick-up current are simulated onto the steady state thermal profile. The simulation time taken for the system to reach the maximum permissible temperature is recorded at each of the fault current values thus creating a break curve. A safety curve is then determined by considering the breaker operating time and the user defined safety margin which would normally account for any safety factors and equipment tolerances.

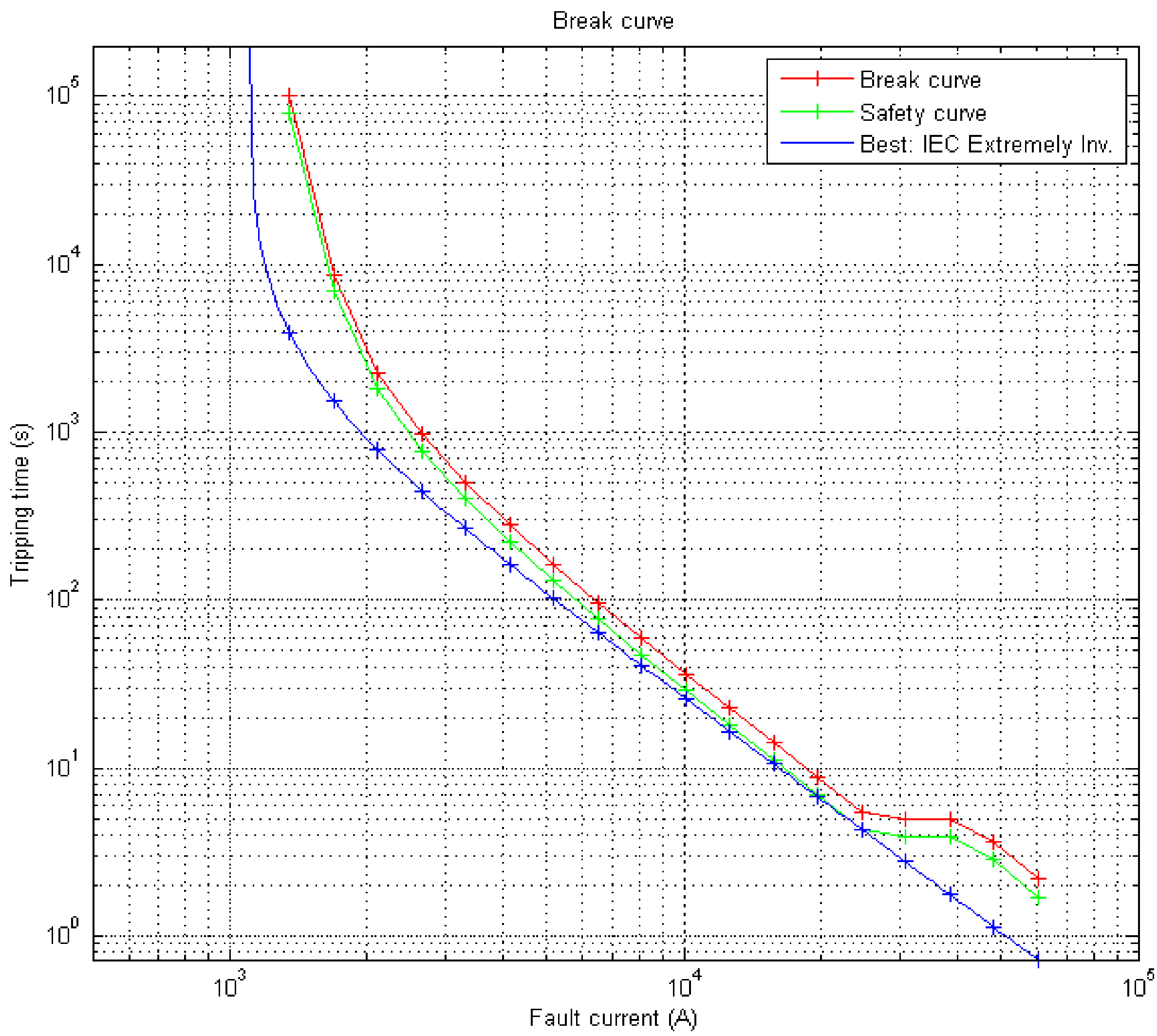


Figure 4.5 – Break curve of simulated cable system

4.5.2) Curve fitting

A brute force approach is used to fit the industry standard protection curves outlined in Section 2.2.6). Two variables are required to configure the curves; the pick-up current and the time multiplier setting (TMS). At this stage, the pick-up current is already know from the method described in section 4.4).

The system individually solves each curve for the TMS value by starting with TMS = 0, the curve is checked against the safety curve and if all points fall below this, the TMS is incremented by 0.1, which shifts the curve up slightly and, re-checked. Once the TMS value causes the curve to exceed any of the points on the safety curve, the previous TMS value is saved and the same procedure is undertaken to solve for the other curves.

4.5.3) Best fit curve

After all the curves have been fit to the safety curve, the next step is to find the curve that fits best. To achieve this, the vertical gap (time) between each point of the best fit and safety curve, is used to determine the regression. As this is a logarithmic system, linear regression cannot be used as the points at the lower end of the tripping time should carry the same weight as the tripping time at t_{max}. For this reason, a logarithmic regression is used at each point of the curve, as depicted by crosses in Figure 4.5. The regression values for each curve are summated with the lowest total regression representing the most appropriate protection settings. This curve is then displayed with the setting values to the user along with the break and safety curves for the cable system.

4.6) Assumptions, approximations and limitations

Whilst every effort was made to develop an accurate model of an underground cable system and account for all the significant factors influencing the thermal properties of the system, the following limitations should be noted as they may enhance the accuracy of the model. These may provide basis for further analytical work in this field.

4.6.1) Limitations of the 2-dimensional model

For the 2-dimensional model to operate, it is assumed that all parts along the cross section of the cable heat homogeneously. In reality, there would be a variation in temperature along the cable system as the cable passes through materials with varying thermal properties (Williams, 1999). This temperature differential along the cable would allow areas of increased temperature to not only conduct heat outwards through the insulating material and into the surrounding soil but also along the cable in a transverse direction. This would not have a significant impact on the analysis of the cable unless the material properties changed suddenly. However, this could be influential for a cable joint which has the potential for a significant thermal gradient (dT/dz, where z is the distance along the cable). This would promote thermal transfer in the z direction and ultimately a change in steady state temperature which has not been factored into the simulation model.

4.6.2) Interfacial thermal resistance

The thermal diffusivity between finite elements were determined by taking the average of the thermal diffusivity by the two neighbouring elements. This neglects the interfacial thermal resistance between the two material surfaces which acts to increase the thermal resistance due to molecular variations in the materials.

4.6.3) Boundary conditions

For the finite element analysis to work, boundary conditions are required. These conditions are required to keep the simulation referenced to the ambient conditions. The effect of the boundary conditions can be reduced by increasing the simulation space, however, this is a trade of with the simulation time.

4.6.4) Surface heating

The effects of surface heating, such as where an underground cable passes under a road, can affect the thermal conditions of the cable system. In the case of this simulation, no provision was added to accommodate the additional heating effects of surface heating

4.6.5) Joint resistance

Whilst many of the material properties within the system are dynamically updated as the system changes with temperature, this information was not available for cable joints, therefore the resistance remains fixed with respect to temperature. Information on cable joint resistance is not readily available as in practice, any resistance measurements on cable joints would require destructive intervention making the joint unserviceable.

4.6.6) Method for earthing the cable screen

The voltage induced on the shield of the cable by the main conductor has the potential to generate currents within the shield. These currents cause additional heating within the cable system and can result in a reduction of capacity. For this reason, many cable installations only terminate the shield of the cable at the supply end of the cable system. If this is done, the screen is still bonded to earth and any insulation breakdowns will be detected by the protection device, however, there is a risk of voltage potential developing between the cable shield and earth reference at the downstream plant. This simulation assumes no current flowing within the shield of the conductor and therefore, best models single earth bonding of a cable system.

4.6.7) Skin and proximity effect

The skin effect has been omitted from this simulation due to the relatively small, and round conductors. The effect of this phenomenon generally results in an uneven current distribution within the conductor and an increase in the joule heating due to the displaced current flow. It starts to become significant for conductors of 1600 to 2000A and is very important above 4000A where it can generate up to 10% additional heating within the conductor (Schneider Electric, 2002).

4.6.8) Heating within the insulation

As the voltage within a power cable is charging and discharging the electric field within the insulation material 50-60 times per second, this can lead to heating within the insulating material. For the purpose of this research project, this effect has been neglected.

4.6.9) Free convection

Free convection, as described by Farouke, (1981, p. 7) is caused by changes in density with respect to temperature. However, Farouke states that in soils, the convection through air or water is negligible due to the very small nature of the pores. For the purpose of this simulation, only the thermal conductive properties have been considered for thermal transfer within the system.

4.7) Validation of model

The simulation model must be assessed to determine if the results from the simulation will be useful. Whilst sophisticated and expensive cable analysis software was not accessible throughout the duration of this project, design guidelines from manufacturer's data and Australian standards provided the basis for the assessment. This information was compared to the break points generated by the simulation to determine the accuracy of the model. It would have been an added benefit to compare the steady state temperature values with real-world test results, however, no results could be found to make a valid comparison.

4.7.1) GEMSCAB

Data from the Gemscab datasheet was used to determine the current rating of the cable system as outlined in Table 4.3. These values were also implemented into the simulation model so the results could be compared for a direct buried 11kV cable with 630mm^2 cross-section. The rating factors are used to determine a more accurate cable ampacity given the environmental conditions of the cable system.

Variable	Value used	Rating factor
Cable configuration	Single trefoil	1
Conductor cross-section	630 mm^2	1
Nominal rating	553 A	1
Depth of lay	600 mm	1
Soil thermal resistance	0.5 W.m^{-1}.K^{-1}	0.89
Soil temperature	25 °C	1.04
Current rating	**512 A**	**0.926**

Table 4.3 – Gemscab current rating, data: Gemscab (2014)

By using a similar load current in the simulation model, the results from the break curve can be analysed against the short circuit rating of the cable system. These values are shown in Figure 4.6 and are derived from the following equation for short circuit rating as defined by Gemscab (2014, p. 18).

$$I_{sh} = \frac{KA}{\sqrt{t}}$$

Where, K, the thermal constant for the Gemscab 630mm^2 aluminium conductor, is equal to 59.0. From this, the short circuit current value, I_{sh}, can be equated from the conductor cross-sectional area, A, and trip time, t. This is an adiabatic approximation and not effects of thermal transfer within the system are considered.

4.7.2) Australian Standard 3008.1.1-2009

The Australian Standard, AS3008, sets out a method for cable selection and determining sustained current-carrying capacities for cable installations in Australia (AS 3008, 2009). This method is specifically for cable systems operating at voltages below 1kV, however, for the purpose of cable ampacity only, this will provide a valid benchmark to compare the simulation model.

Variable	Value used	Rating factor
Cable configuration	Single trefoil	1
Conductor cross-section	630 mm^2	1
Nominal rating	688 A	1
Depth of lay	600 mm	0.97
Soil thermal resistance	0.5 W.m^{-1}.K^{-1}	0.81
Soil temperature	25 °C	1
Current rating	**540 A**	**0.785**

Table 4.4 – Australian Standard current rating, data: AS3008 (2009)

AS3008 outlines a method for determining trip times with the addition of the safety period where the cable system can operate up to 250 ºC for less than 5 seconds. For a fault duration of more than 5 seconds, the maximum operating temperature of XLPE insulation is 90 ºC.

The values for K for faults lasting more or less than five seconds can be obtained from AS3008 - table 52.

$$K = \begin{matrix} 111 \\ 62.4 \end{matrix} \Big\} \begin{matrix} t < 5 \\ t \leq 5 \end{matrix}$$

Using these values, the following equation can be used to determine the recommended trip times of the cable system. These values are shown in Figure 4.6.

$$I^2 t = K^2 S^2$$

4.7.3) Simulation results

By simulating a cable system identical to that discussed above, the results from the simulation can be validated against the methods from the Gemscab datasheet and AS3008. The variables used in the simulation are as follows. It should be noted that 520 A was used as the load current which is between the two values determined above.

Variable	Value used
Cable configuration	Single trefoil
Conductor cross-section	630 mm^2
Load current	520 A
Depth of lay	600 mm
Soil thermal resistance	0.5 $W.m^{-1}.K^{-1}$
Soil temperature	25 °C

Table 4.5 – Values used for simulation verification

The two methods outlined above use an adiabatic model ignoring any thermal transfer within the system. This approximation is acceptable for determining thermal behaviour over short fault periods, however, as fault time increases, heat transfer from the conductor will become more apparent. Figure 4.6 shows that initially the simulation results and the trip times determined from the Gemscab and AS3008 methods are very similar. It is important to note that the AS3008 method, and similarly the simulation model, consider a 5 second period where the cable system can tolerate a maximum temperature of 250 °C. The similarity between the Gemscab and AS3008 methods for t > 5 seconds outlines that these methods offer an accurate point of comparison for the simulation results. The simulated break curve tracks very closely to that of the AS3008, especially at the higher fault levels where t < 5 seconds. The effect of thermal transfer on the protection time can be seen as time increases and the simulated curves diverge from the adiabatic curves. This provides a more realistic representation of how the system would behave at low fault levels and proves the system is comparable to industry standard approaches for determining trip times at high fault levels.

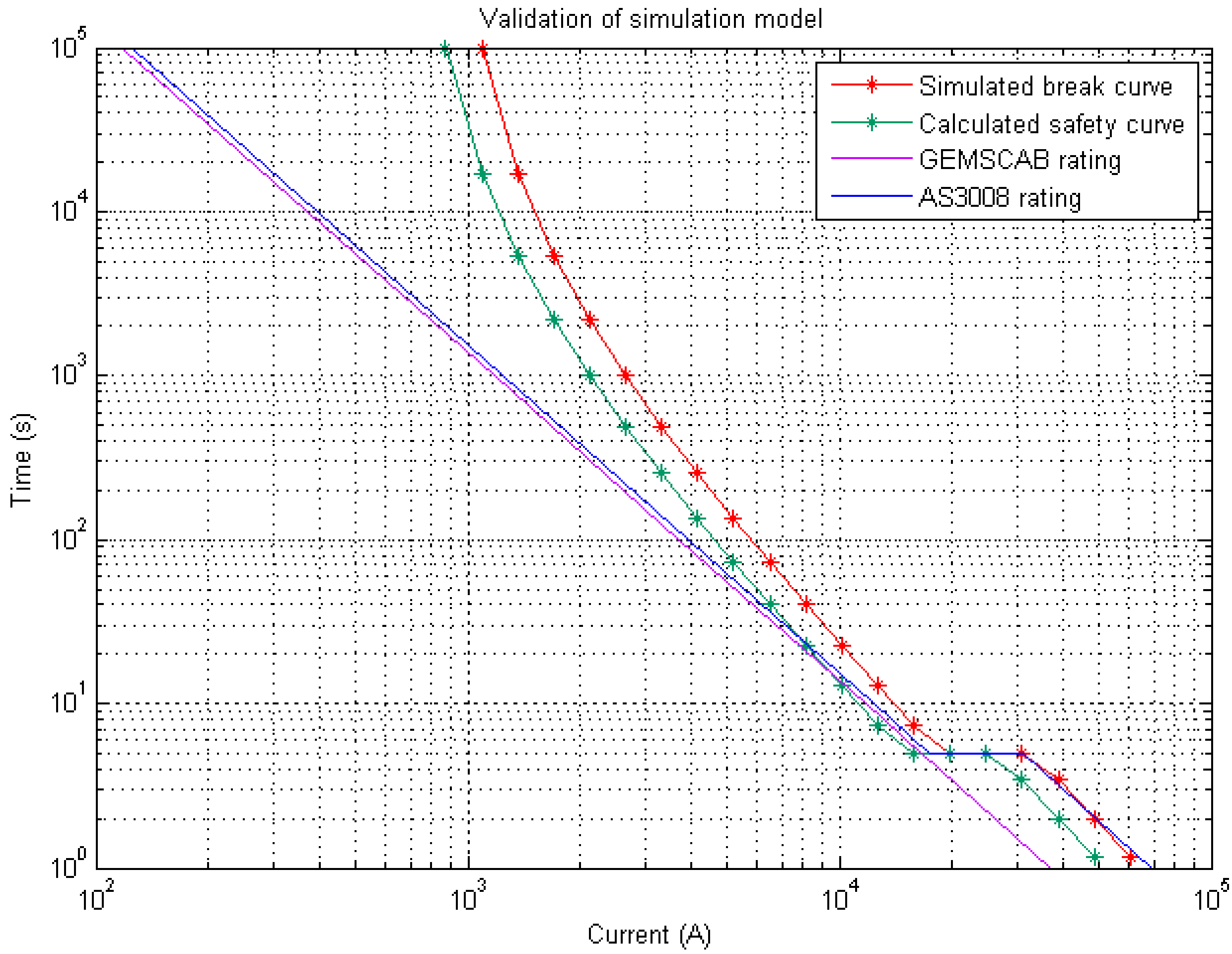


Figure 4.6 – Simulation validation using protection curves

5) CASE STUDIES AND PRACTICAL USE

5.1) Chapter overview

The following chapter covers a variety of simulated cable installations. In each case study, one key variable (Table 5.1) of the system was changed to understand the effects this variable would have on the required protection settings of a cable system.

Case study	System property variation
1	Comparison of single trefoil and parallel run trefoil with the same load current.
2	The use of a trefoil cable compared to three single cables.
3	Cable system with and without bedding sand.
4	Pre-fault load current on the cable system.
5	Variation in the ambient temperature of the soil.
6	Core conductor material - copper and aluminium.
7	How deep the cable has been buried.
8	Soil saturation level.
9	Cable joint health.

Table 5.1 – Case study overview

5.2) Case study 1 - Parallel run trefoil

One the motivating factors for conducting this research project was to analyse the maintenance options of parallel run trefoil installations. The assessment was based on taking one of the cables out of service and restoring the downstream supply via the single healthy cable. It is important to note that this will affect the voltage drop and rated current of the system, however, if these effects on the system were tolerable, modified protection settings would need to be considered in order to provide adequate protection to the reduced system. This case study will analyse if such a measure could be used to restore supply to critical downstream equipment during fault rectification and maintenance of the complementary cable. The following system parameters were used for case study 1 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B
Cable configuration	Parallel trefoil	Parallel trefoil
Cables in service	**2**	**1**
Depth of lay	600 mm	600 mm
Bedding sand around cables	50 mm	50 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)
Separation between cables	20 mm	20 mm
Conductor material	Aluminium	Aluminium
Conductor cross-section	400 mm^2	400 mm^2
XLPE thickness	12 mm	12 mm
Shield thickness	3 mm	3 mm
PVC thickness	4 mm	4 mm
Soil temperature	15 °C	15 °C
Load current	630 A	630 A

Table 5.2 – Case study 1 variables

5.2.1) Thermal results

The following plots were generated with a fixed maximum axis of 50 °C.

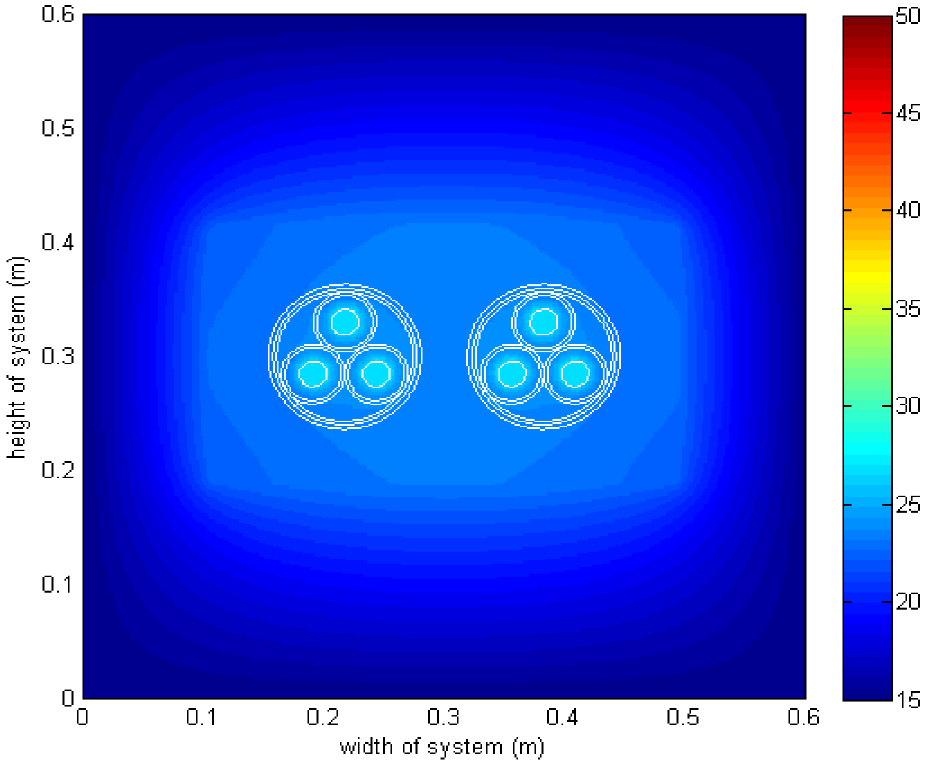


Figure 5.1 – Steady state thermal profile (all cables in-service)

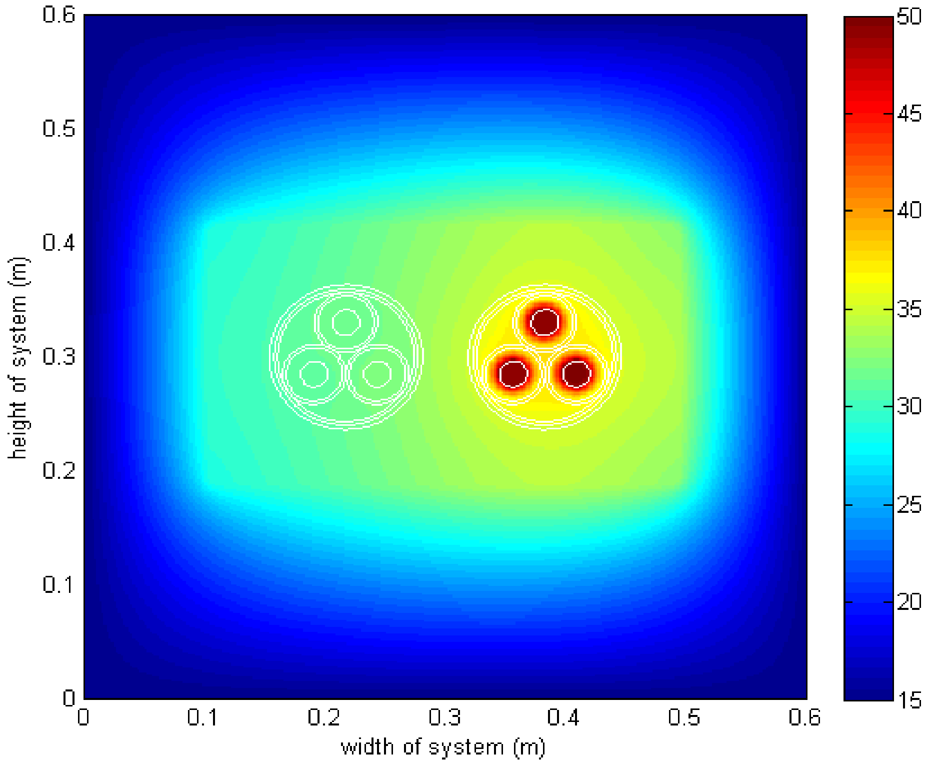


Figure 5.2 – Steady state thermal profile (single trefoil in-service)

5.2.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

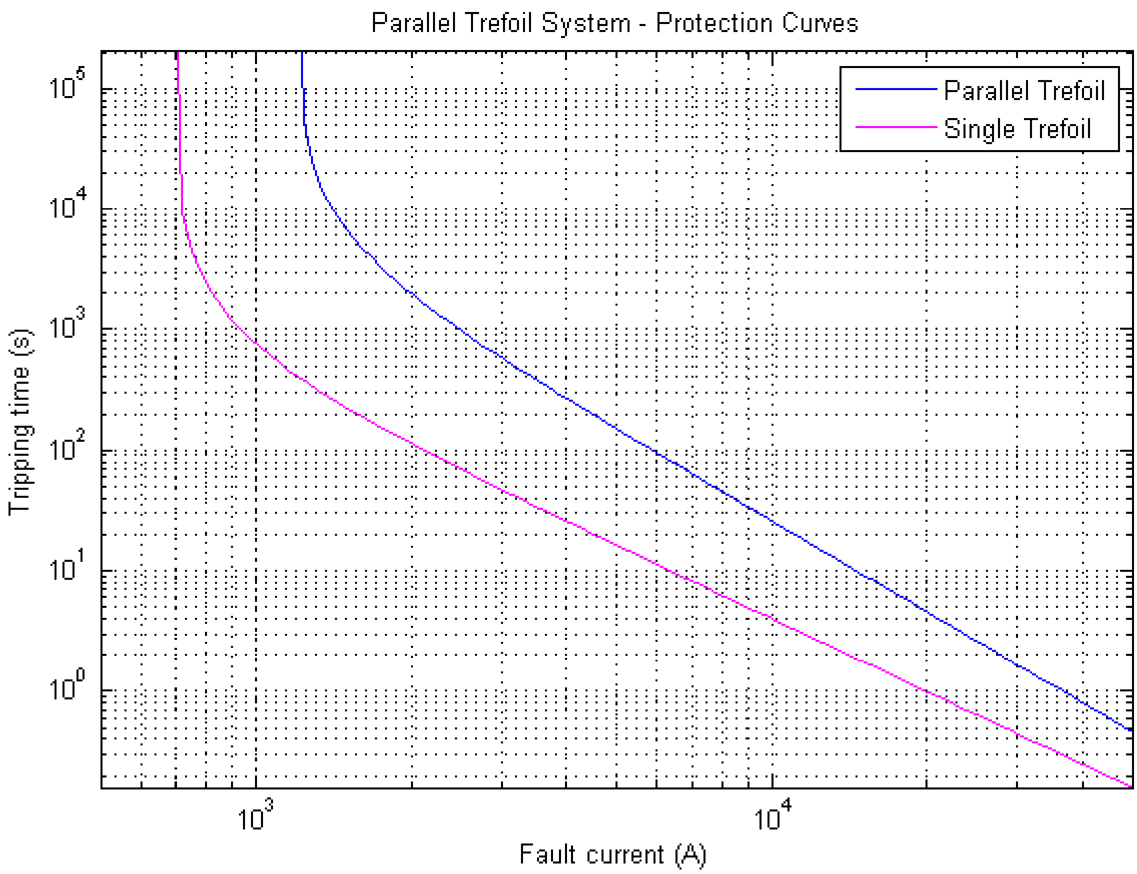


Figure 5.3 – Case study 1 IDMT protection curves

5.2.3) Discussion

Results	**Simulation A** Parallel trefoil	**Simulation B** Single trefoil
Steady state, max T	26.6 °C	48.1 °C
Steady state, ΔT	11.6 °C	33.1 °C
Maximum pick-up current	1489 A	862 A
IDMT curve	IEC Ultra	IEC Extremely
Pick-up setting	1191 A	689 A
Time multiplier setting	16.7	8.5

Table 5.3 – Case study 1 results

Table 5.3 outlines the key differences between the operating limits of parallel versus single trefoil configuration. The temperature rise of the single cable is 2.85 times that of the parallel run cable. This would increase the fatigue of the cable and reduce the expected life, however, it is still within operating limits so restoring the system as a single cable run is feasible under the results of this simulation. Obviously, the current capacity of the single cable is about half that of the single cable and looking at Figure 5.3, there is a significant shift in the required protection curve. If a single cable is to be put into service in this configuration, care must be taken to ensure the protection settings will provide adequate protection to the cable in-service.

5.3) Case study 2 - Trefoil versus three single cables

This case study investigates the variation in capacity and required protection settings when using three single phase cables, compared to a trefoil cable. The following system parameters were used for case study 2 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B
Cable configuration	**Single trefoil**	**3 single cables**
Depth of lay	600 mm	600 mm
Bedding sand around cables	0 mm	0 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)
Separation between cables	NA	0 mm
Conductor material	Copper	Copper
Conductor cross-section	1000 mm^2	1000 mm^2
XLPE thickness	30 mm	30 mm
Shield thickness	5 mm	5 mm
PVC thickness	10 mm	10 mm
Soil temperature	15 °C	15 °C
Load current	1000 A	1000 A

Table 5.4 – Case study 2 variables

5.3.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system’s maximum steady state temperature.

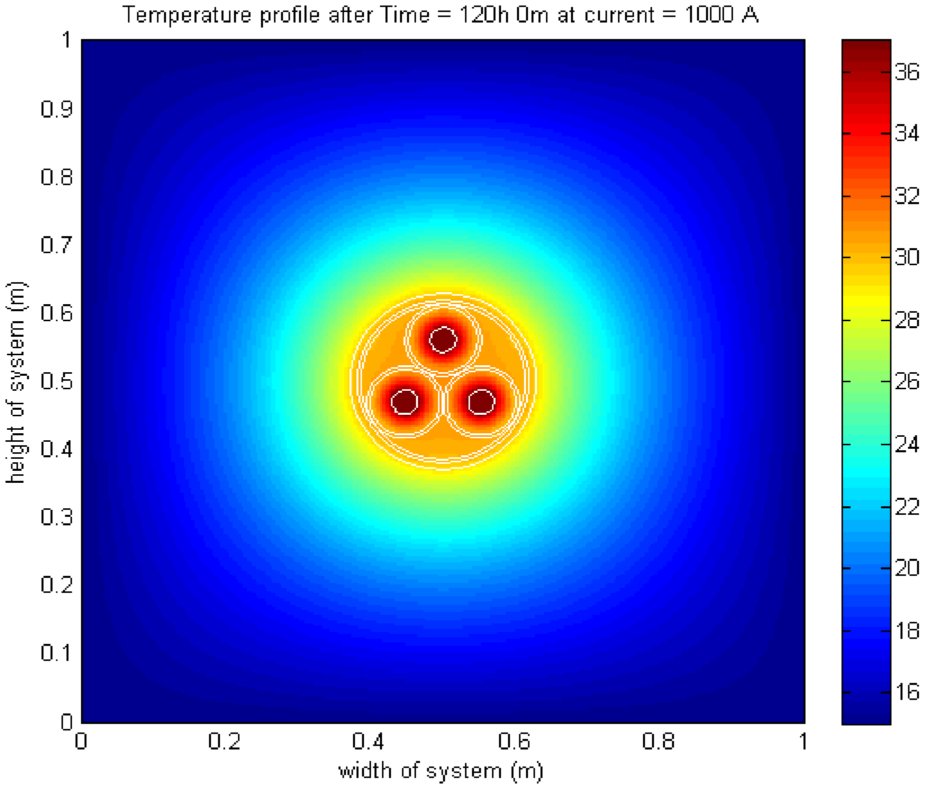


Figure 5.4 – Steady state thermal profile (trefoil)

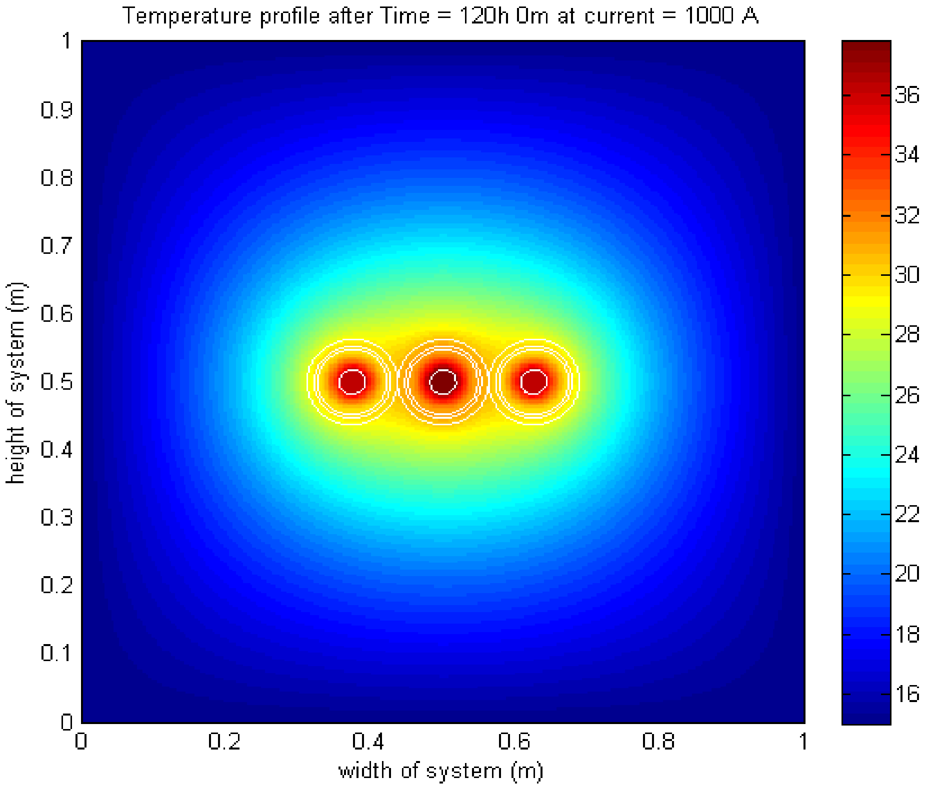


Figure 5.5 – Steady state thermal profile (three single cables)

5.3.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

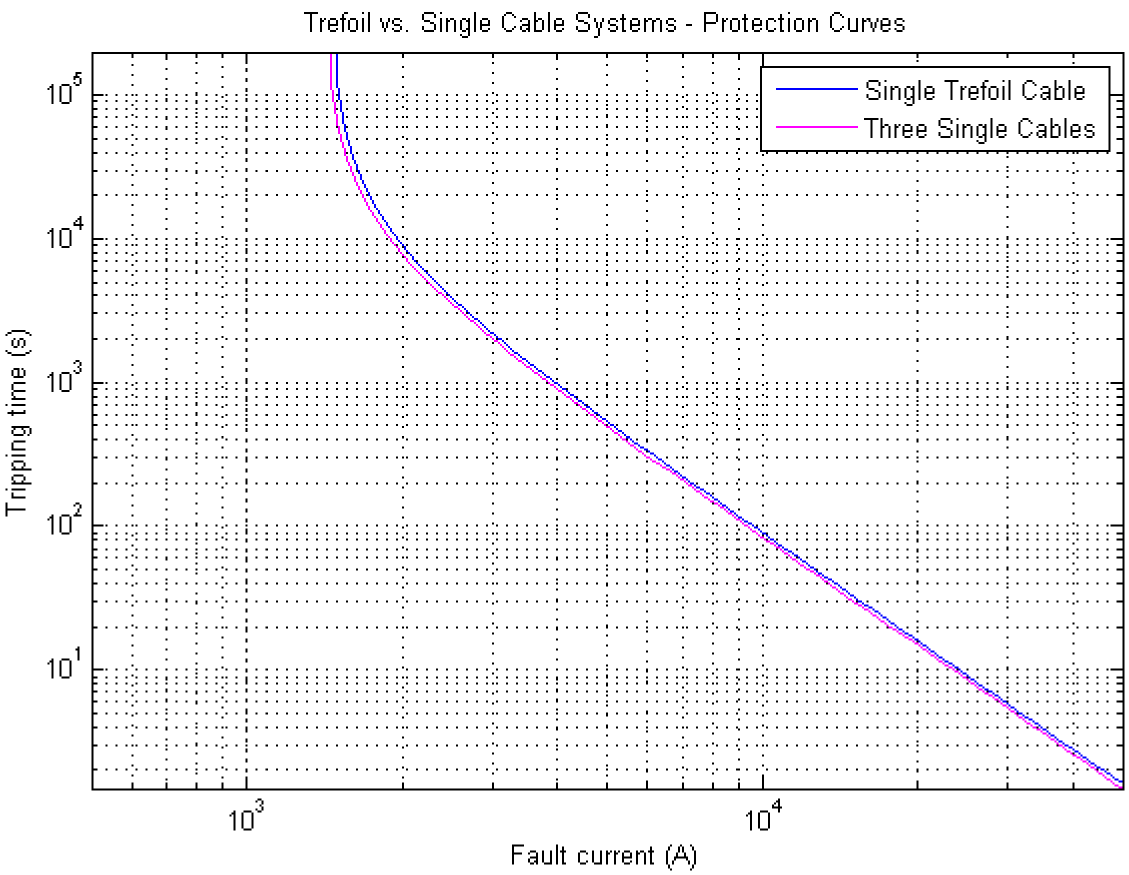


Figure 5.6 – Case study 2 IDMT protection curves

5.3.3) Discussion

Results	**Simulation A** Single trefoil	**Simulation B** 3 single cables
Steady state, max T	37.0 °C	37.9 °C
Steady state, ΔT	22.0 °C	22.9 °C
Maximum pick-up current	1803 A	1755 A
IDMT curve	IEC Ultra	IEC Ultra
Pick-up setting	1442 A	1404 A
Time multiplier setting	36.3	35.9

Table 5.5 – Case study 2 results

The thermal profile of the three single cables shows that the middle conductor sits in the centre of a symmetrical thermal system, Figure 5.5. For this reason, the centre cable endures a higher temperature than the outside cables and also a higher temperature than the trefoil system as in the trefoil system, each of the phases have an equal opportunity for heat dissipation to the surrounding environment.

5.4) Case study 3 - Using bedding sand

This case study investigates the effects using bedding sand can have on the capacity of a cable system. The following system parameters were used for case study 3 where the variation between the simulation models has been outlined in red

Variable	Simulation A	Simulation B
Cable configuration	Single phase	Single phase
Depth of lay	600 mm	600 mm
Bedding sand around cables	**150 mm**	**0 mm**
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)
Conductor material	Copper	Copper
Conductor cross-section	2000 mm^2	2000 mm^2
XLPE thickness	40 mm	40 mm
Shield thickness	2 mm	2 mm
PVC thickness	15 mm	15 mm
Soil temperature	15 °C	15 °C
Load current	2000 A	2000 A

Table 5.6 – Case study 3 variables

5.4.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system’s maximum steady state temperature.

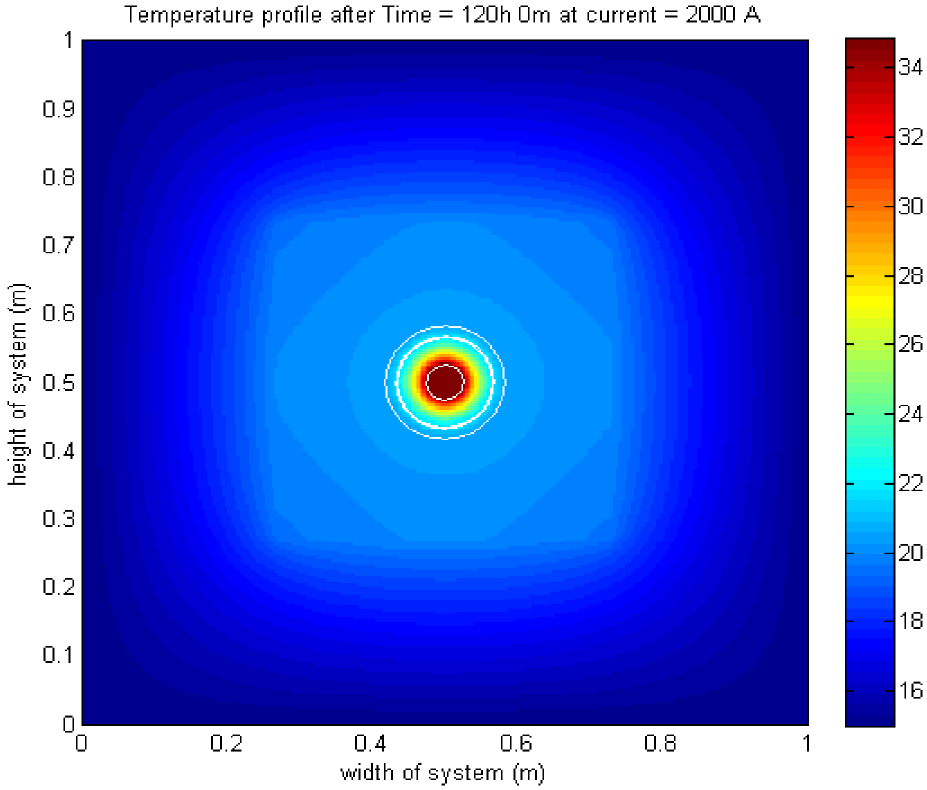


Figure 5.7 – Steady state thermal profile (with bedding sand)

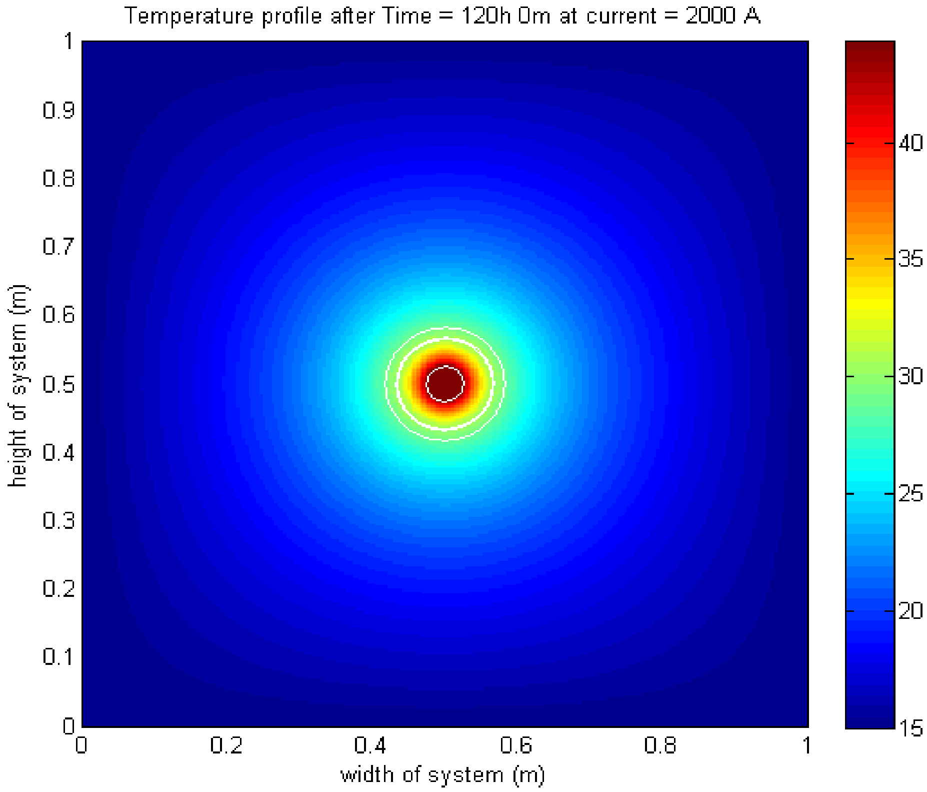


Figure 5.8 – Steady state thermal profile (without bedding sand)

5.4.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

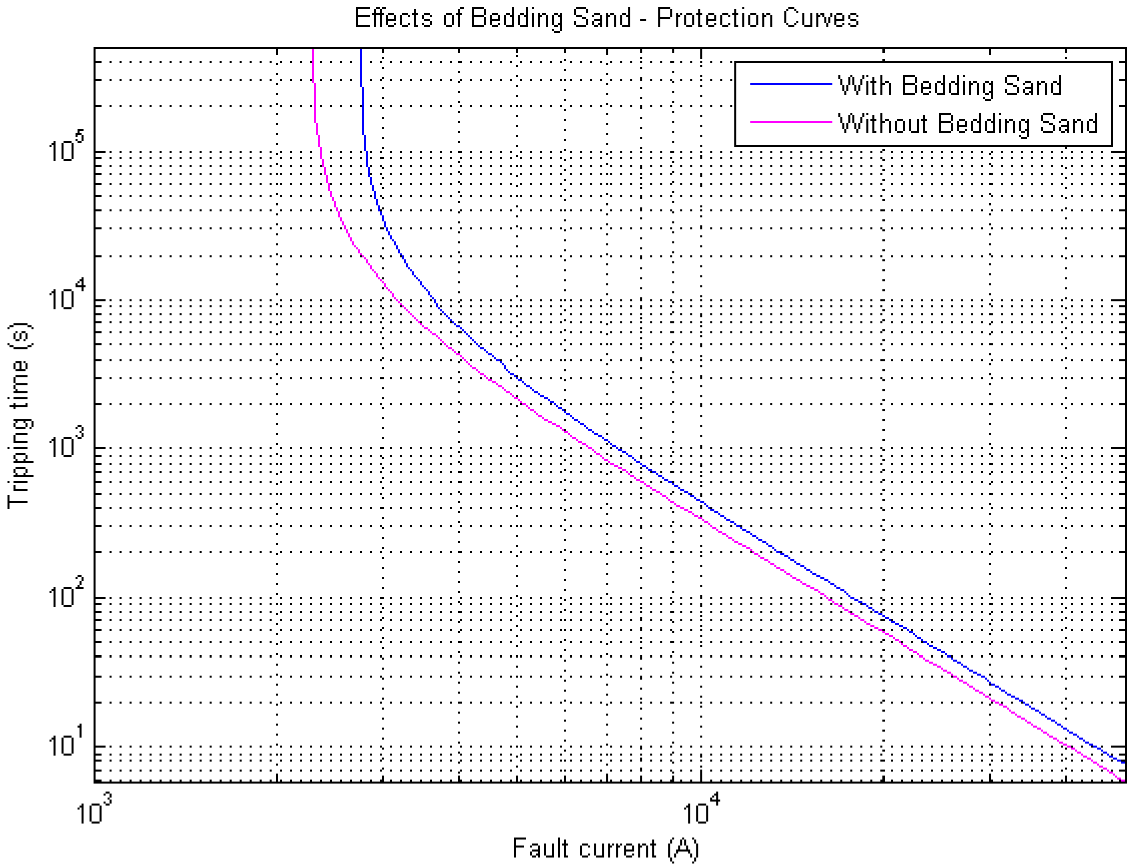


Figure 5.9 – Case study 3 IDMT protection curves

5.4.3) Discussion

Results	**Simulation A** With bedding	**Simulation B** Without bedding
Steady state, max T	34.8 °C	44.4 °C
Steady state, ΔT	19.8 °C	29.4 °C
Maximum pick-up current	3360 A	2799 A
IDMT curve	IEC Ultra	IEC Ultra
Pick-up setting	2688 A	2239 A
Time multiplier setting	35.5	43.7

Table 5.7 – Case study 3 results

Table 5.7 outlines the variation in operating limits of a cable system when bedding sand is used. The thermal properties of the bedding sand promote heat flow away from the cable reducing the operating temperature and thus increasing the capacity of the cable system. Figure 5.7 shows a clear increase in temperature where the bedding sand exists to that of Figure 5.8 where there is no bedding sand. This suggests that the sand is absorbing and distributing more of the heat generated within the conductor.

5.5) Case study 4 - Pre-fault load current

This case study endeavours to determine if there is any difference between the required protection settings of an underground cable depending on the current that was flowing through the conductor prior to a fault occurring. The following system parameters were used for case study 4 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B	Simulation C
Cable configuration	Single trefoil	Single trefoil	Single trefoil
Depth of lay	600 mm	600 mm	600 mm
Bedding sand around cables	150 mm	150 mm	150 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)	0.8 W/(m.K)
Conductor material	Aluminium	Aluminium	Aluminium
Conductor cross-section	800 mm^2	800 mm^2	800 mm^2
XLPE thickness	15 mm	15 mm	15 mm
Shield thickness	3 mm	3 mm	3 mm
PVC thickness	5 mm	5 mm	5 mm
Soil temperature	15 °C	15 °C	15 °C
Load current	**50 A**	**500 A**	**1000 A**

Table 5.8 – Case study 4 variables

5.5.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system’s maximum steady state temperature.

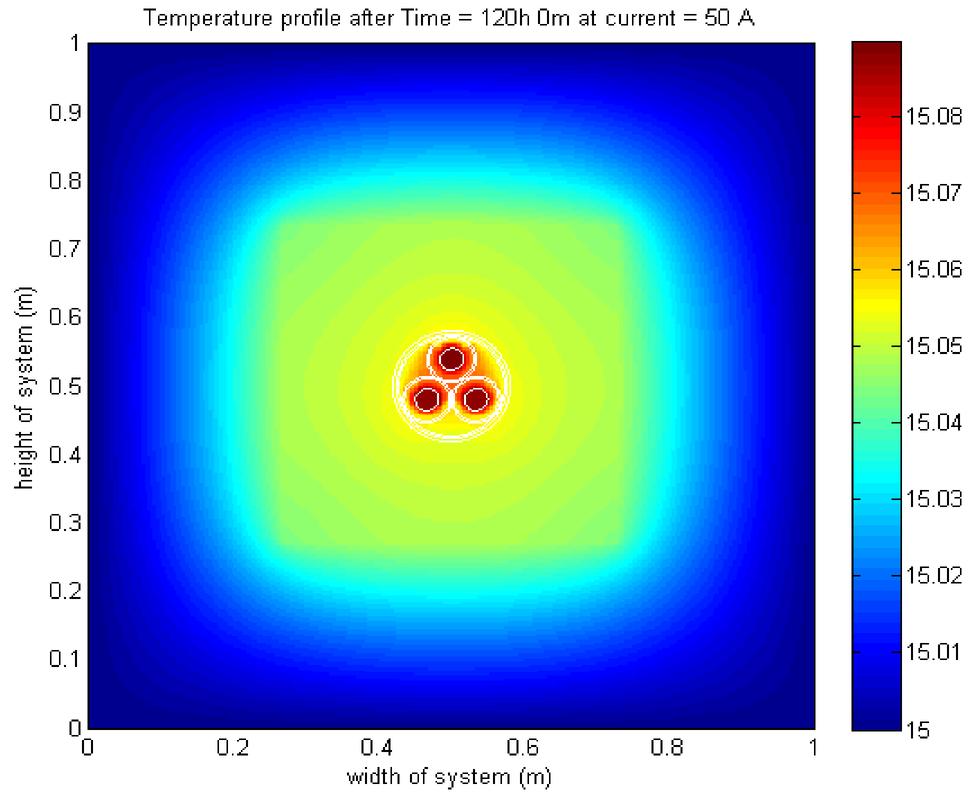


Figure 5.10 – Steady state thermal profile (50A load)

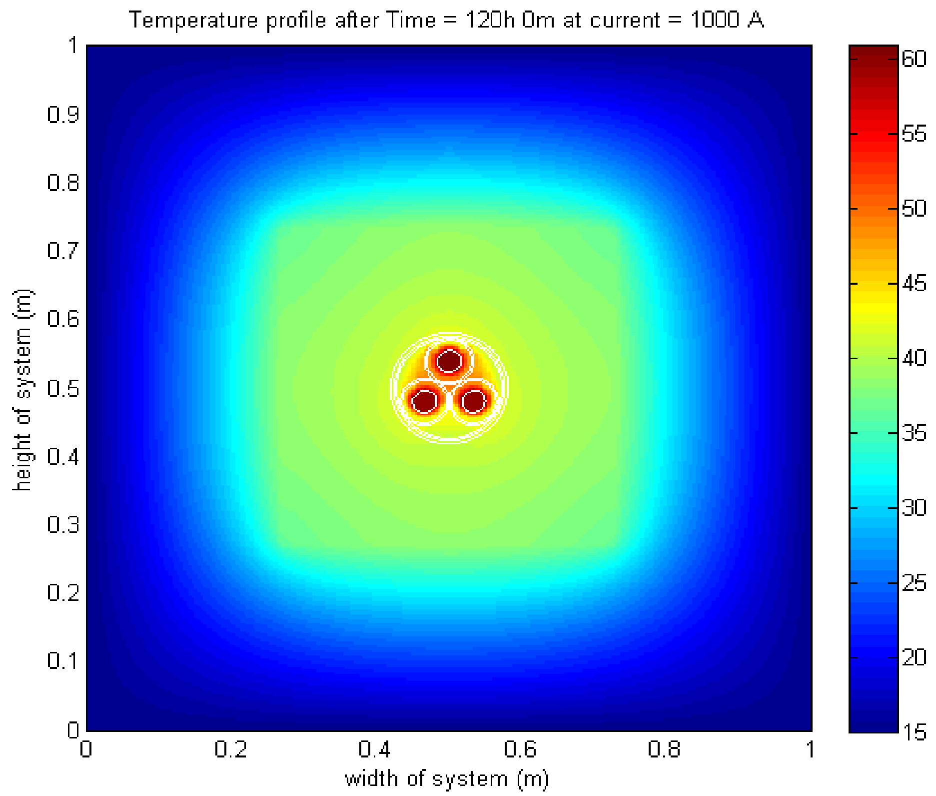


Figure 5.11 – Steady state thermal profile (1000 A load)

5.5.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

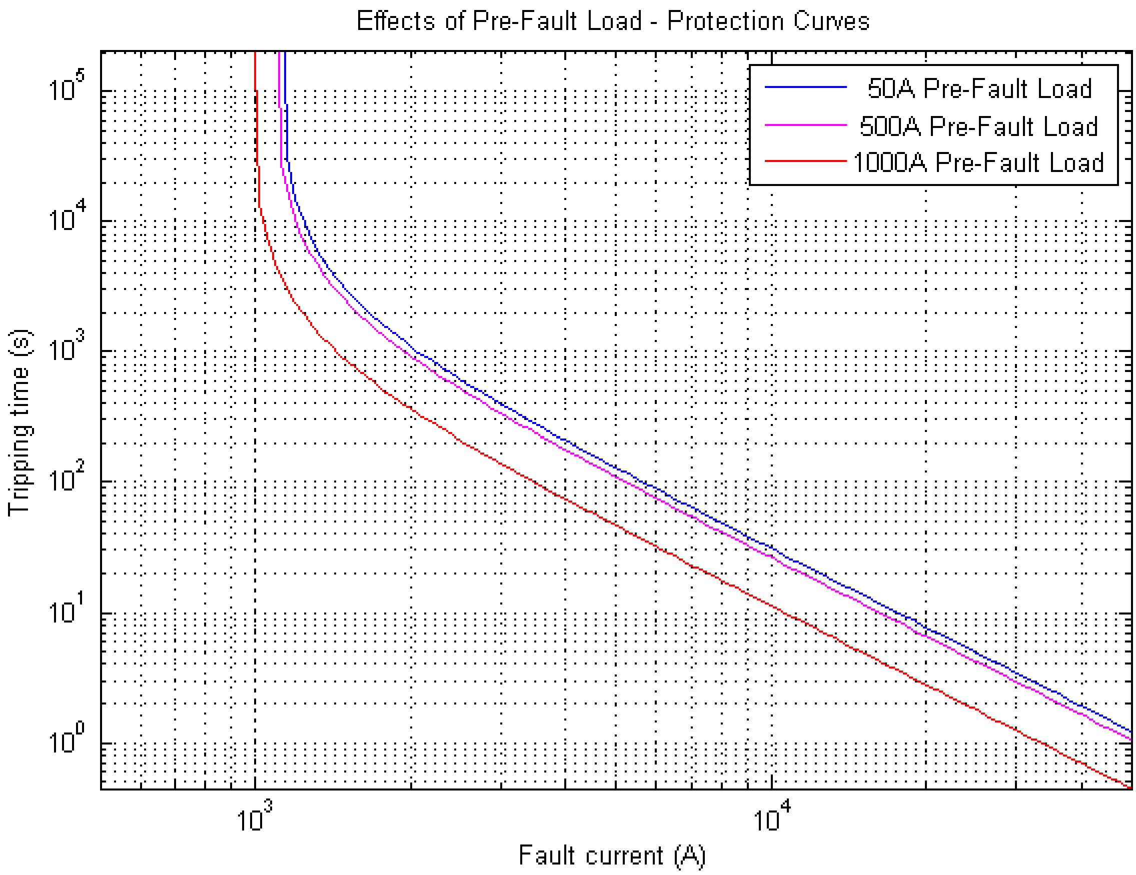


Figure 5.12 – Case study 4 IDMT protection curves

5.5.3) Discussion

Results	**Simulation A** Pre-fault 50A	**Simulation B** Pre-fault 500A	**Simulation C** Pre-fault 1000A
Steady state, max T	15.1 °C	24.4 °C	61.0 °C
Steady state, ΔT	0.1 °C	9.4 °C	46.0 °C
Maximum pick-up current	1387 A	1335 A	1227 A
IDMT curve	IEC Extremely	IEC Extremely	IEC Extremely
Pick-up setting	1109 A	1084 A	981 A
Time multiplier setting	25.2	22.5	11.7

Table 5.9 – Case study 4 results

It is clear from Figure 5.12 that the pre-fault load can significantly change the required protection settings. As the cable's maximum operating temperature is fixed, the pre-fault load current affects the steady state temperature and hence the thermal buffer should a fault occur on the system. The Gemscab (2014) datasheet outlines this cable configuration as having a rated load of 662A. By overloading this cable with 1000A, the cable has heated to 60°C which takes the cable significantly closer to the maximum specified temperature and therefore requires more sensitive protection compared to the other simulated load currents which are within manufacturer specified limits.

5.6) Case study 5 - Ground Temperature

The purpose of this case study is to determine how the temperature of the soil effects the rating and therefore the protection settings required for a cable system. For this simulation, only the ground temperature has been changed, however, in reality the soil properties would also change with respect to temperature, especially if the ground was frozen (Farouke, 1981, p. 102). The following system parameters were used for case study 5 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B
Cable configuration	Single trefoil	Single trefoil
Depth of lay	600 mm	600 mm
Bedding sand around cables	150 mm	150 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)
Conductor material	Aluminium	Aluminium
Conductor cross-section	800 mm^2	800 mm^2
XLPE thickness	15 mm	15 mm
Shield thickness	3 mm	3 mm
PVC thickness	5 mm	5 mm
Soil temperature	**-15 °C**	**15 °C**
Load current	500 A	500 A

Table 5.10 – Case study 5 variables

5.6.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system's maximum steady state temperature.

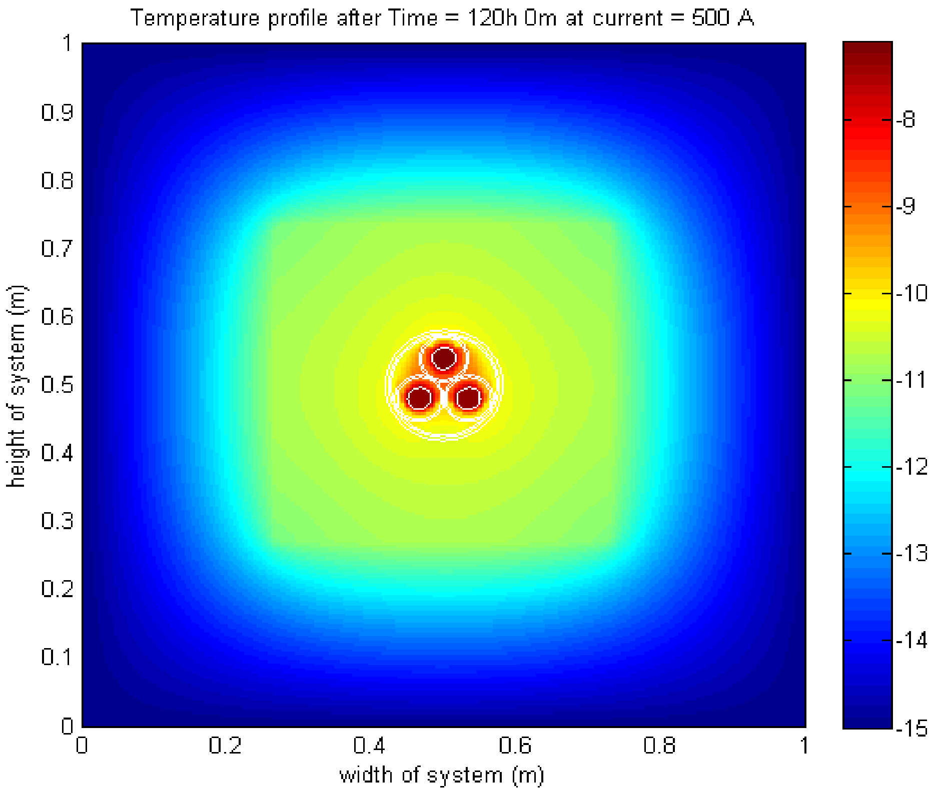


Figure 5.13 – Steady state thermal profile (ground at -15°C)

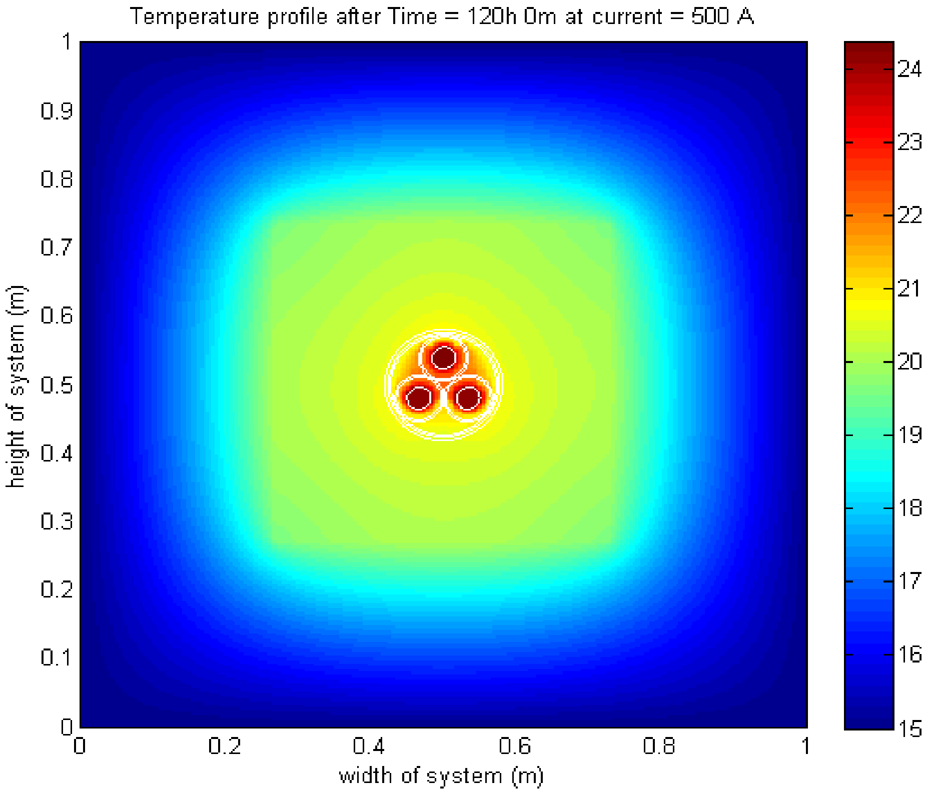


Figure 5.14 – Steady state thermal profile (ground at 15°C)

5.6.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

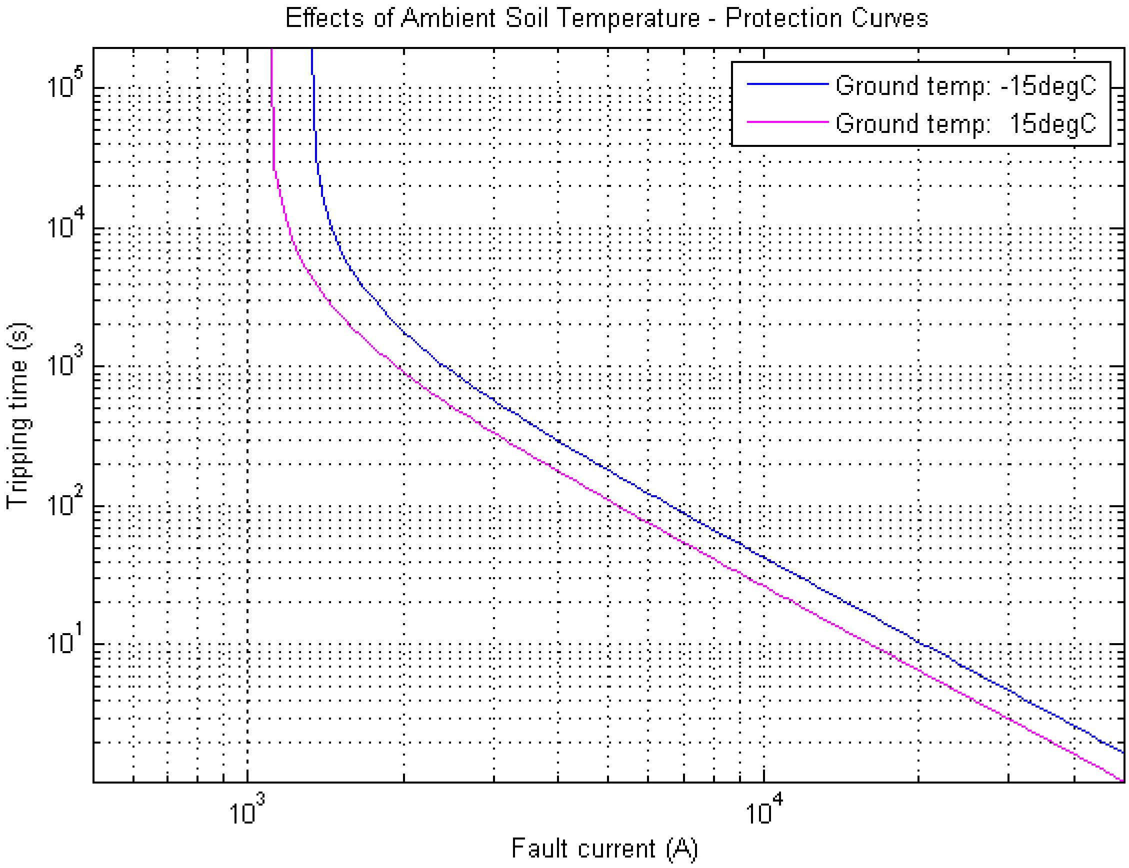


Figure 5.15 – Case study 5 IDMT protection curves

5.6.3) Discussion

Results	**Simulation A** Soil at -15 °C	**Simulation B** Soil at 15 °C
Steady state, max T	-7.1 °C	24.4 °C
Steady state, ΔT	7.9 °C	9.4 °C
Maximum pick-up current	1628 A	1355 A
IDMT curve	IEC Extremely	IEC Extremely
Pick-up setting	1302 A	1084 A
Time multiplier setting	25.0	22.5

Table 5.11 – Case study 5 results

It comes as no surprise to see the cable system operating at the lower ambient temperature requires less sensitive protection settings as there is an increased thermal buffer between the operating temperature and the maximum operating temperature of the system. Something to note here is the difference in the temperature rise between the systems. The system operating at -15 °C ambient has a lower ΔT. This is likely to be due to the change in material properties with respect to temperature such as an increase in conductor resistivity at higher temperatures.

5.7) Case study 6 - Conductor material

One of the decisions power system engineer's must make when defining the specifications for an underground cable system is the conductor material. While copper conductors offer an increased current rating, the additional material cost for copper metal and transport cost due to the additional weight often makes aluminium conductors more appropriate for cable installations. The purpose of this case study is to determine how the protection settings of a cable system would change depending on whether aluminium or copper was used as the main conductor. The following system parameters were used for case study 6 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B
Cable configuration	Single trefoil	Single trefoil
Depth of lay	600 mm	600 mm
Bedding sand around cables	150 mm	150 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)
Conductor material	**Copper**	**Aluminium**
Conductor cross-section	800 mm^2	800 mm^2
XLPE thickness	15 mm	15 mm
Shield thickness	3 mm	3 mm
PVC thickness	5 mm	5 mm
Soil temperature	15 °C	15 °C
Load current	500 A	500 A

Table 5.12 – Case study 6 variables

5.7.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system's maximum steady state temperature.

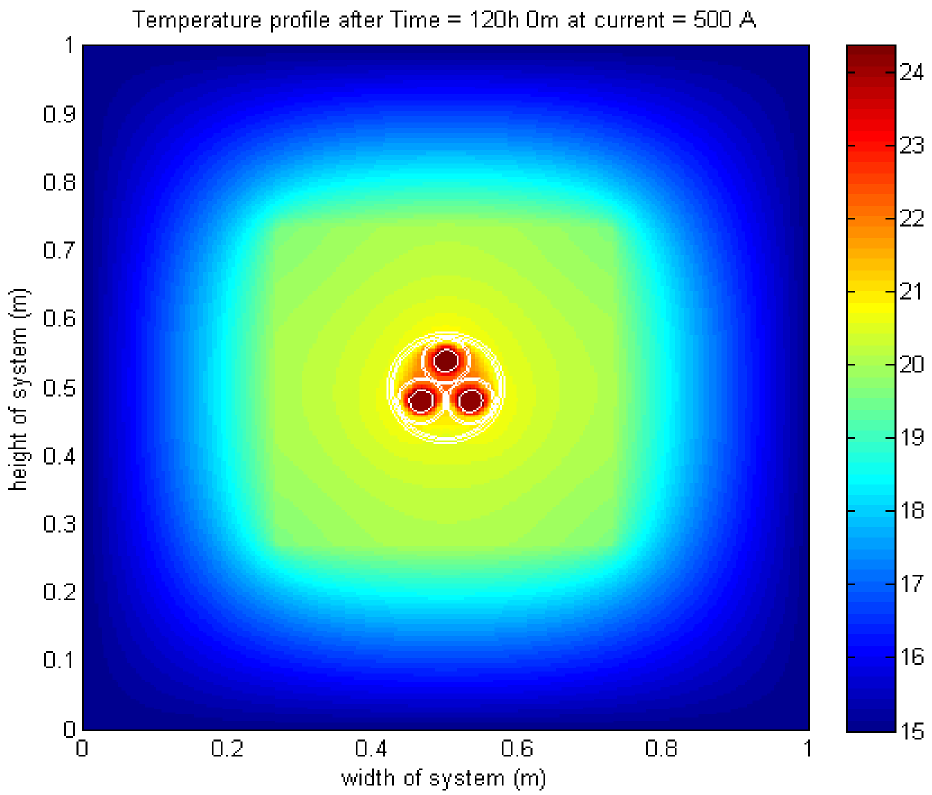


Figure 5.16 – Steady state thermal profile (aluminium)

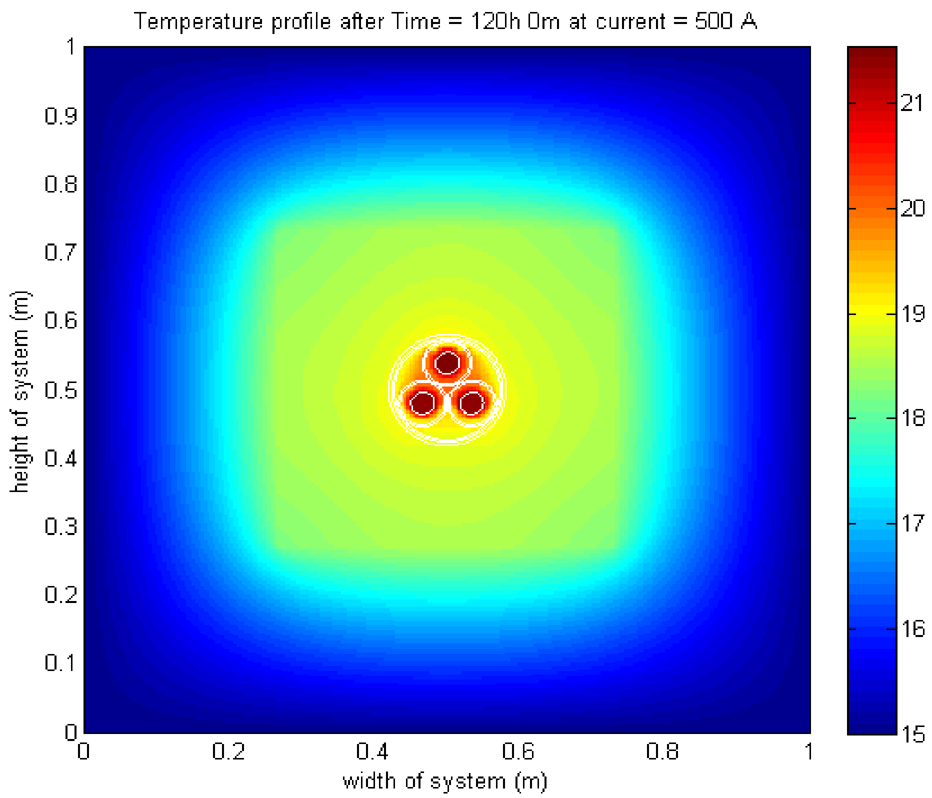


Figure 5.17 – Steady state thermal profile (copper)

5.7.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

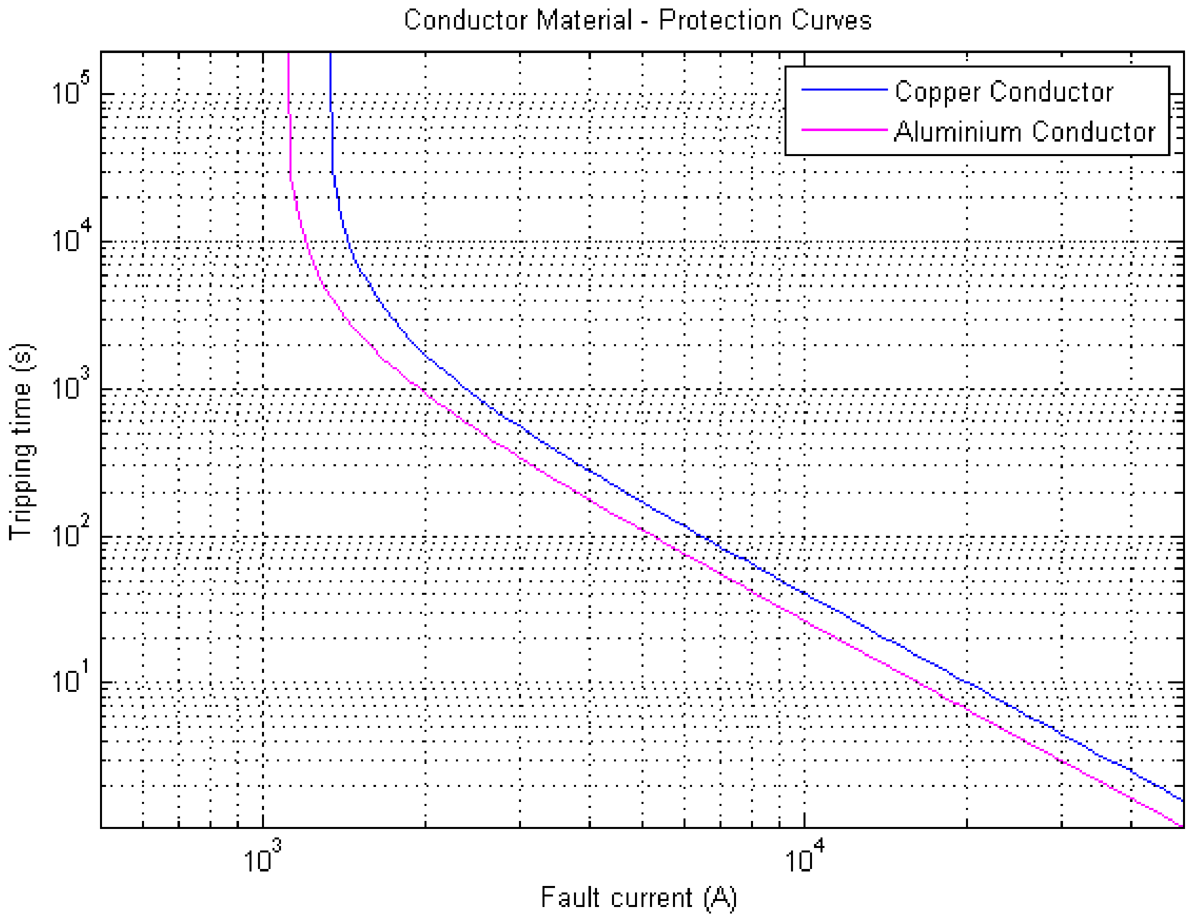


Figure 5.18 – Case study 6 IDMT protection curves

5.7.3) Discussion

Results	**Simulation A** Copper	**Simulation B** Aluminium
Steady state, max T	21.5 °C	24.4 °C
Steady state, ΔT	6.5 °C	9.4 °C
Maximum pick-up current	1623 A	1355 A
IDMT curve	IEC Extremely	IEC Extremely
Pick-up setting	1298 A	1084 A
Time multiplier setting	23.8	22.5

Table 5.13 – Case study 6 results

As anticipated, the higher resistance of the aluminium conductor causes a greater temperature increase in the conductor and the protection must be more sensitive. However, at times it may be feasible to use a larger aluminium cross-section over a smaller copper cross-section to achieve the same load capacity.

5.8) Case study 7 - Depth of lay

Australian standards (AS3000, 2007, p. 159) states direct-buried cables should be a minimum depth of 500mm, legislation on electrical safety regulations states that direct-buried power cables operating at voltages up to 22 kV shall have a minimum depth of 900mm (Victorian Government, 2009). Whilst the majority of the cable system will be buried at least 500mm below the surface, the purpose of this section is to determine if the cable will be more venerable to overheating as it rises up to terminate at equipment above the ground. The following system parameters were used for case study 7 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B	Simulation C
Cable configuration	3 single cables	3 single cables	3 single cables
Depth of lay	**900 mm**	**100 mm**	**0 mm**
Bedding sand around cables	0 mm	0 mm	0 mm
Bedding sand thermal resistance	0.25 W/(m.K)	0.25 W/(m.K)	0.25 W/(m.K)
Soil thermal resistance	0.8 W/(m.K)	0.8 W/(m.K)	0.8 W/(m.K)
Separation between cables	0 mm	0 mm	0 mm
Conductor material	Copper	Copper	Copper
Conductor cross-section	1000 mm^2	1000 mm^2	1000 mm^2
XLPE thickness	30 mm	30 mm	30 mm
Shield thickness	5 mm	5 mm	5 mm
PVC thickness	10 mm	10 mm	10 mm
Soil temperature	15 °C	15 °C	15 °C
Air temperature	20 °C	20 °C	20 °C
Load current	1000 A	1000 A	1000 A

Table 5.14 – Case study 7 variables

5.8.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system's maximum steady state temperature.

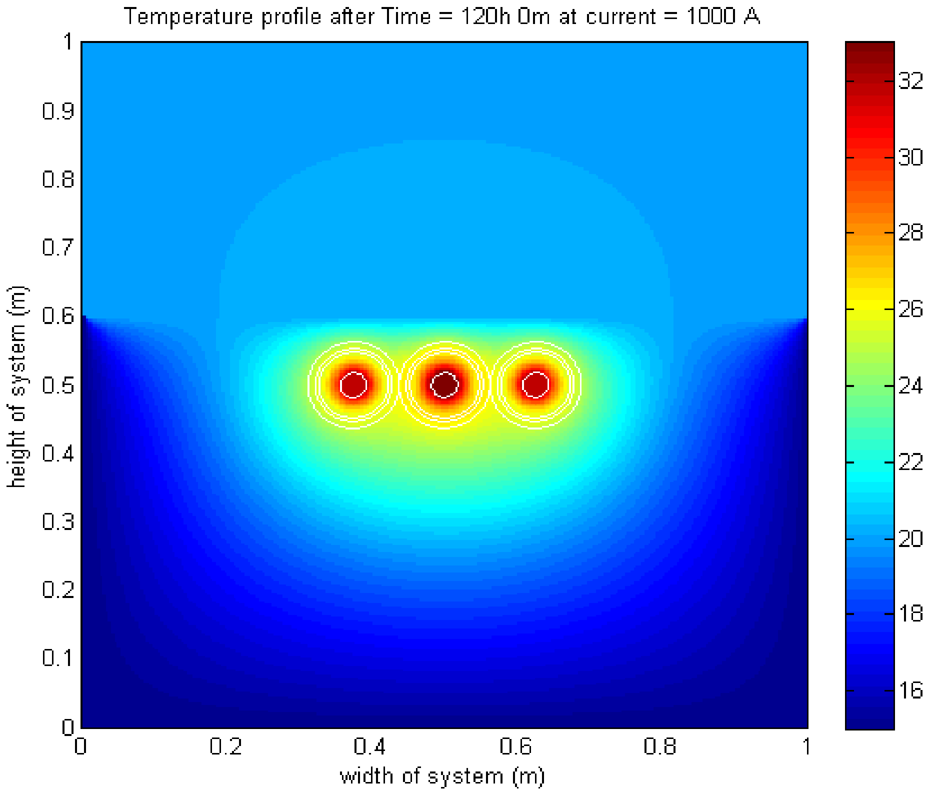


Figure 5.19 – Steady state thermal profile (depth 100mm)

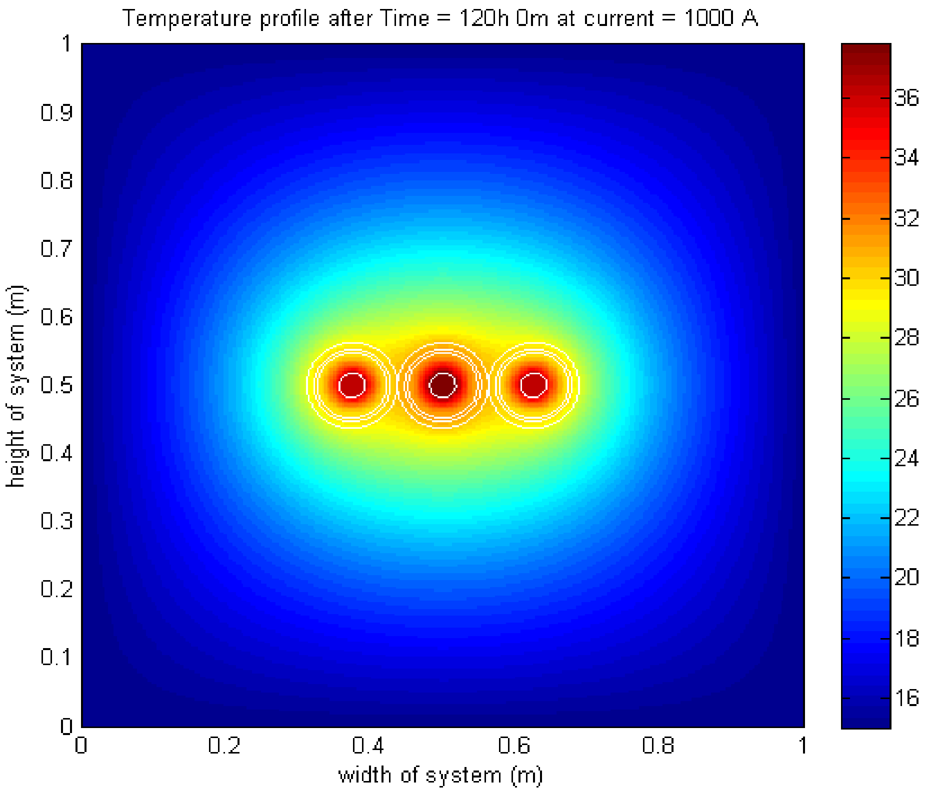


Figure 5.20 – Steady state thermal profile (depth 600mm)

5.8.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

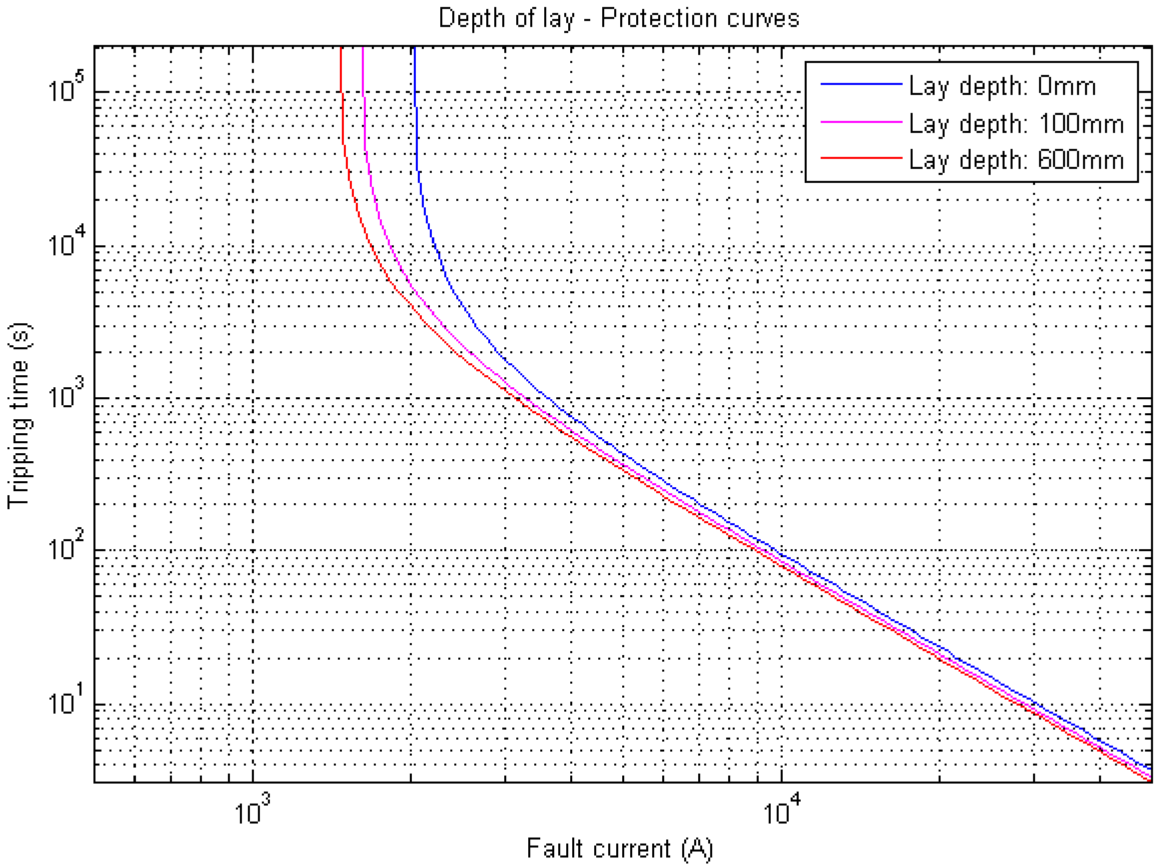


Figure 5.21 – Case study 7 IDMT protection curves

5.8.3) Discussion

Results	**Simulation A** 600mm	**Simulation B** 100mm	**Simulation C** 0mm
Steady state, max T	36.5 °C	32.3 °C	27.6 °C
Steady state, ΔT	21.5 °C	17.3 °C	12.6 °C
Maximum pick-up current	1780 A	1969 A	2483 A
IDMT curve	IEC Extremely	IEC Extremely	IEC Extremely
Pick-up setting	1424 A	1575 A	1986 A
Time multiplier setting	38.7	34.0	23.5

Table 5.15 – Case study 7 results

The results of the above simulation suggest the capacity of the cable increases as it gets closer to the surface. This is consistent with the de-rating values listed in AS3008 and the Gemscab datasheet. Figure 5.21 shows that, for shorter fault times, the conductors have similar trip times, suggesting that the initial temperature rise is governed by the cable materials. As the fault duration increases, heat is shed more quickly to the air than the soil resulting in an increased pick-up current value for cables closer to the surface.

The model does not take into consideration the additional effects of cooling provided by the natural convection of the air. For this reason, care should be taken when using the simulation to model thermal properties of shallow buried cables.

5.9) Case study 8 - Soil properties due to moisture content

The properties of soil are very complex and can vary significantly depending on environmental conditions (Farouke, 1981). This case study was conducted to determine the effect soil moisture content has on the protection required for a cable system. The values used represent very dry and saturated soil to simulate and compare extreme soil conditions. The following system parameters were used for case study 8 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B
Cable configuration	Single trefoil	Single trefoil
Depth of lay	600 mm	600 mm
Bedding sand around cables	0 mm	0 mm
Soil condition	**Wet**	**Dry**
Soil thermal resistance	**2.4 W/(m.K)**	**0.8 W/(m.K)**
Soil specific heat capacity	**1.48 J/(g.K)**	**0.8 J/(g.K)**
Conductor material	Aluminium	Aluminium
Conductor cross-section	630 mm^2	630 mm^2
XLPE thickness	10 mm	10 mm
Shield thickness	3 mm	3 mm
PVC thickness	4 mm	4 mm
Soil temperature	15 °C	15 °C
Load current	630 A	630 A

Table 5.16 – Case study 8 variables

5.9.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system’s maximum steady state temperature.

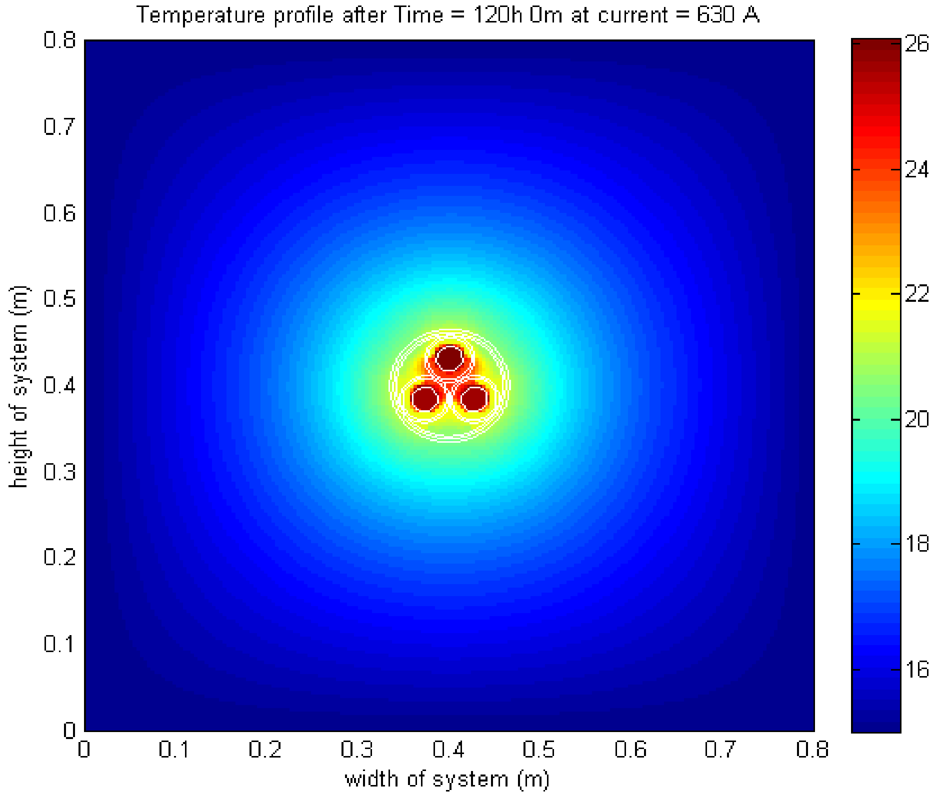


Figure 5.22 – Steady state thermal profile (wet soil)

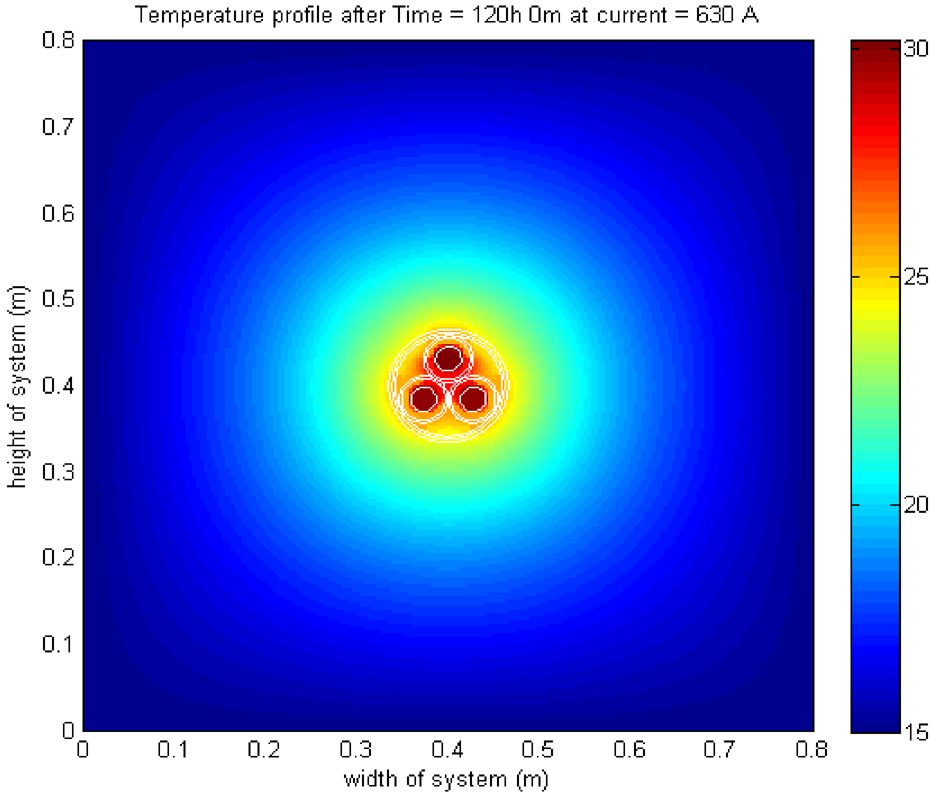


Figure 5.23 – Steady state thermal profile (dry soil)

5.9.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

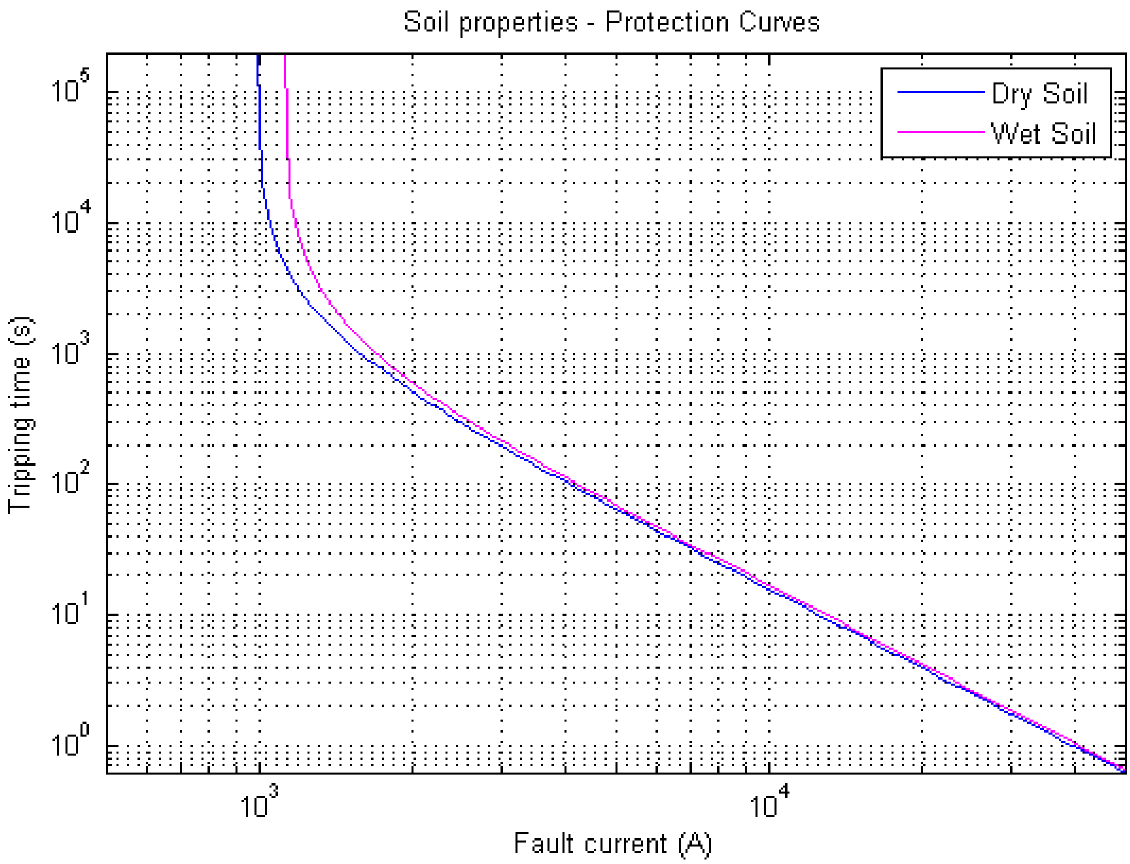


Figure 5.24 – Case study 8 IDMT protection curves

5.9.3) Discussion

Results	**Simulation A** Wet	**Simulation B** Dry
Steady state, max T	26.1 °C	30.2 °C
Steady state, ΔT	11.1 °C	15.2 °C
Maximum pick-up current	1373 A	1211 A
IDMT curve	IEC Extremely	IEC Extremely
Pick-up setting	1098 A	968 A
Time multiplier setting	13.8	16.7

Table 5.17 – Case study 8 results

The results from Table 5.18 show that the capacity of the cable system varies by approximately 10% depending on the soil moisture content. While this is not a huge difference, the most sensitive protection settings would need to be used if designing for all scenarios or modified during a significant drought period. It should be noted that soil properties also vary greatly depending on the soil material at the specific location and there could be significant variation throughout the course of a single cable run.

5.10) Case study 9 - Joint health

The purpose of this case study was to determine how an unhealthy joint may affect the required system protection settings. Generally, the system would be designed with the assumption that all joints were healthy, however, hot-spots may develop within a cable system due to poor workmanship and degradation over time. This case study is not suggesting that protection settings be configured in order to provide longevity to unhealthy joints, rather to determine how such joints will perform under normal operating conditions. The cable joint modelled here is suitable for use on both 400mm^2 and 300mm^2 cable systems therefore both cables were also simulated. The following system parameters were used for case study 9 where the variation between the simulation models has been outlined in red.

Variable	Simulation A	Simulation B	Simulation C	Simulation D
Cable configuration	Trefoil cable	Trefoil cable	Trefoil cable	Trefoil cable
Depth of lay	600 mm	600 mm	600 mm	600 mm
Bedding around cables	**95 mm**	**95 mm**	**100 mm**	**105 mm**
Conductor material	Aluminium	Aluminium	Aluminium	Aluminium
Conductor cross-section	**NA**	**NA**	**400 mm^2**	**300 mm^2**
XLPE thickness	10 mm	10 mm	10 mm	10 mm
Shield thickness	4 mm	4 mm	4 mm	4 mm
PVC thickness	5 mm	5 mm	5 mm	5 mm
Soil temperature	15 °C	15 °C	15 °C	15 °C
Load current	410 A	410 A	410 A	410 A
Joint condition	**Good**	**Poor**	**None**	**None**
Joint resistance	**15μΩ**	**48μΩ**	**NA**	**NA**

Table 5.18 – Case study 9 variables

5.10.1) Thermal results

The following plots were generated without a fixed temperature axis but may be used as a guide to determine the steady state thermal profile. The maximum value on the right hand colour bar reflects the system’s maximum steady state temperature. Note, the use of material boundary lines were removed for these results to demonstrate a feature of the simulation program.

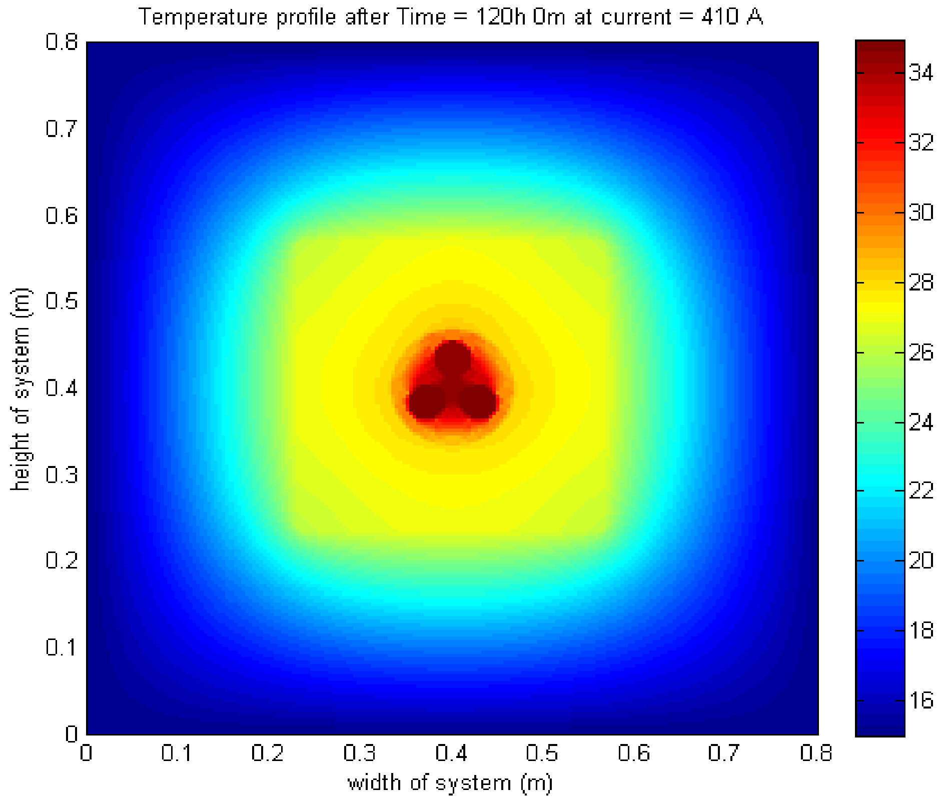


Figure 5.25 – Steady state thermal profile (healthy joint)

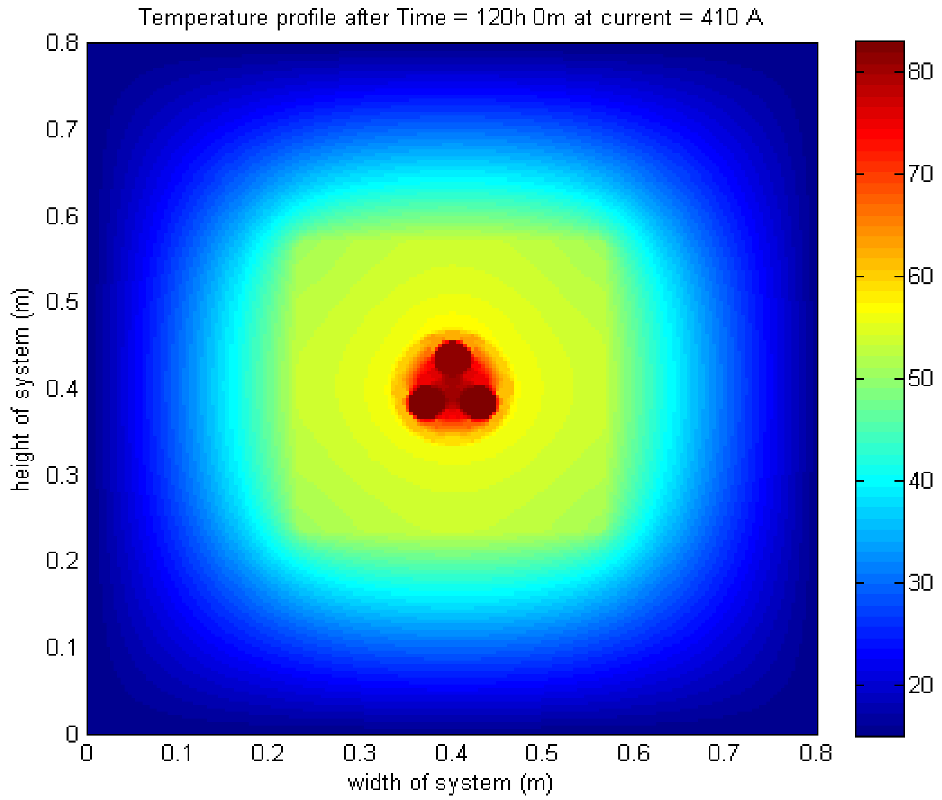


Figure 5.26 – Steady state thermal profile (un-healthy joint)

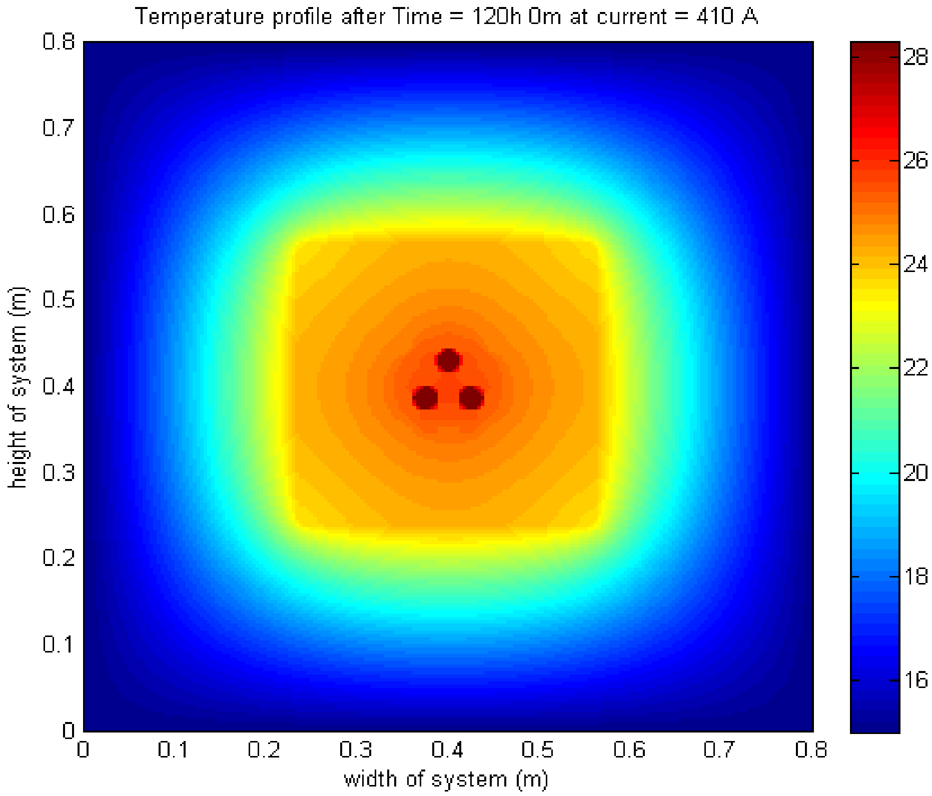


Figure 5.27 – Steady state thermal profile (400mm^2 cable)

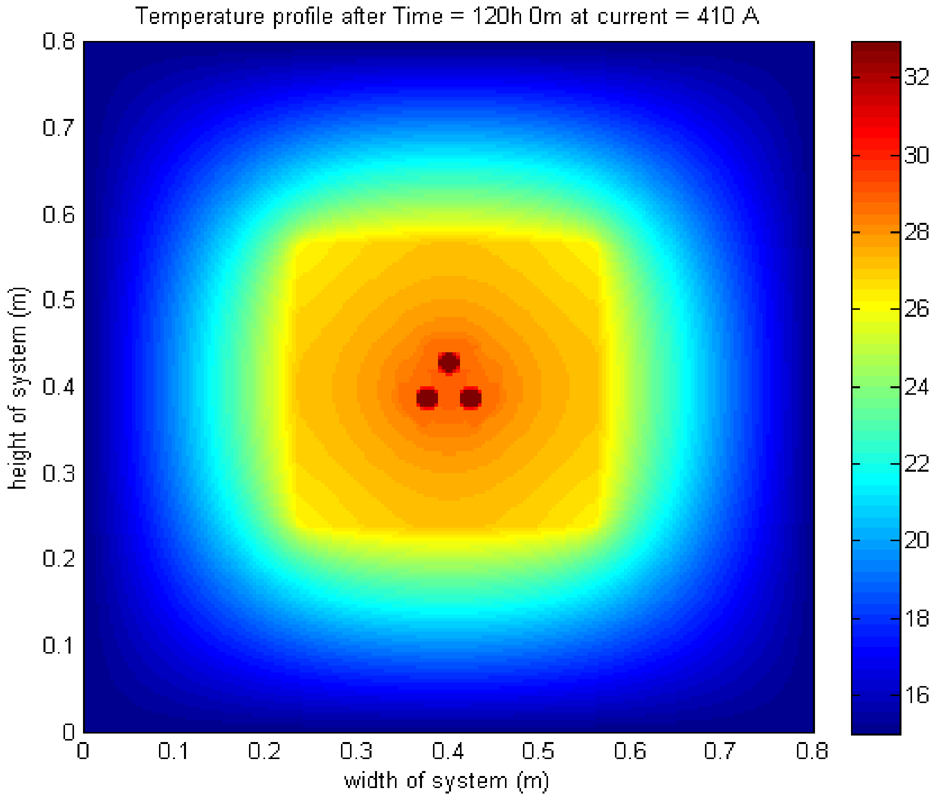


Figure 5.28 – Steady state thermal profile (300mm^2 cable)

5.10.2) IDMT protection curves

The following plot contains a combination of the protection curves found by the simulation.

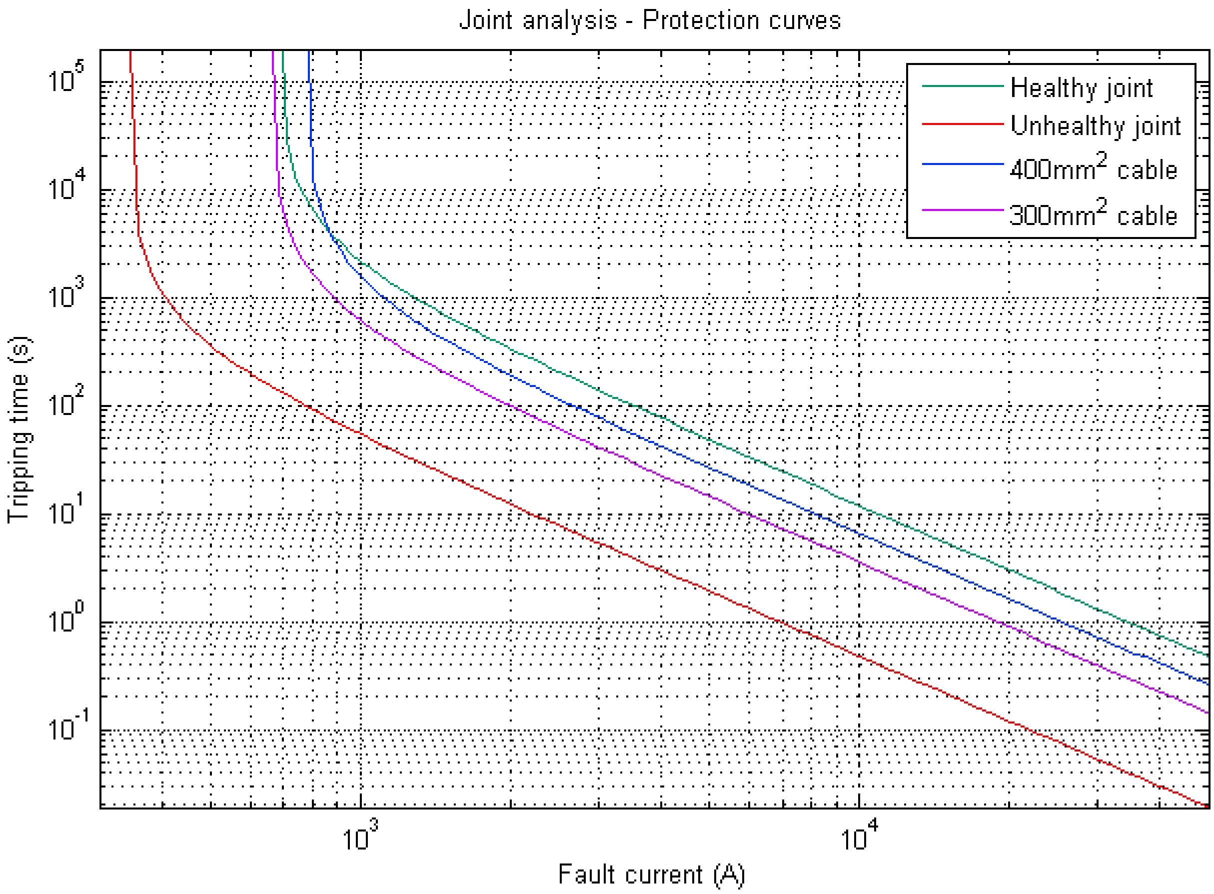


Figure 5.29 – Case study 9 IDMT protection curves

5.10.3) Discussion

Results	**Simulation A** Healthy joint	**Simulation B** Poor joint	**Simulation C** 400mm^2 cable	**Simulation D** 300mm^2 cable
Steady state, max T	34.9 °C	82.9 °C	28.3 °C	33.0 °C
Steady state, ΔT	19.9 °C	67.9 °C	13.3 °C	18.0 °C
Max. pick-up current	848 A	428 A	963 A	819 A
IDMT curve	IEC Extreme	IEC Extreme	IEC Extreme	IEC Extreme
Pick-up setting	678 A	342 A	770 A	655 A
Time multiplier setting	26.0	4.1	11.0	8.3

Table 5.19 – Case study 9 results

The results from the above simulations suggest that the pick-up current is similar for a healthy cable joint in comparison the appropriate cable sizes. The pick-up current for the unhealthy cable joint is approximately half that of the normal system values. The extra resistance at the unhealthy joint also generates a much greater temperature rise in the cable system, which will result in higher levels of fatigue in the insulation surrounding the cable joint.

One thing that can be noticed in Figure 5.29 is the upwards shift in the protection curve of the healthy cable joint. This is due to the increased TMS value and suggests there is a thermal lag associated with the heating of the cable joint. This was further analysed with Figure 5.30 showing the maximum temperature of each simulation throughout the first hour of loading. From this, it can be seen that the initial temperature rise of the healthy joint is slower than that of the cable albeit the joint reaching a higher steady-state temperature. This confirms there is an increased thermal time constant at the joint. The thermal lag is due to the additional conductor material at the joint location which requires more energy and hence more time to heat up the material.

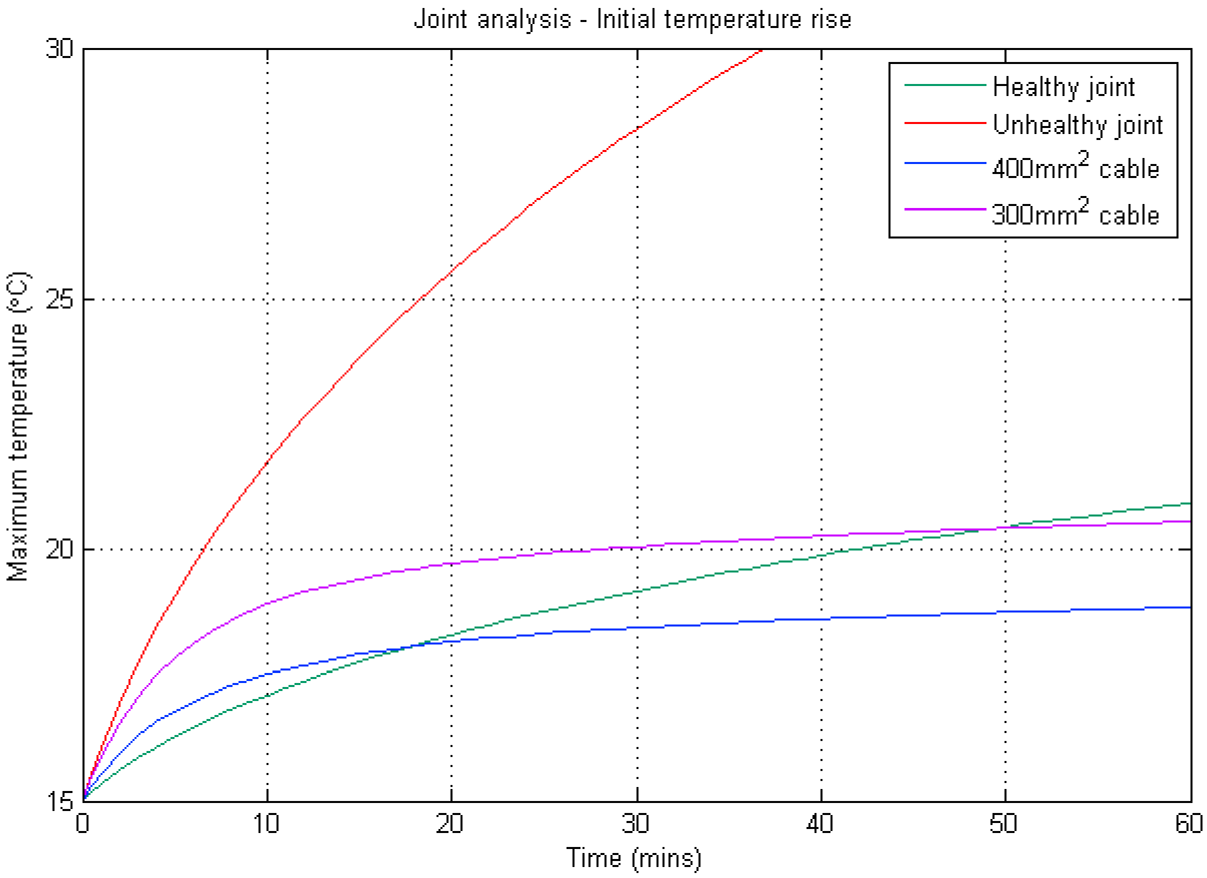


Figure 5.30 – Case study 9 initial temperature rise

The following points should be noted for joint simulation:

1) The size of the surrounding bedding sand was varied slightly to ensure the surface area of soil to bedding sand was constant in each simulation as the outer diameter of the cable varied with the different configurations.
2) The cable joint is modelled as though the resistance is spread across the length of the joint. In reality, a deteriorating joint would create a concentrated hotspot, however, due to the thermal diffusivity properties of the conductor materials, it is assumed the heat from these hotspots would be distributed throughout the cable joint.

5.11) Statistical analysis of joints

A statistical analysis of joints was conducted to create a model that can predetermine the condition of joints and allow for preventative measures to be put in place to prolong the life of underground cable systems. This would offer a reduction in unplanned maintenance cost and improve the delivery of supply through the prevention of faults. The theory outlined in section 3.4) has been used to determine a resistance value representing the statistical worst case joint in the system.

By selecting the "Statistical analysis" feature of the simulation, a joint resistance is calculated depending on the failure rate as a function of system age and the number of joints within the cable run.

Joint resistance (μΩ)		Age of cable system (years)							
		1	**2**	**5**	**10**	**15**	**20**	**30**	**40**
Number of joints in cable system	**1**	15.0	15.0	15.0	15.0	15.0	15.1	15.4	16.1
	3	15.0	15.0	15.0	15.0	15.1	15.3	16.2	18.3
	6	15.0	15.0	15.0	15.1	15.2	15.6	17.4	21.6
	12	15.0	15.0	15.0	15.1	15.4	16.2	19.9	28.2
	30	15.0	15.0	15.0	15.3	16.1	18.0	27.2	48.1
	60	15.0	15.0	15.0	15.5	17.2	20.9	39.3	81.2
	90	15.0	15.0	15.1	15.8	18.3	23.9	51.5	114.3
	300	15.0	15.0	15.2	17.7	25.9	44.7	136.6	346.0
	600	15.0	15.0	15.5	20.3	36.8	74.4	258.3	676.9
	1200	15.0	15.0	16	25.6	58.6	133.7	501.6	1338.8

Table 5.20 – Statistical impedance of the joints within a cable system

Table 5.20 outlines the joint resistance value calculated from the statistical analysis using the number of joints in the system and the age of the system. This data is better represented graphically and Figure 5.31 shows the variation in the statistical joint impedance with respect to joint quantity and system age. The colour represent scenarios where the joint would behave similarly to a healthy (blue) through to an unhealthy (orange) joint. All the scenarios represented as red suggest that at this time, it is likely that at least one joint will have deteriorated beyond satisfactory operating levels and will require replacement. These values appear reasonable when considering the magnitude of cable joint failures on an 11kV underground cable installation, as reported by Mehairjan (2010).

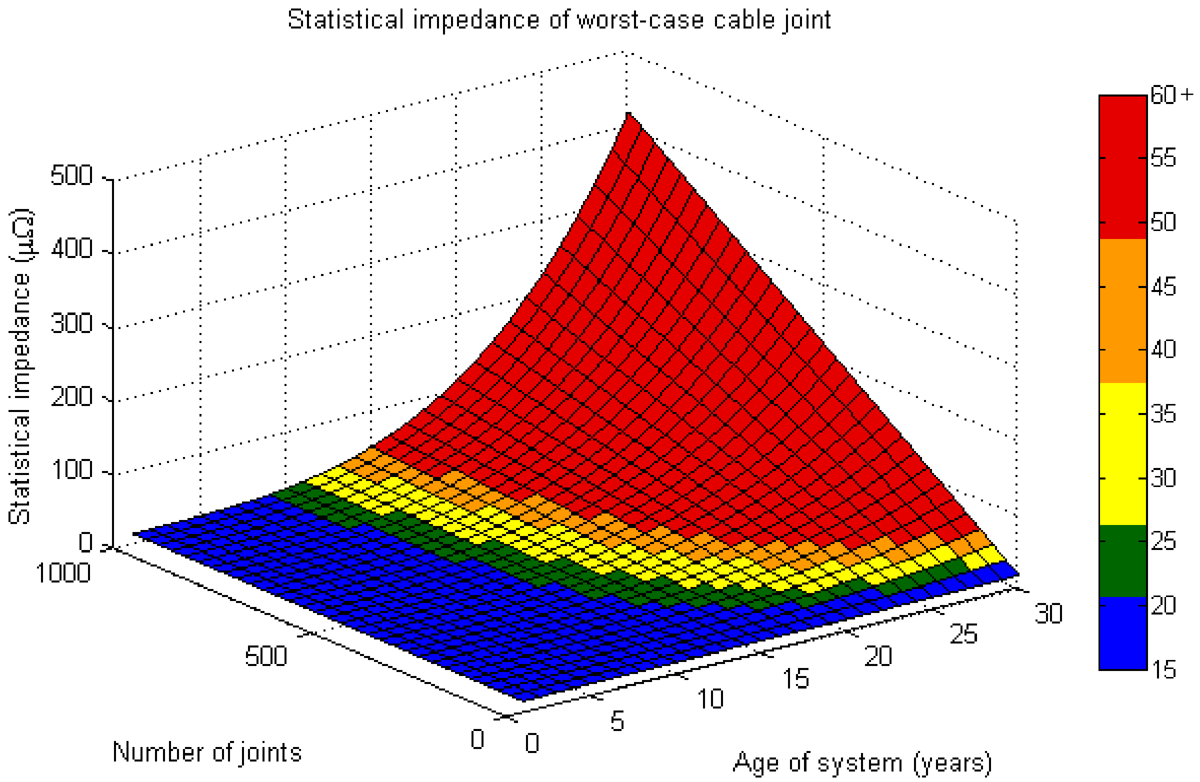


Figure 5.31 – Joint impedance with respect to system age and joint quantity

The statistical model suggests the rate of failure increases exponentially with system age and linearly with the quantity of cable joints.

There are many limitations on the above statistical analysis of cable joints, however, it may provide information to assist in the development of maintenance action plans or modification of protection settings as an aging cable installation transitions into a higher-risk category to improve longevity of the cable system.

6) CONCLUSION

The aim of this research project was to develop a simulation model using finite element analysis that can be used to determine the thermal profile of underground power cables. This model was successfully developed and validated as indicated by the results discussed in *Chapter five*. The model analyses how an underground cable system reacts under load and fault conditions, and subsequently determines the protection settings that are required for the system The range of case studies presented here are indicative of how the simulation model can be used to characterise the influence that operational and environmental conditions have on the required protection of underground cable systems. The findings from these simulations suggest that the protection requirements of a cable system vary significantly depending on the layout, ambient conditions, and operation levels prior to a fault occurring. For example, it was observed that the pre-fault load on the system changed the steady-state thermal profile of the system, which greatly affected the required protection settings (Section 5.5).

A further goal of this research project was to reduce the need for cable de-rating tables (currently used by practicing engineers when determining protection settings), by allowing the user to define and analyse a specific cable system. By reducing the use of generic assumptions and rating tables such as those found in the Australian Standard AS3008, the simulation offers an improved way to determine protection settings for the user's specific application, thus offering greater security and selectivity. To determine the accuracy, and thus the potential applicability of the simulation model, results were validated against trip times derived from manufacturing datasheets and Australian Standards in Section 4.7). Furthermore, in appreciating the need for configurability and ease of use for such a model to be widely applicable to all engineers, it was developed with a user-friendly graphical interface. This interface, as outlined in Appendix B, provides the user with a comprehensive range of configurable variables, which allows the user to determine a specific system's capacity more accurately and thus, better determine the required protection setting values.

This research project builds on the theory outlined in Section 2.1), to not only determine the thermal heating and ampacity of a cable, but to generate protection settings depending on the system itself. The use of this simulation by power system engineers will enhance the accuracy of underground cable protection settings; for the design phase, all the way through to the operation and maintenance of the system. Indeed, the simulation results would work to provide guidance toward electrical network capabilities during times of peak loading, as well as assurance when changing the network configuration during planned maintenance or fault response scenarios. It could also assist in determining the temperature profile of media surrounding a cable system, which would provide engineers with an additional tool to eliminate hotspots within a power system and to understand how an underground power system may interact with other local structures and plant. All of these factors help to ensure power systems operate with the highest level of reliability, minimising unplanned outages and maintaining supply to consumers and production facilities.

6.1) Further work

This research project has successfully developed a configurable simulation tool that has the potential to solve real world engineering problems. However, before this model can be widely utilised, the limitations outlined in Section 4.6) will need to be addressed. This would involve investigating some of these issues and adapting the simulation model to account for any foreseeable variation. For example, the simulation program in its current form takes some time to run the simulation. This could be improved by utilising a mesh approach similar to that outlined by Nguyen (2010) and Zang (2012), which effectively works to reduce the total number of finite elements within the system, therefore, resulting in faster processing time without compromising the accuracy of the model. This would require significant changes to the current mathematical model, however, this simulation could provide an important point of reference in benchmarking the performance of any improved models. Another benefit of using the mesh approach would be to model a larger cross-sectional area without significantly increasing the matrix size; reducing the effects of the boundary conditions of the system and allowing temperature to stabilise over the larger area.

Whilst technologies such as distributed temperature sensing (DTS) using fibre optics provide a real-time analysis of the temperature within a cable system allowing full utilisation of the underground power cable, protection settings remain unchanged with variation to the thermal conditions. This means that the optimal protection is not always available to the system. As technology continues to advance in the field of power protection, new methods such as dynamic configuration of protection settings within the intelligent electrical device (IED) become feasible. For the IED to achieve this, it could be as simple as using the historical load current to determine the settings, or as advanced as using remote measurement stations such as; DTS or devices measuring soil thermal properties, throughout the cable run to better determine the properties of the system. Whilst in theory such a system could be implemented, in allowing the IED to take control of such a critical application, the algorithm and associated equipment would need to undergo rigorous testing to ensure the system is safe and build confidence amongst the potential users.

REFERENCES

AgriInfo. (2011). *Density of Soil: Bulk Density and Particle Density*. Retrieved May 16, 2014, from http://www.agriinfo.in/?page=topic&superid=4&topicid=271

AS 3008. (2009). *Electrical installations - Selection of cables.* Sydney: Standards Australa Limited.

AS3000. (2007). *AS/NZ Wiring Rules.* Sydney: Standards Australia.

Bascom, E. C. (2011). Underground Power Cable Considerations: Alternatives to Overhead. *47th Minnesota Power Systems Conference.* New York: MIPSYCON.

Cigre. (2009). *UPDATE OF SERVICE EXPERIENCE OF HV UNDERGROUND AND SUBMARINE CABLE SYSTEMS.* Paris: Cigre.

Farouke, O. T. (1981). *Thermal Properties of Soils* (1st ed.). Hanover: CRREL.

Fournier, D. (1998). *Aging of defective electrical joints in underground power distribution systems.* Quebec: IEEE.

Fournier, D., & Amyon, N. (2001). *Diagnostic of overheating underground distribution cable joints.* Canada: IEEE.

Gemscab. (2014). *HT-XLPE Cables.* Retrieved May 07, 2014, from http://www.gemscab.com/Gemscab_HT-XLPE.pdf

Hampton, N., Hartlein, R., Lennartsson, H., Orton, H., & Ramachandran, R. (2012). Long-Life XLPE Insulated Power Cable. *Transactions on Dielectrics and Electrical Insulation, 19*(1), 273-282.

Han, S. J. (2006, May 21-24). Overview of Semiconductive Shield Technology in Power Distribution Cables . *Transmission and Distribution Conference and Exhibition*, pp. 641 - 646.

Herpertz, P. (2013, October/November). Cable Fault Location. *Transmission & Distribution*, pp. 30, 33.

Janick, M. (2000). Going underground. *Electrical World, 214*(6), 20.

Lee, K., Yang, J., Choi, Y., & Park, D. (2006, May). Specific Heat and Thermal Conductivity Measurement of XLPE Insulator and Semiconductor Materials. *IEEE*, pp. 805-809.

Megger. (2003). *Fault Finding Solutions.* Dallas: Megger.

Mehairjan, R. P. (2010, November). Application of Statistical Life Data Analysis for Cable Joints in MV Distribution Networks. *Delft University of Technology*.

My Electrical. (2014). *Electromechanical Relays*. Retrieved 8 15, 2014, from http://myelectrical.com/notes/entryid/159/electromechanical-relays

Naskar, A. K. (2013). Thermal Analysis of Underground Power Cables using Two Dimensional Finite Element Method. *CATCON2013* (pp. 94 - 99). IEEE.

Navrud, S., & Ready, R. C. (2008). Valuing the social benifits of avoiding landscape degredation from overhead power transmission lines. *Landscape Research, 33*(3), 281 - 296.

Neher, J. H., & McGrath, M. H. (1957). The Calculation or the Temperature Rise and Load Capability of Cable Systems. *AIE Transactions, 76*(III), 752 - 772.

New Mexico Tech. (n.d.). *Chapter 3 - Basic reliability mathematics.* Retrieved September 28, 2014, from http://infohost.nmt.edu/~olegm/484/Chap3.pdf

Nexans. (2010, Mar). *Nexans.* Retrieved from 6-36kV Medium Voltage Underground Power Cables: http://www.nexans.co.uk/UK/files/Underground Power Cables Catalogue 03-2010.pdf

Nguyen, N., Vu, P., & Tlusty, J. (2010). *New Approach of Thermal Field and Ampacity of Underground Cables Using Adaptive hp-FEM.* IEEE.

Nikishkov, G. P. (2010). Finite Element Equations for Heat Transfer. In XVI (Ed.), *Programming Finite Elements in Java* (p. 402). Springer.

NKT cables. (2009). *High Voltage Cable Systems - Cables and Accessories up to 550 kV.* Cologne, Germany: nkt.

Orton, H. (2013). History of Underground Power Cables. *IEEE Electrical Insulation Magazine*, *29*(4), 52-57.

Peck, D., & Seebacher, P. (2000). *Distributed Temperature Sensing using Fibre-Optics (DTS Systems).* Auckland: Tyree Optech Pty Limited.

QueensU. (2014). *Two-Dimensional Conduction: Finite-Difference Equations.* Retrieved May 09, 2014, from Queen's University - Faculty of Engineering and Applied Science: me.queensu.ca/Courses/346/L_9_4b.pdf

Schneider Electric. (2002). *Extra losses caused in hich current conductors by skin and proximity effects.* Grenoble: Schneider Electric.

Schneider Electric. (n.d.). *Functions: Sepam series 80 - Protection: Tripping curves.* Retrieved July 05, 2014, from http://www2.schneider-electric.com/documents/electrical-distribution/en/shared/interactive-catalogue/seped303005en/seped303005en/pdfs/page_103.pdf

TeKa. (2014). *Laboratory tests of soil samples and sand materials.* Berlin.

The Engineering Toolbox. (2014a). *The Engineering Toolbox*. Retrieved May 07, 2014, from http://www.engineeringtoolbox.com/thermal-conductivity-d_429.html

The Engineering Toolbox. (2014b). *Specific Heat of some common Substances*. Retrieved May 16, 2014, from http://www.engineeringtoolbox.com/specific-heat-capacity-d_391.html

The Engineering Toolbox. (2014c). *Densities of some Common Materials*. Retrieved May 16, 2014, from http://www.engineeringtoolbox.com/density-materials-d_1652.html

The Engineering Toolbox. (2014d). *Metals and Alloys - Densities*. Retrieved May 16, 2014, from http://www.engineeringtoolbox.com/metal-alloys-densities-d_50.html

The Engineering Toolbox. (2014e). *Air Properties*. Retrieved May 17, 2014, from http://www.engineeringtoolbox.com/air-properties-d_156.html

Tyco Electronics. (2000, Aug). *Engineering supplies.* Retrieved Oct 06, 2014, from http://www.engineeringsupplies.com.au/download/section5a.pdf

Tyco Electronics. (2009). *Installation Instruction - Raychem Joint for 3-Core Polymeric insulated Cable with Wire Shield 12 kV to 24 kV.* Ottobrun/Germany: Energy Division.

USQ ELE3804. (2013). USQ Protection Training Stage 2 - Chapter 1: Introduction. Toowoomba.

USQ ENG4104. (2013). *Engineering problem solving simulations.* Toowoomba: University of Southern Queensland.

Victorian Government. (2009). *Electrical Safety (Installations) Regulations.* Retrieved 10 2014, 4, from http://www.esv.vic.gov.au/Portals/0/Electricity Professionals/Files/Legisilation and regulations/ES_Installations_Regulations_2009.pdf

Wiki. (2014a, May 2). *Wikipedia - Joule heating.* Retrieved May 14, 2014, from http://en.wikipedia.org/wiki/Joule_heating

Williams, J. A. (1999, July). Increasing cable rating by distributed fiber optic temperature monitoring and ampacity analysis. *IEEE*, pp. 128-134.

Zang, W. (2012). *A Technique for Assessment of Thermal condition and Current Rating of Underground Power Cables Installed in Duct Banks.* IEEE.

APPENDIX A - PROJECT SPECIFICATION

ENG4111/4112 Research Project

FOR: Greg Nagel

TOPIC: Using thermal properties to determine Inverse Definite Minimum Time (IDMT) settings of underground cable protection

SUPERVISOR: Dr. Tony Ahfock

ENROLMENT: ENG4111 – S1, 2014 – External
ENG4112 – S2, 2014 – External

PROJECT AIM: This project seeks to investigate a means for calculating the IDMT values of underground power system conductors using the thermal properties of the conductive/insulator materials versus the temperature profile of the conductor during the transition from load to fault current.

PROGRAMME: *Revision: 2 – 2/04/14*

1) Research the methods currently used to determine the IDMT values for overcurrent protection of underground cables.
2) Research/determine common underground cable conductive materials (metal compounds) and insulation materials to understand the maximum permissible temperature of the conductor.
3) Research/devise a method for calculating the steady state temperature of the cables based on user defined nominal downstream loads.
4) Research/devise a method for calculating the temperature rise in the conductors during fault/overload currents. Will this be uniform across the conductor or will skin affect come into play here especially at higher voltage levels?
5) Use the temperature rise with respect to the conductor/insulator properties to determine a break curve with respect to fault/overload current. This will be determined by the using values from 4) and user defined information in 2).
6) Use the information calculated in 5), the worst case CB tripping time and a user defined safety margin to best fit the IEEE standard protection curves.
7) Points 3-6 will be implemented in MATLab.
8) Present all of the above in final thesis document.

As time permits:

1) Create user interface in MATLab to allow ease of use and clear presentation of results.
2) Include cable joints in thermal models. These will become hotspots and the resistance will depend on the quality of the joint.
3) Determine how the values will be affected as the cable and joint age.
4) Determine if protection relays could dynamically adjust protection setting values depending on the operating conditions.
5) Include means for overlaying upstream protection curves to ensure there is discrimination and the backup protection will provide sufficient tripping times to avoid damaging the underground cables.
6) Include means to overlay downstream protection curves from IDMT settings and standard fuse curves to ensure the IDMT settings will allow discrimination if the fault to be cleared is not in the primary protection zone.

APPENDIX B - SIMULATION OPERATING INSTRUCTIONS

This appendix gives an overview of how the simulation is configured and run by the user. It may also help in troubleshooting the simulation if It will not start due to an error in the user configuration.

Simulation operating instructions

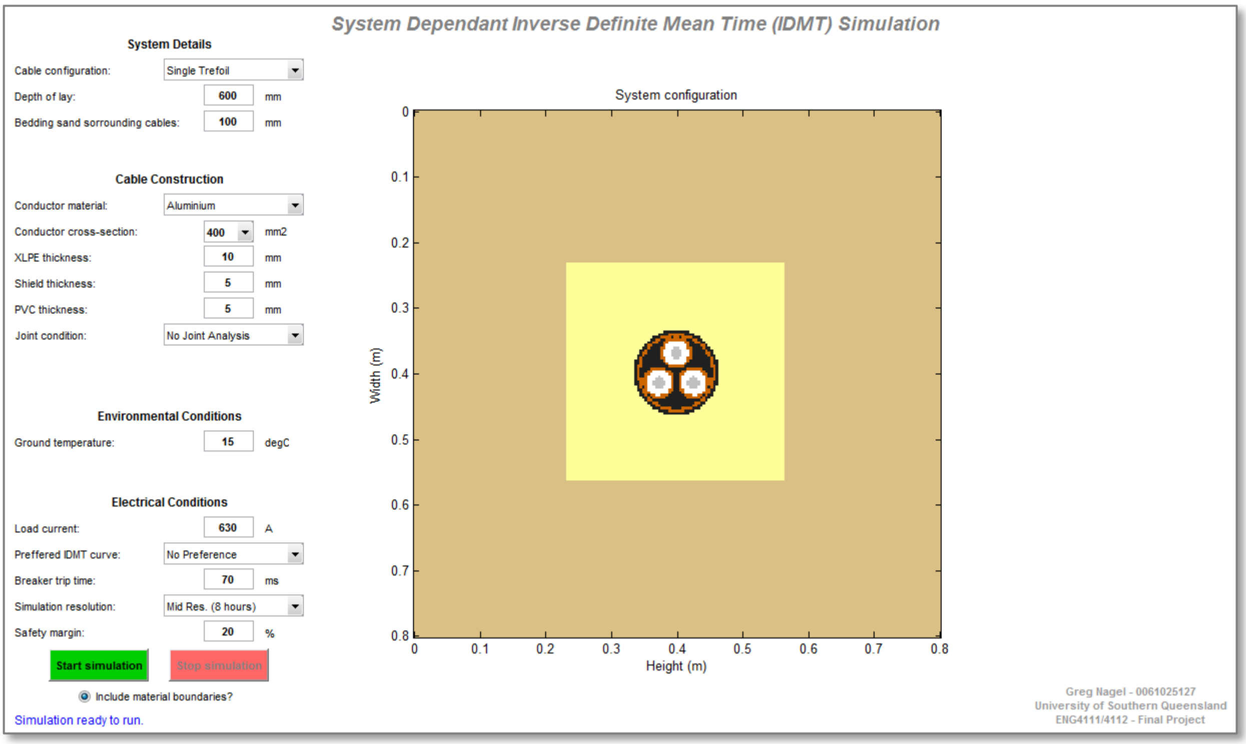
To operate the simulation, the Master.m file shall be run from Matlab.

It is mandatory that the following files reside in the same directory as Master.m or the simulation will generate an error message.

- breakcurve.m
- gui.fig
- gui.m
- layoutMatrix.m
- materialProperties.m
- Parameters.xls
- tempCalc.m
- tempPlot.m

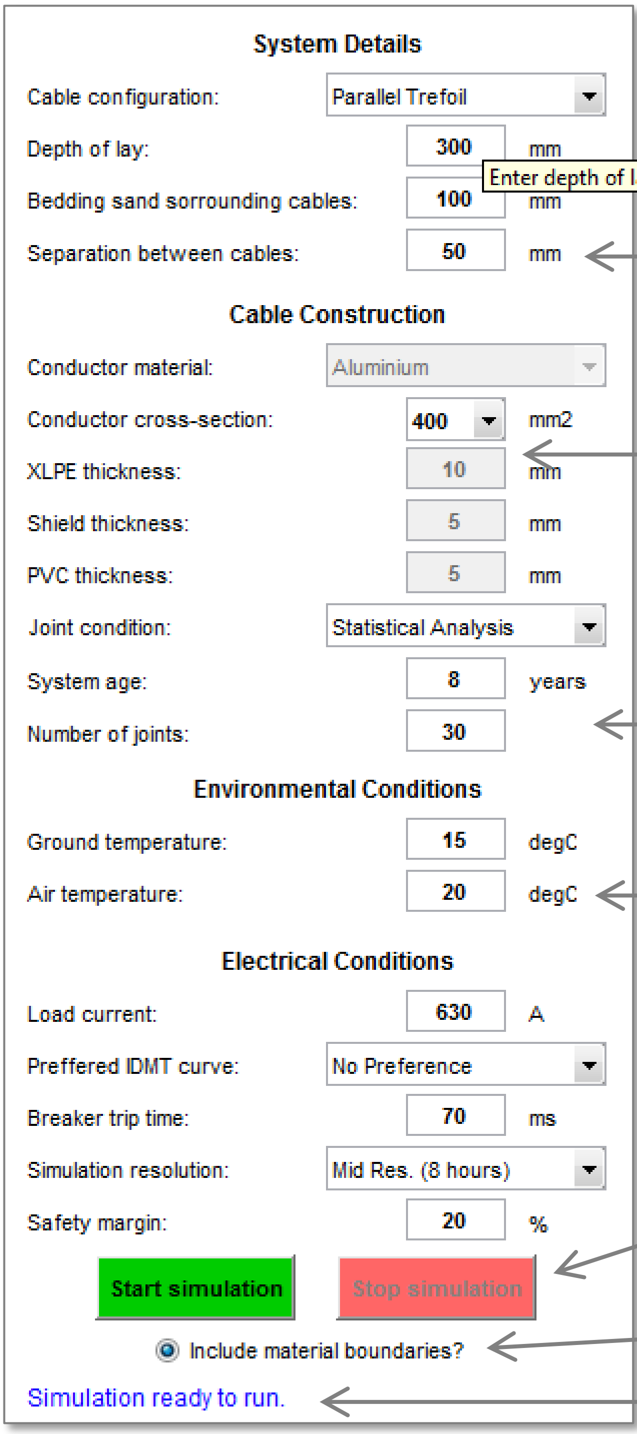
INITIALISATION – SYSTEM CONFIGURATION

When the Master.m file is run, the following window will open.



The left side of this window is where the user will specify the cable system parameters.

If soil thermal resistivity needs to be changed, these can be changed within the Parameters.xls file.



Hovering the mouse over data fields will bring up tool tip information outlining the variable's range.

The 'separation' configuration box will only be visible when a multi-cable system is selected

Fields will go grey when configuration is not available due to another setting in the configuration.

The 'Joint condition' configuration box will only be visible when the cross-section is less than $500mm^2$

'System age' and 'Number of joints' fields will only be visible when 'Statistical Analysis' is selected above.

The 'Air Temperature' configuration box will only be visible when the depth is less than half the system height.

The start button will only be available when all of the user information is valid. Any invalid data fields will appear red. The stop button will only be available once the simulation is running.

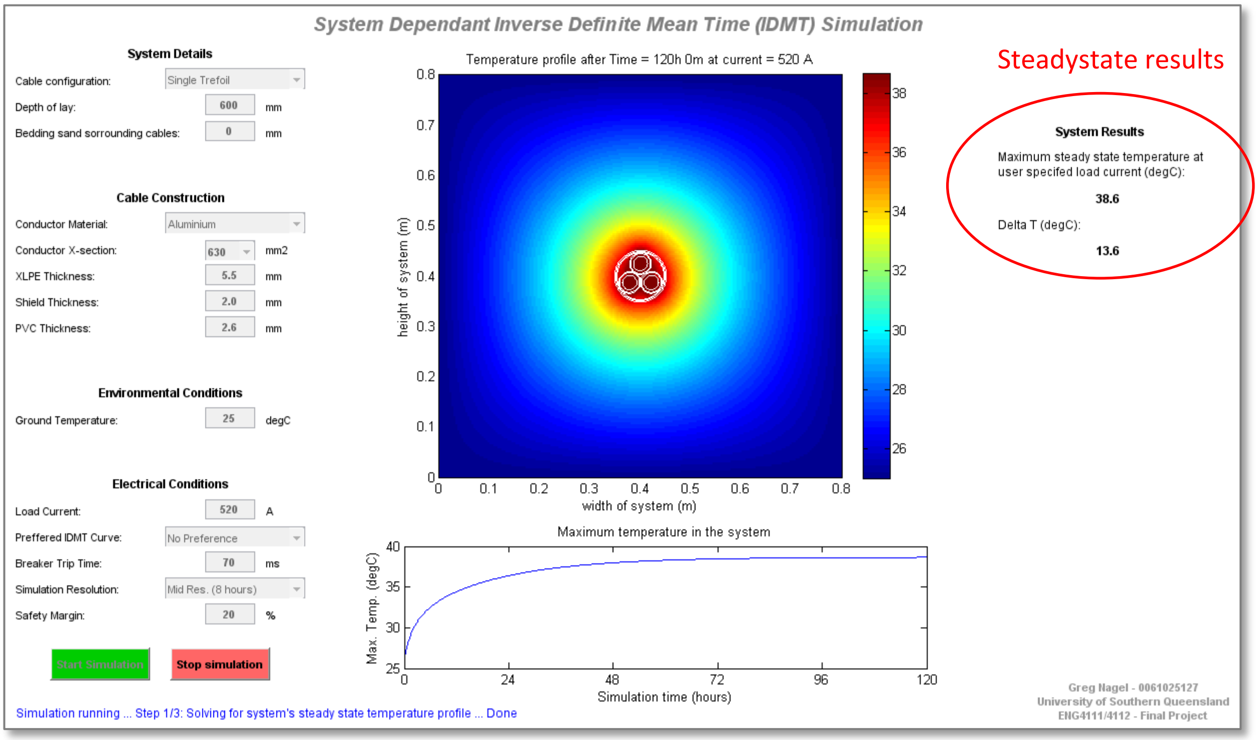
Including material boundaries will show white circles outlining different materials in the cable.

The blue text field updates the user with the status and progress of the simulation.

Simulation Resolution - This field determines the size of each of the finite elements. Increasing the resolution will result in more accurate results, however, the simulation runtime will increase significantly.

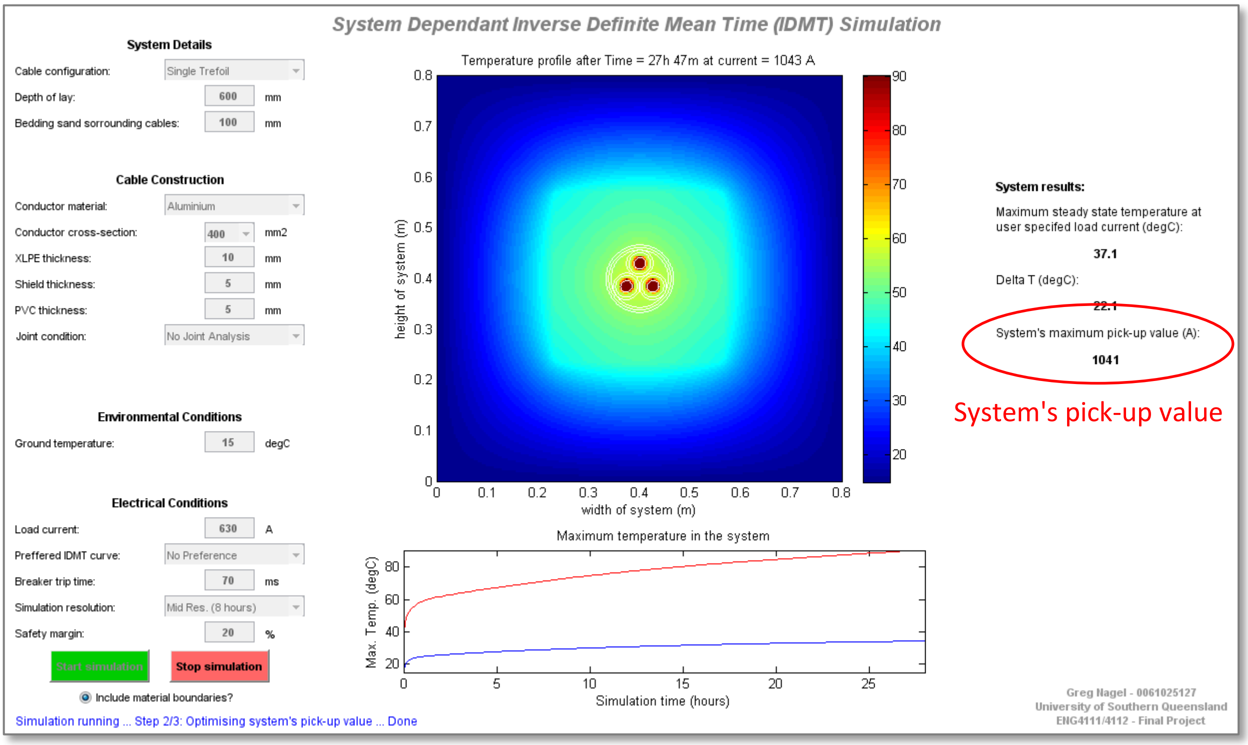
STAGE 1 – STEADYSTATE SIMULATION

This stage simulates the load current on the system to determine the temperature profile at which the system will stabilise.



STAGE 2 – DETERMINING PICK-UP CURRENT

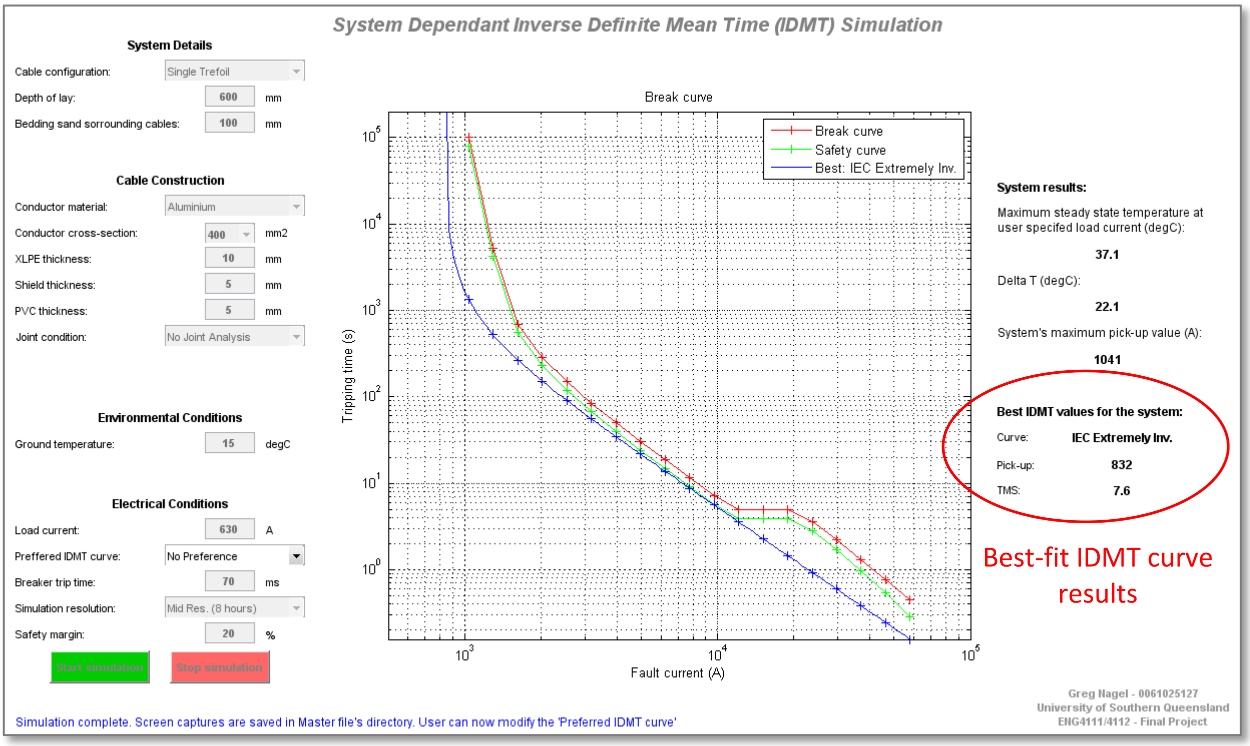
This stage simulates different current values to determine what current value will cause the system's temperature to reach the maximum operating temperature of the cable system.



STAGE 3 – SOLVING SYSTEM BREAK POINTS

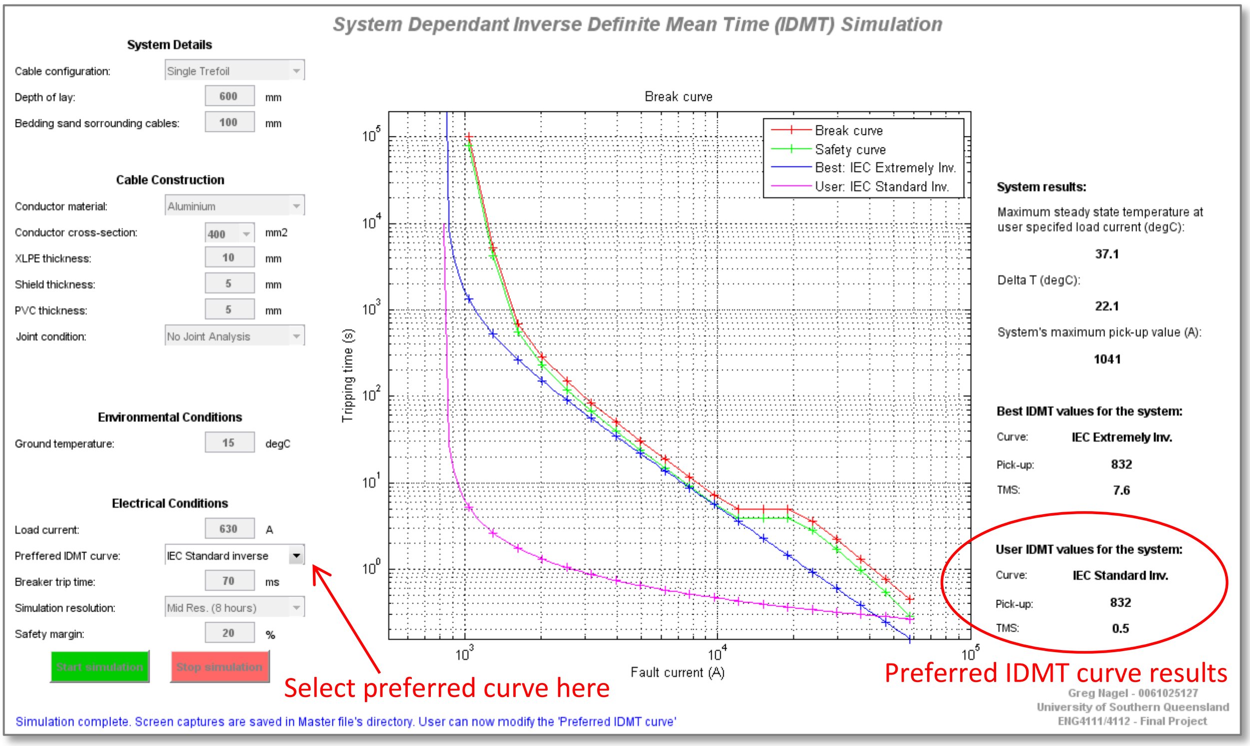
This stage simulates fault current above the level of the pickup current determined in stage 2. Each fault current simulation returns a time value at which the cable will exceed operating requirements, plotted in red. The user defined safety margin and breaker trip time are considered in the green safety curve.

The blue curve is the protection curve the simulation has found to best suit the required protection of the cable system.



RESULTS – COMPARING DIFFERENT PROTECTION CURVES

At this stage, the user may modify the 'Preferred IDMT curve' to determine the protection settings required should another industry standard curve be applied to the cable system.



RESULTS SAVED THROUGHOUT SIMULATION

The following files will be saved to the directory of Master.m as the simulation progresses:

Lx Rx Ixxx Cx Xxxx Jx - 1) Simulation Start

Captures the system layout and user defined settings

Lx Rx Ixxx Cx Xxxx Jx - 2) Steadystate Thermal Profile

Captures the steady state thermal profile and graphically displays this information to the user

Lx Rx Ixxx Cx Xxxx Jx - 3) Pickup Current Thermal Profile

Captures the thermal profile of the pickup current and displays this information to the user

Lx Rx Ixxx Cx Xxxx Jx - 4) IDMT Results

Captures the best fit IDMT curve and protection settings and displays overcurrent plots to the user

Lx	Layout of system (1 = Single cable, 2 = Single trefoil, 3 = 3 x Single, 4 = 2 x Trefoil)
Rx	Resolution of system (1 = Low, 2 = Mid, 3 = High)
Ixxx	Load current in the simulation (xxx designates the current value in A)
Cx	Conductor material (1 = Copper, 2 = Aluminium)
Xxxx	Conductor cross-section (xxx designates the conductor cross-section in mm^2)
Jx	Joint configuration (1 = No joint, 2 = Healthy joint, 3 = Unhealthy joint, 4 = Statistical joint analysis)

Note: Results saved in file 4 will only be saved once at the completion of the simulation. If user would like to save the user defined curve, the curve must be defined prior to starting the simulation.

APPENDIX C - MATLAB CODE STRUCTURE

This appendix gives an overview of how the MATLAB files interact as the simulation progresses through the three major stages.

Stage 1

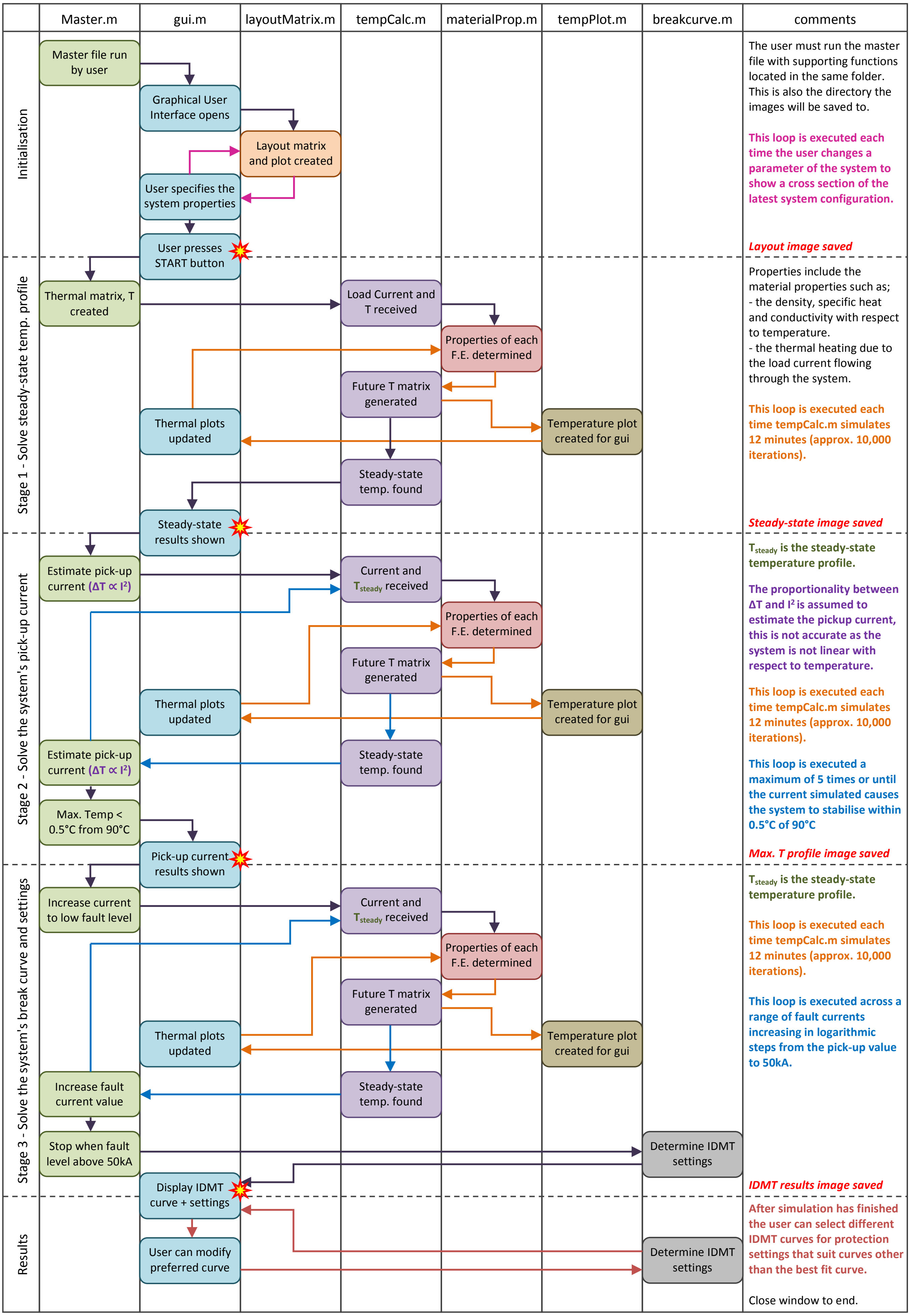
Simulates the current in the system until the temperature profile of the system stabilises and the maximum operating temperature can be found.

Stage 2

Pick-up current values are estimated and simulated for 10,000 seconds onto the steady state profile. This simulation is repeated until the system stabilises within 0.5°C of 90°C (the damage point of XLPE insulation).

Stage 3

The pick-up current found in stage 2 is used as the first point for the break curve. The current value is increased at logarithmic intervals and simulated until the system reaches 90°C. Each fault current and break time is recorded and used to plot the break curve on a log-log axis. Another curve, the safety curve, is then plotted with respect to the safety margin and breaker operating time, as defined by the user. Various industry standard protection curves are then fitted and the regression to the safety curve determined. The best fit curve and setting values are then displayed to the user.


Master.m
gui.m
layoutMatrix.m
tempCalc.m
materialProp.m
tempPlot.m
breakcurve.m
comments
Initialisation
Master file run by user
Graphical User Interface opens
Layout matrix and plot created
User specifies the system properties
User presses START button
The user must run the master file with supporting functions located in the same folder. This is also the directory the images will be saved to.
This loop is executed each time the user changes a parameter of the system to show a cross section of the latest system configuration.
Layout image saved
Stage 1 - Solve steady-state temp. profile
Thermal matrix, T created
Load Current and T received
Properties of each F.E. determined
Future T matrix generated
Thermal plots updated
Temperature plot created for gui
Steady-state temp. found
Steady-state results shown
Properties include the material properties such as;
- the density, specific heat and conductivity with respect to temperature.
- the thermal heating due to the load current flowing through the system.
This loop is executed each time tempCalc.m simulates 12 minutes (approx. 10,000 iterations).
Steady-state image saved
Stage 2 - Solve the system's pick-up current
Estimate pick-up current (ΔT ∝ I²)
Current and T steady received
Properties of each F.E. determined
Future T matrix generated
Thermal plots updated
Temperature plot created for gui
Steady-state temp. found
Estimate pick-up current (ΔT ∝ I²)
Max. Temp < 0.5°C from 90°C
Pick-up current results shown
T steady is the steady-state temperature profile.
The proportionality between ΔT and I² is assumed to estimate the pickup current, this is not accurate as the system is not linear with respect to temperature.
This loop is executed each time tempCalc.m simulates 12 minutes (approx. 10,000 iterations).
This loop is executed a maximum of 5 times or until the current simulated causes the system to stabilise within 0.5°C of 90°C
Max. T profile image saved
Stage 3 - Solve the system's break curve and settings
Increase current to low fault level
Current and T steady received
Properties of each F.E. determined
Future T matrix generated
Thermal plots updated
Temperature plot created for gui
Steady-state temp. found
Increase fault current value
Stop when fault level above 50kA
Determine IDMT settings
Display IDMT curve + settings
T steady is the steady-state temperature profile.
This loop is executed each time tempCalc.m simulates 12 minutes (approx. 10,000 iterations).
This loop is executed across a range of fault currents increasing in logarithmic steps from the pick-up value to 50kA.
IDMT results image saved
Results
User can modify preferred curve
Determine IDMT settings
After simulation has finished the user can select different IDMT curves for protection settings that suit curves other than the best fit curve.
Close window to end.

APPENDIX D - MATLAB CODE

Files:	**Description:**
Master.m (5 pages)	The master file is the core of the system and maintains control of the various stages of the simulation. This file is executed to start the simulation.
breakcurve.m (4 pages)	This function plots the IDMT results and fits the best and user specified curves.
gui.m (22 pages)	The gui function is the interface between the user and the simulation code. This creates the interface window, extracts inputs and outputs results.
layoutMatrix.m (7 pages)	This function creates a matrix that represents each of the finite elements with an integer to map to specific materials within the system.
materialProperties.m (3 pages)	This function file updates the material properties as the temperature varies within the system.
tempCalc.m (6 pages)	This function maintains a temperature matrix representing each finite element and simulates the temperature of each finite element through time.
tempPlot.m (6 pages)	This function creates a colour plot of the system as it is simulated through time. This plot is shown in gui window to show the thermal progression.

```
%% Master.m -------------------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependant IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       05/09/14    Initial release to supervisor for partial review
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This Master file should be run to begin the simulation software developed by Greg
% Nagel to be used as a guideline for determining the IDMT protection settings for
% underground power cables.
%
% Supporting files required the same directory as Master
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% Output files created by simulation
% Portable Network Graphic (.png) images will be created as the software executes.
% This is to allow the user to maintain the information solved by this simulation
% software after the simulation has been closed.
% ----------------------------------------------------------------------------------

%% --------------------------------- Initialisation -----------------------------------

clear all;                                      % clear all variables
clc;                                            % clear command window
close all;                                      % close all figure windows
if ~isempty(findall(0,'Type','Figure'))         % check if gui is already open
    close(gui);                                 % close previous gui's if open already
end

% Initialise global variables that will be used by other functions
global  stepsize ...     % used to determine the time between plot updates in the gui
        rows ...         % number of rows in system  matrices
        cols ...         % number of columns in system matrices
        TmaxS ...        % short term temperature rating of the cable (< 5 sec.)
        TmaxL ...        % long term temperature rating of the cable
        Layout ...       % layout matrix containing different integer for each material
        TFmaxSave ...    % contains fault temperatures and times for stage 2 temp plot
        loops ...        % incremented to count the number of iterations
        Gtemp ...        % ground temp as specified by the user
        I ...            % value of load current as specified by the user
        Atemp ...        % air temp as specified by the user
        wait ...         % flag used to determine when the user inputs are complete
        dt ...           % simulation time step at each iteration
        run ...          % flag used to determine if user has stopped the simulation
        SSMT ...         % steady state maximum temperature caused by load current
        PickUp ...       % pick-up current that will cause system to heat to TmaxS (90)
        SimStep ...      % integer that reflects which stage the simulation is up to
        StatusString...  % string that is output on gui to give status messages to user
```

```
        breakPoints ... % array of break point values solved for the cable system
        Snapshot ...    % flag that defines when the gui should be saved to file
        TempMat ...     % temperature matrix containing temperature of each F.E.
        St2Percent ...  % used to output the percentage of stage 2 completed
        PickupTmax ...  % maximum time relay will count for. i.e. to trip at pick-up
        DeltaT;         % difference between ground temp and maximum steady state temp


%% ----------------------------- Execution ---------------------------------

% Set flags used to determine simulation stages
wait = 1;               % used as a flag to wait for gui information to be complete
run = 1;                % used as a flag to continue solving FE temperatures

% call the GUI for user to enter system information
gui();                  % call graphical user interface (gui)
fprintf('Please operate via gui window\n');  % inform via command window
while wait == 1         % wait until gui inputs are complete
    pause(.1);          % pause 100ms to free up computer processor
end

if run == 1             % if simulation is running (not stopped by user)
    fprintf('Simulation running ...\n');        % inform via command window
end

tic;                    % start timer, used to provide feedback on the run time

% set known global variables
stepsize = 12*60/ dt;   % print out plot every 12 minutes in the simulation
TmaxS = 250;            % short term thermal rating of cable (5 seconds)
TmaxL = 90;             % long term thermal rating of cable
loops = 0;              % counter to aid in the calculation and display of T values
PickupTmax = 10e4;      % maximum time (s) used to determine the pickup values



%% create initial matrix for T

% initialise T matrix to have all entries equal to the ground temperature
T = ones(rows,cols)*Gtemp;      % all temp values are ambient values
T(Layout==7) = Atemp;           % set elements that are above ground to equal air temp

% find the material properties for each point
TempMat = T;                    % set global temperature matrix to equal T
materialProperties();           % call function materialProperties


%% Stage 1, calculate the steady-state temperature matrix for the defined cable system
Tsave = T;                      % save the T matrix

SimStep = 1;                    % simulation step 1/3 is underway

% call the tempCalc function which will solve for a steady state temperature profile
% at the user specified load current
[Tsteady,tsteady] = tempCalc(T,dt,I);  % call tempCalc function

% check if user terminated the program during simulation
if run == 0             % if fun flag has been cleared, user has exited
   break                % exit from the master routine and stop running all together
end                     % if (line 113)

```

```
% find the maximum temperature reached during the steady state simulation
Tmax = max(max(Tsteady(Layout <= 4))); % max temp of conductor/insulator/shield/PVC
SSMT = roundn(Tmax,-1); % round to nearest 1 d.p.

% used in section 2 for pick-up current estimation
DTrequired = TmaxL - Gtemp; % temperature rise above ground temp without cable damage
DeltaT = SSMT - Gtemp;      % temperature rise during steady state

gui();                      % call gui window to update information to the user

Snapshot = 1;               % set flag to save a snapshot of the gui window
gui();                      % call gui function to take snapshot

% output the run time required to solve step 1 of the simulation
toc1 = toc/60;              % save the amount of minutes for stage 1
fprintf('Stage 1 complete, run time = %.0f mins\n\n',toc1)    % output to command window
tic;                        % reset timer, used to provide feedback on the run time


%% Stage 2, determine the pickup current of the cable system
SimStep = 2;                % simulation step 2/3 is underway

% using the fact that T is proportional to I^2 estimate the pickup current that will
% achieve 90 degrees at the maximum relay set time. This is not exact because the
% material properties are not linear within the system
Ip = sqrt( DTrequired / DeltaT) * I;     % solve for estimated Ip

i = 1;                      % initial value for iteration counter
maxT = Tmax;                % Maximum temp within the cable

% solve for pickup current until it is within 1/2 a degree of 90, for no more than 5
% iterations. This is used to improve the initial estimate as system is not linear and
% the heat generated in the cable gets worse as the temperature increases, the above
% estimate will overshoot 90 degrees. A new current value is estimated and this is
% averaged with the previously calculated value and then re simulated.
while abs(maxT - 90) > 0.5 && i <= 5 % while more than 1 deg and < 5 attempts to solve

    TFmaxSave = [];     % clear TFmaxSave so the new attempt can be plotted

    St2Percent = (i-1) / 5 * 100; % progress percentage base value, 20% more each time

    % run simulation with the estimated value
    [Tp,tf] = tempCalc(Tsteady,dt,Ip);  % solve Temp matrix for Tp at Ip

    DTrequired = 90 - SSMT;             % calculation of the required temp difference
    maxT = max(max(Tp(Layout <= 4)));   % Maximum temp within cable materials (1-4)
    DTachieved = maxT - SSMT;           % temperature achieved above the SS maximum

    % display in the command window, the value of the temperature reached so user can
    % be assured the system has diverged to within half a degree of 90 upon review
    fprintf('At iteration %.0f, current used %.0fA, max temp. %.2f \n',i,Ip,maxT)

    % estimate the next current value that will be used to attempt to reach 90 degrees
    % using the theory that delta T is proportional to I^2
    Ipadj = sqrt( DTrequired / DTachieved ) * Ip; % adjusted pickup current
    Ip = (Ipadj + Ip) / 2;  % average the new and last pick-up current values

%     TFmaxSave = [];     % clear TFmaxSave so the new attempt can be plotted
    i = i + 1;          % increment iteration counter

end
```

```matlab

StatusString = ...       % update the status displayed at the bottom of the gui window
    'Simulation running ... Step 2/3: Optimising system''s pick-up value ... Done';

PickUp = floor(Ip);      % round the pick-up current down to the nearest amp

gui();                   % update Pickup results to the user
Snapshot = 1;            % set flag to save a snapshot of the gui window
gui();                   % call gui function to take snapshot

% output the run time required to solve step 2 of the simulation
toc2 = toc/60;           % save the amount of minutes for stage 2
fprintf('Stage 2 complete, run time = %.0f mins\n\n',toc2)    % output to command window
tic;                     % reset timer, used to provide feedback on the run time

%% Stage 3, determine break points of cable system across a range of fault currents

SimStep = 3;             % simulation step 3/3, in the final stage

If = Ip;                 % reference current from which to begin fault calculations
n = 1;                   % initial value for n

% first point of break curve is the pick-up current at the maximum configuration time
tf = PickupTmax;         % maximum pick-up time
fault(n,:) = [If tf];    % store the results for the pickup value

% determine the number of loops that will be executed below to allow percent complete
% to be calculated and displayed to the user
SolveScale = 1.25;       % determines the number of breakpoint calculations
repeats = ceil(log(50e3/Ip) / log(SolveScale)) + 1; % number of breakpoints to solve

% calculate more fault trip times across log based step size to complete the break
% curve, these calues are calculated between pickup current and one point above 50kA
while (If < 50e3)        % calculate up to 50kA

    TFmaxSave = [];      % clear TFmaxSave to allow new thermal plot

    Progress = n / repeats * 100;   % progress percentage to be displayed to the user
    StatusString = sprintf( ...     % update the status at the bottom of gui window
        'Simulation running ... Step 3/3: Solving for system break points to determine break curve ... %0.1f %%',Progress);

    n = n + 1;                      % increase the value of n for the next iteration
    If = SolveScale^(n-1)*Ip;       % break point current value to be next simulated

    [Tf,tf] = tempCalc(Tsteady,dt,If); % solve for above values

    fault(n,:) = [If tf];   % store the results for this fault calculation

    % reduce the simulation time steps to the nearest millisecond as the trip time
    % becomes less to improve the accuracy of the simulation as the temperature
    % changes become more extreme due to the excessive heating of the fault current
    if tf < 10           % if previous trip time was less than 10 seconds
        dt = 0.001;      % reduce simulation time step to 1 ms intervals
    elseif tf < 100      % if previous trip time was less than 100 seconds
        dt = 0.008;      % reduce simulation time step to 8 ms intervals
    end

end                      % end while loop
breakPoints = fault;     % save all the fault calculation information to the global tag

```

```
gui();                     % call the gui function to display the breakpoint plot

Snapshot = 1;                  % set flag to save a snapshot of the gui window
gui();                         % call gui function to take snapshot

% output the run time required to solve step 3 of the simulation
toc3 = toc/60;             % save the amount of minutes for stage 3
fprintf('Stage 3 complete, run time = %.0f mins\n\n',toc3)    % output to command window

% output total run time to the command window
minTot = (toc1+toc2+toc3);    % determine the total number of minutes for the simulation
hours = floor(minTot/60);     % the number of hours to solve the simulation
mins = mod(minTot,60);        % remainder of the hours into minutes
fprintf('Total simulation run time = %.0fh %.0fm\n',hours,mins)     % to command window

% ---------------------------------- End - Master.m -------------------------------- %
```

```
%% breakcurve.m --------------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependant IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       05/09/14    Initial release to supervisor for partial review
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function uses the break points found by simulating different fault currents
% on an underground cable system. Industry standard IEC and IEEE curves are fitted to
% this data to determine the best IDMT protection settings to prevent the underground
% cable from reaching damaging temperatures during a fault on the system. This
% function also allows the user to try different curve and get the best suited setting
% values for the defined curve
% ----------------------------------------------------------------------------------

function breakcurve()

% Initialise global variables shared between MATLAB files
global  curveU ...       % user defined curve
        breakerOp ...    % time for circuit breaker to clear a fault after trip signal
        curveBest ...    % text string of the best-fit curve as found by this function
        TMSet ...        % time multiplier setting for best-fit curve
        Plot ...         % flag used to inform gui when a new thermal plot is ready
        PickUp ...       % pickup current found solved during step 2 of master.m
        breakPoints ...  % break point currents and times
        PickUpSet ...    % pickup setting value considering safety margin and breakerOp
        SafetyM ...      % safety margin as defined by the user
        TMSetUser ...    % time multiplier setting for user defined curve
        PickupTmax ...   % maximum time relay will count for. i.e. to trip at pick-up
        curveUser;       % text string for user defined curve

% set local variables from global data to retain global data
% Fault = breakPoints;          %
Ip = PickUp;                    % pick-up current
Ifault = breakPoints(:,1);      % break point currents from FE analysis
Tfault = breakPoints(:,2);      % break point times from FE analysis
SM = (100-SafetyM)/100;         % convert safety margin to a decimal value


%% find the best fit IEC / IEEE protection curve
Is = Ip * SM;                 % pickup current setting of the system with safety margin
```

```
PickUpSet = floor(Is);      % round down to the nearest amp

% create vectors of 100 points to solve a continuous curve for continuous output plot
a = log10(Is+.1);           % first current point, (just above pickup current)
b = log10(max(Ifault));     % last point to calculate, maximum current value solved
Iplot = logspace(a,b,100);  % create log spaced vector from a to b, 100 intervals

% adjust the original break curve to consider the breaker trip time and safety margin
Fadjusted = Tfault * SM  - breakerOp;

% create an array of strings containing the name of the various curves to be solved
curveNames = ['IEC Standard Inv.    ';
              'IEC Very Inv.        ';
              'IEC Long Time Inv.   ';
              'IEC Extremely Inv.   ';
              'IEC Ultra Inv.       ';
              'IEEE Moderately Inv.';
              'IEEE Very Inv.       ';
              'IEEE Extremely Inv. ';];
% convert to array of strings as above is just a matrix of characters
cellNames = cellstr(curveNames);

% solve the best fit for each of the curve options
for curveB = 1:8          % repeat for each of the 8 curves
    TMS = 0.1;            % initial value for TMS

    % set information about the curve calculations IEC and IEEE
    switch curveB   % get the relevant values to solve 'curveB'
        case 1 %IEC Standard inverse
            k = 0.14;    alpha = 0.02;   beta = 2.97;
        case 2 % IEC Very inverse
            k = 13.5;    alpha = 1;      beta = 1.5;
        case 3 % IEC Long time inverse
            k = 120;     alpha = 1;      beta = 13.33;
        case 4 % IEC Extremely inverse
            k = 80;      alpha = 2;      beta = 0.808;
        case 5 % IEC Ultra inverse
            k = 315.2;   alpha = 2.5;    beta = 1;
        case 6 % IEEE Moderately inverse
            A = 0.01;    B = 0.023;   p = 0.02; beta = 0.241;
        case 7 % IEEE Very inverse
            A = 3.922;   B = 0.098;   p = 2;    beta = 0.138;
        case 8 % IEEE Extremely inverse
            A = 5.64;    B = 0.0243; p = 2;    beta = 0.081;
    end

    tripS(curveB,:) = zeros(1,length(Ifault));     % initiate tripS row to all zeros
    TripPlot(curveB,:) = zeros(1,length(Iplot));  % initiate tripPlot row to all zeros

    while 1    % execute always until a 'break' command is reached

        % solve for the curve using relevant IEC or IEEE equation
        if curveB <= 5    % use IEC equation

            tripT = (k ./ ((Ifault/Is).^alpha - 1) ) * TMS / beta;

        else   % use IEEE equation

            tripT = (A ./ ((Ifault/Is).^p - 1) + B ) * TMS / beta;

        end
```

```
        % check if any value exceeds the break curve of the cable, if so, best case
        % has been solved
        if any(tripT >= Fadjusted)          % if any values exceeds break curve
            TMSsave(curveB,:) = TMS - 0.1;  % use previous value for TMS

            % calculate values for smooth plot
            if curveB <= 5 % use IEC equation

                TripPlot(curveB,:) = ...
                    (k ./ ((Iplot/Is).^alpha - 1) ) * TMSsave(curveB,:) / beta;

            else % use IEEE equation

                TripPlot(curveB,:) = ...
                    (A ./ ((Iplot/Is).^p - 1) + B ) * TMSsave(curveB,:) / beta;

            end

            break                       % exit from this while loop

        else  % curve is acceptable, increase TMS and try again

            TMS = TMS + 0.1;            % increase the value of TMS and try again
            tripS(curveB,:) = tripT;    % save the trip times

        end

    end

    % calculate the curve regression of the log plot vs the adjusted break points
    reg = abs(log10(tripT) - log10(Fadjusted));  % array of regression values
    totReg(curveB) = sum(reg);                   % sum of regression values

end

% find the index location of the minimum regression value, this is the best fit curve
[C,index] = min(totReg);        % find index of best fit curve

% create text to put in legend and gui depending on the curve that was the best fit
bestFit = ['Best: ', cellNames{index}]; % create string for plot legend
curveBest = cellNames{index};           % string to be output in gui
TMSet = TMSsave(index,:);               % set the TMS value for the best fit curve

% check if user has defined a curve to plot alongside the best fit
if curveU ~= 1  % if user curve was selected
    % dynamically create text to put in legend and gui
    userDef = ['User: ', cellNames{curveU-1}]; % create string for plot legend
    TMSetUser = TMSsave(curveU-1,:);           % TMS value for user defined curve
    curveUser = cellNames{curveU-1};           % string to be output in gui
end

Plot = 0;           % turn off plot flag to hide and prevent gui updating thermal plot


%% plot break curves
% plot the break curve points of the system found during stage 3 of Master.m
loglog(Ifault,Tfault,'-+r')         % plot with points and line
hold on                             % retain data so other lines can be added

% plot the break curve points of the adjusted curve considering breaker operating time
```

```
% and safety margin
loglog(Ifault,Fadjusted,'-+g')      % plot with points and line

title('Break curve')                % set title of plot
xlabel('Fault current (A)')         % set x axis label
ylabel('Tripping time (s)')         % set y axis label
grid on;                            % activate grid lines

% add best fit IDMT curve to the loglog plot
loglog(Iplot,TripPlot(index,:),'b')

if curveU ~= 1  % if user curve was selected
    % add user defined curve to the loglog
    loglog(Iplot,TripPlot(curveU-1,:),'m')

    % add legend to plot including the name of the user defined curve
    legend('Break curve','Safety curve', bestFit, userDef, 'Location','NorthEast');

    % show points used to solve curve
    loglog(Ifault,tripS(curveU-1,:),'+m')

else
    % add legend to plot including only the name of the best fit curve
    legend('Break curve','Safety curve', bestFit, 'Location','NorthEast');
end

% show points used to solve best curve
loglog(Ifault,tripS(index,:),'+b')

Imin = min(roundn(0.8*Ifault(1),2) , 500);    % find the minimum current value to plot
xlim([Imin 100000]);                          % set x axis values for plot
ylim([0 2*PickupTmax]);                       % keep graph on a consistent axis

hold off                                      % turn off hold of plot data

% -------------------------------- End - breakcurve.m ------------------------------ %
```

```matlab
%% gui.m ---------------------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependent IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       13/09/14    Initial release to supervisor for partial review
% 1.1       30/09/14    Include statistical analysis of cable joint and radio button
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function defines how the Graphical User Interface (GUI) interfaces between the
% user and the associated files required for the simulation. Many of the functions
% within this file are automatically generated by the MATLAB gui creator. This file is
% complemented by gui.fig which is the file that contains all the graphical
% information required for the gui to operate.
% -------------------------------------------------------------------------------------

% MATLAB automated function
function varargout = gui(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% end - gui()
% End initialization code - DO NOT EDIT


% This function executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
```

```

% Initialise global variables that are shared between this and other functions
global  depth ...       % burial depth of cable (used to determine if air is shown)
        separation ...  % distance between conductors (when multiple conductors)
        insul ...       % insulation thickness of the cable
        shield ...      % thickness of cable shield
        pvc ...         % thickness of pvc
        Gtemp ...       % ground temp as specified by the user
        Atemp ...       % air temp as specified by the user
        bedding ...     % thickness of bedding sand surrounding the cable
        I ...           % value of load current as specified by the user
        breakerOp ...   % time for circuit breaker to clear a fault after trip signal
        wait ...        % flag used to determine when the user inputs are complete
        run ...         % flag used to determine if user has stopped the simulation
        breakPoints ... % break point currents and times
        StatusString... % string that is output on gui to give status messages to user
        Snapshot ...    % flag that defines when the gui should be saved to file
        SafetyM ...     % safety margin as defined by the user
        filename ...    % used throughout gui for filename of image saves
        system ...      % cable system configuration
        Joint;          % integer set by user to determine joint status (1 = no joint)

% Automatic code generated by MATLAB
handles.output = hObject;    % Choose default command line output for gui
guidata(hObject, handles);   % Update handles structure

% place gui in the centre of the screen, until the simulation has started, then it
% will remain where it is or where the user has moved it to.
if wait == 1                 % if simulation is waiting to start
    movegui(gcf,'center')    % place gui in the centre of the screen
end

% hide cable separation field until the three phase or dual trefoil has been selected
if system >= 3               % if system is 3 or 4
    set(handles.Separation, 'Visible', 'On');   % show cable separation fields
    set(handles.text6, 'Visible', 'On');        % show cable separation fields
    set(handles.text9, 'Visible', 'On');        % show cable separation fields
else                         % if system is 1 or 2
    set(handles.Separation, 'Visible', 'Off');  % hide cable separation fields
    set(handles.text6, 'Visible', 'Off');       % hide cable separation fields
    set(handles.text9, 'Visible', 'Off');       % hide cable separation fields
end

if Joint == 4        % check if statistical analysis has be requested
else                 % no statistical analysis, hide analysis data fields
    hideJointFields(handles)                    % hide joint data fields
end


% Disable push buttons until conditions are acceptable for them to be pressed.
set(handles.StartButton,'Enable','off');     % turn on the function of the start button
if (run == 1 & wait == 0)                    % if system is running through simulation
   set(handles.StopButton,'Enable','on');    % turn on the function of the stop button
else                                         % if simulation is not running
    set(handles.StopButton,'Enable','off');  % turn off the function of the stop button
end

% Disable joint select as initial cable cross-section is too large
if (wait == 1)                    % if system is running through simulation
    set(handles.JointSelect,'Visible','off');   % hide joint selection field
    set(handles.JointText,'Visible','off');     % hide joint text field
```

```
end

% During initialisation of the gui window, save the default values for the user
% inputs to be used within other functions.
depth = str2num(get(handles.DepthOfLay,'string'))/1000; % depth of cable lay
separation = str2num(get(handles.Separation,'string'))/1000;  % cable separation
Atemp = str2num(get(handles.Atemp,'string'));            % ambient air temperature
Gtemp = str2num(get(handles.Gtemp,'string'));            % ambient ground temperature
bedding = str2num(get(handles.Bedding,'string'))/1000;   % thickness of bedding sand
I = str2num(get(handles.Load,'string'));                 % load current
breakerOp = str2num(get(handles.BreakOp,'string'))/1000;% breaker trip time
SafetyM = str2num(get(handles.safetyM,'string'));        % safety margin
insul = str2num(get(handles.XLPE,'string'))/1000;        % thickness of XLPE
shield = str2num(get(handles.Shield,'string'))/1000;     % thickness of Shield
pvc = str2num(get(handles.PVC,'string'))/1000;           % thickness of PVC

% check if there are results for the break curve of the system. If there is, configure
% the gui window for displaying the break curve outputs.
if breakPoints              % if there are results for the curve

    set(handles.StopButton,'Enable','Off');     % disable stop button

    % clear and hide axis 1
    axes(handles.axes1);     % select plot1 as active set of axis
    delete(colorbar);        % remove the colourbar
    cla(handles.axes1);      % clear the axis 1
    set(handles.axes1, 'Visible', 'Off');       % make axis 1 invisible

    % clear and hide axis 2
    axes(handles.axes2);     % select plot2 as active set of axis
    delete(legend);          % delete the plot legend
    cla(handles.axes2);      % clear the axis 2
    set(handles.axes2, 'Visible', 'Off');       % make axis 2 invisible

    % output IDMT curves onto axis 3
    set(handles.axes3, 'Visible', 'On');        % make axis 3 visible
    axes(handles.axes3);     % select plot3 to be updated
    breakcurve();            % call the break curve function which will update plot

    % take a snapshot of the system when the IDMT values have been solved
    if Snapshot   % if snapshot flag has been set
        % save an image of GUI for user
        hgexport(gcf, sprintf([filename '- 4) IDMT results']), ...   % filename info
            hgexport('factorystyle'), 'Format', 'png');              % image type
        Snapshot = 0;       % clear snapshot flag
    end

    % Update the status to inform the user of the status of the simulation
    StatusString = 'Simulation complete. Screen captures are saved in Master file''s directory. User can now modify the ''Preferred IDMT curve''';          % status update to the user
    set(handles.Status, 'String', StatusString) % update status

else            % if no results are available for the IDMT curve

if wait == 1    % if still waiting for user configuration of the system
    set(handles.boundaryButton,'Value',1);  % default radio button to on

    % hide other axes as not required yet
    set(handles.axes1, 'Visible', 'Off');       % make axis 1 invisible
    set(handles.axes2, 'Visible', 'Off');       % make axis 2 invisible
```

```
    % initialise the graphical representation of the default system configuration
    axes(handles.axes3);  % update plot onto axis 3
    layoutMatrix();       % update the graphical representation of the system

else            % if simulation has begun
    % clear any data from axes 3 (layout colour by numbers)
    cla(handles.axes3)                          % clear active axis
    set(handles.axes3, 'Visible', 'Off');       % make axis 3 invisible

end

% call function to check if user inputs are acceptable and if so, make start button
% available for user to begin simulation
checkValid(handles)        % call in-line function

end
% end - gui_OpeningFcn()


% this function executes when the gui function is called after it has been established
function varargout = gui_OutputFcn(hObject, eventdata, handles)

% Automatic code generated by MATLAB
varargout{1} = handles.output;% Get default command line output from handles structure

% Initialise global variables that are shared between this and other functions
global  Gtemp ...       % ground temp as specified by the user
        run ...         % flag used to determine if user has stopped the simulation
        StatusString... % string that is output on gui to give status messages to user
        Snapshot ...    % flag that defines when the gui should be saved to file
        filename ...    % used throughout gui for filename of image saves
        Plot ...        % flag used to inform gui when a new thermal plot is ready
        TmaxSave ...    % contains fault temperatures and times for stage 1 temp plot
        TFmaxSave ...   % contains fault temperatures and times for stage 2 temp plot
        SSMT ...        % steady state maximum temperature caused by load current
        PickUp ...      % pick-up current that will cause system to heat to TmaxS (90)
        SimStep ...     % integer that reflects which stage the simulation is up to
        PickUpSet ...   % pickup setting value considering safety margin and breakerOp
        TMSet ...       % time multiplier setting for best-fit curve
        curveBest ...   % text string of the best-fit curve as found by this function
        TMSetUser ...   % time multiplier setting for user defined curve
        curveU ...      % user defined curve
        PickupTmax ...  % maximum time relay will count for. i.e. to trip at pick-up
        curveUser ...   % text string for user defined curve
        Days ...        % No. of days simulation should run for to reach steady state
        Layout ...      % layout matrix containing different integer for each material
        DeltaT ...      % difference between ground temp and maximum steady state temp
        Button;         % radio button status for showing the boundary circles on plot

set(handles.Status, 'String', StatusString)   % update status at the bottom of the gui

Button = get(handles.boundaryButton,'Value');  % get status of the radio button


if Plot == 1            % if the plot flag has been set
    if run == 1         % and simulation is still running

        % take a snapshot of the system following the system reaching steady state
        if Snapshot     % if snapshot flag has been set
            if SimStep == 1
```

```
                % save an image of GUI for user to review the data from steady state
                hgexport(gcf, sprintf([filename '- 2) Steadystate Thermal Profile']), hgexport('factorystyle'), 'Format', 'png');
            elseif SimStep == 2
                % save an image of GUI for user to review data from pick-up simulation
                hgexport(gcf, sprintf([filename '- 3) Pickup Current Thermal Profile']),hgexport('factorystyle'), 'Format', 'png');
            end
            Snapshot = 0;           % clear snapshot flag
        end

        % update thermal distribution colour plot
        axes(handles.axes1);        % select axis 1 to be updated
        tempPlot();                 % call function temp plot to update the plot
        set(handles.axes1, 'Visible', 'On');  % turn on axis 1

        % update temp vs time plot
        axes(handles.axes2);        % select axis 2 to be updated

        if SimStep == 3             % if simulation is in stage 3

            plot(TFmaxSave(:,2)*60,TFmaxSave(:,1))  % plot the temp. vs time (sec)
            xlabel('Simulation time (seconds)');    % x axis label
            Ym = max(90, (max(TFmaxSave(:,1))) );   % y axis max is at least 90 deg
            ylim([Gtemp Ym]);                       % set y axis limits

        else                        % simulation is in stage 1 or 2

            plot(TmaxSave(:,2)/60,TmaxSave(:,1))    % plot the temp. vs time (hours)

            if SimStep == 1         % simulation is in stage 1

                xmax = max(TmaxSave(:,2)/60);       % find maximum time value of plot
                hours = (Days*24);                  % find the max simulation time

                if xmax >= hours    % if max time exceeds the max sim time (cosmetic)
                    xlim( [0 hours] );              % limit graph axis to max time
                    set(gca,'XTick',[0:24:hours])   % set X marker locations every 24h
                end

            elseif SimStep == 2     % simulation is in stage 2

                hold on             % hold the plot data from stage 1 for comparison
                plot(TFmaxSave(:,2)/60,TFmaxSave(:,1),'r')  % plot the temp. vs time
                Ym = max(90, (max(TFmaxSave(:,1))) );  % y axis max is at least 90 deg
                ylim([Gtemp Ym]);                      % set y axis limits
                xlim([0 ceil(PickupTmax/60/60)]);      % set x axis limits
                hold off            % release the plot data hold

            end

            xlabel('Simulation time (hours)');         % x axis label

        end

        title('Maximum temperature in the system');    % title of the plot
        ylabel('Max. Temp. (degC)');                   % y axis label

    end
end

```

```
% remove the fields for air temperature if the depth of lay means no air is in system
if Layout ~= 7                      % if no entries are equal to 7 (air)
    set(handles.text29, 'Visible', 'Off');        % hide Atemp info
    set(handles.text32, 'Visible', 'Off');        % hide Atemp info
    set(handles.Atemp, 'Visible', 'Off');         % hide Atemp info
end


if SSMT          % if SSMT has been set (end stage 1) display the results
    set(handles.SystemRes, 'Visible', 'On');      % show heading
    set(handles.MaxTtext, 'Visible', 'On');       % show heading for maxT
    set(handles.MaxT, 'Visible', 'On');           % make field visible
    set(handles.deltaTtext, 'Visible', 'On');     % show heading for deltaT
    set(handles.deltaT, 'Visible', 'On');         % make field visible
    set(handles.MaxT, 'String', SSMT)             % update SSMT result
    set(handles.deltaT, 'String', DeltaT)         % update deltaT
else             % steady state has not been reached, hide the data fields
    set(handles.SystemRes, 'Visible', 'Off');     % show heading
    set(handles.MaxTtext, 'Visible', 'Off');      % hide result text for time
    set(handles.deltaTtext, 'Visible', 'Off');    % show heading
end

if PickUp        % if PickUp has been set (end stage 2) display the results
    set(handles.PickUtext, 'Visible', 'On');      % show heading
    set(handles.PickUpR, 'Visible', 'On');        % make field visible
    set(handles.PickUpR, 'String', PickUp)        % update PickUp result
else             % pick up value not yet found, hide the data fields
    set(handles.PickUtext, 'Visible', 'Off');     % hide result text for distance
end

if TMSet >= 0    % if TMSet has been set (end stage 3) display the results

    set(handles.boundaryButton, 'Visible', 'Off');  % disable radio button

    % update best fit results
    set(handles.IDMTheading, 'Visible', 'On');    % show heading
    set(handles.PickUpText, 'Visible', 'On');     % show heading
    set(handles.PickUpS, 'Visible', 'On');        % make field visible
    set(handles.PickUpS, 'String', PickUpSet)     % update PickUpSet result
    set(handles.TMStext, 'Visible', 'On');        % show heading
    set(handles.TMS, 'Visible', 'On');            % make field visible
    set(handles.TMS, 'String', TMSet)             % update TMS result
    set(handles.CurveBtext, 'Visible', 'On');     % show heading
    set(handles.CurveB, 'Visible', 'On');         % make field visible
    set(handles.CurveB, 'String', curveBest);     % update curveBest result
    set(handles.Ucurve,'Enable','on');            % make user curve selection available

    if curveU ~= 1  % if preferred curve has been selected by user

        % update user results
        set(handles.UserHeading, 'Visible', 'On');   % show heading
        set(handles.PickUpTextU, 'Visible', 'On');   % show heading
        set(handles.PickUpU, 'Visible', 'On');       % make field visible
        set(handles.PickUpU, 'String', PickUpSet)    % update PickUpSet result
        set(handles.TMStextU, 'Visible', 'On');      % show heading
        set(handles.TMSU, 'Visible', 'On');          % make field visible
        set(handles.TMSU, 'String', TMSetUser)       % update TMSetUser result
        set(handles.CurveUtext, 'Visible', 'On');    % show heading
        set(handles.CurveU, 'Visible', 'On');        % make field visible
        set(handles.CurveU, 'String', curveUser);    % update curveUser result

```

```
    else             % user has not selected a preferred curve

        % hide user curve result text
        set(handles.UserHeading, 'Visible', 'Off'); % show heading
        set(handles.TMStextU, 'Visible', 'Off');     % hide result text
        set(handles.CurveUtext, 'Visible', 'Off');  % hide result text
        set(handles.PickUpTextU, 'Visible', 'Off'); % hide result text
        set(handles.TMSU, 'Visible', 'Off');         % make field invisible
        set(handles.CurveU, 'Visible', 'Off');       % make field invisible
        set(handles.PickUpU, 'Visible', 'Off');      % make field invisible

    end

else                 % if TMS has not yet been set

    % hide best curve result text
    set(handles.IDMTheading, 'Visible', 'Off'); % hide heading
    set(handles.TMStext, 'Visible', 'Off');     % hide result text
    set(handles.CurveBtext, 'Visible', 'Off');  % hide result text
    set(handles.PickUpText, 'Visible', 'Off');  % hide result text

    % hide user curve result text
    set(handles.UserHeading, 'Visible', 'Off'); % hide heading
    set(handles.TMStextU, 'Visible', 'Off');    % hide result text
    set(handles.CurveUtext, 'Visible', 'Off');  % hide result text
    set(handles.PickUpTextU, 'Visible', 'Off'); % hide result text

end
% end - gui_OutputFcn()



%% the following functions execute during creation of the gui interactive fields.

% --- Executes on key press with focus on configuration and none of its controls.
function configuration_KeyPressFcn(hObject, eventdata, handles)
% no action - automatically generated code
% end - configuration_KeyPressFcn()

% --- Executes during object creation, after setting all properties.
function DepthOfLay_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - DepthOfLay_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Separation_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Separation_CreateFcn()


% --- Executes during object creation, after setting all properties.
function xsection_CreateFcn(hObject, eventdata, handles)
```

```matlab
% set the default value for the cross-section dropdown box
set(hObject,'Value',8);
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
selection = get(hObject,'Value');     % extract default value for the system
% call function to convert selection to actual cross-section value
getXsection(selection);               % call the function to define cross section data
% end - xsection_CreateFcn()


% --- Executes during object creation, after setting all properties.
function configuration_CreateFcn(hObject, eventdata, handles)
% set the default value for the system configuration dropdown box
set(hObject,'Value',2);
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Initialise global variables that will be used by other functions
global  system;                   % cable system configuration
system = get(hObject,'Value');  % extract and global save default value for system
% end - configuration_CreateFcn()


% --- Executes during object creation, after setting all properties.
function condMat_CreateFcn(hObject, eventdata, handles)
set(hObject,'Value',2);   % set the default value for the conductor material
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Initialise global variables that will be used by other functions
global  conductor;      % integer set by user to determine if conductor is Cu or Al
conductor = get(hObject,'Value');% extract and global save default value for conductor
% end - condMat_CreateFcn()


% --- Executes during object creation, after setting all properties.
function XLPE_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - XLPE_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Shield_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Shield_CreateFcn()

```

```
% --- Executes during object creation, after setting all properties.
function PVC_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - PVC_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Atemp_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Atemp_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Gtemp_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Gtemp_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Bedding_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Bedding_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Load_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - Load_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Ucurve_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
global  curveU;          % user defined curve
curveU = get(hObject,'Value');      % extract default value of the user defined curve
% end - Ucurve_CreateFcn()
```

```


% --- Executes during object creation, after setting all properties.
function JointSelect_CreateFcn(hObject, eventdata, handles)
% set the default value for the joint selection dropdown box
set(hObject,'Value',1);
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Initialise global variables that will be used by other functions
global  Joint;                  % integer set by user to determine the Joint situation
Joint = get(hObject,'Value');   % extract value of the selected cable system
% end - JointSelect_CreateFcn()


% --- Executes during object creation, after setting all properties.
function BreakOp_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - BreakOp_CreateFcn()


% --- Executes during object creation, after setting all properties.
function safetyM_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - safetyM_CreateFcn()


% --- Executes during object creation, after setting all properties.
function Res_CreateFcn(hObject, eventdata, handles)
% set the default value for the cross-section dropdown box
set(hObject,'Value',1);
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Initialise global variables that will be used by other functions
global  resolution;         % integer that determines the size of each finite element
resolution = get(hObject,'Value');          % extract default value for the system
% end - Res_CreateFcn()


% --- Executes during object creation, after setting all properties.
function ageData_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - ageData_CreateFcn()

```

```

% --- Executes during object creation, after setting all properties.
function jointsData_CreateFcn(hObject, eventdata, handles)
% create object, automatically generated code
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% end - jointsData_CreateFcn()



%% the following functions execute when the user interacts with the gui data fields.

% this function executes when the StartButton is pressed (Run Simulation).
function StartButton_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  wait ...        % flag used to determine when the user inputs are complete
        Gtemp ...       % ground temp as specified by the user
        I ...           % value of load current as specified by the user
        Atemp ...       % air temp as specified by the user
        breakerOp ...   % time for circuit breaker to clear a fault after trip signal
        SafetyM ...     % safety margin as defined by the user
        resolution ...  % integer that determines the size of each finite element
        system ...      % cable system configuration
        conductor ...   % integer set by user to determine if conductor is Cu or Al
        xsection ...    % cross-section of conductor as specified by user
        Joint ...       % integer set by user to determine the Joint situation
        filename;       % used throughout gui for filename of image saves

% save global values to be used throughout the simulation
SafetyM = str2num(get(handles.safetyM,'string'));        % safety margin
breakerOp = str2num(get(handles.BreakOp,'string'))/1000;% breaker trip time
I = str2num(get(handles.Load,'string'));                % load current of system
Gtemp = str2num(get(handles.Gtemp,'string'));           % ambient ground temperature
Atemp = str2num(get(handles.Atemp,'string'));           % ambient air temperature

% create a filename that will be used by the snapshots and includes system information
filename = sprintf('L%0.0f R%0.0f I%0.0f C%0.0f X%0.0f J%0.0f ', ...    % text
    system,resolution,I,conductor,xsection*1e6,Joint);                  % variables

% save a copy of the gui figure window to preserve the settings that were used
hgexport(gcf, sprintf([filename '- 1) Simulation Start.png']), ...    % filename
    hgexport('factorystyle'), 'Format', 'png');                      % style of export

% define the new status string to be displayed at the bottom of the gui
StatusString = 'Simulaition running ... ';          % status to update to the user
set(handles.Status, 'String', StatusString)         % update status

wait = 0;                       % clear wait flag to allow Master script to progress

% disable the input variables, dropdown boxes and start button
set(handles.configuration,'Enable','off')
set(handles.DepthOfLay,'Enable','off')
set(handles.Bedding,'Enable','off')
set(handles.Separation,'Enable','off')
set(handles.condMat,'Enable','off')
set(handles.xsection,'Enable','off')
set(handles.XLPE,'Enable','off')
set(handles.Shield,'Enable','off')
```

```
set(handles.PVC,'Enable','off')
set(handles.Atemp,'Enable','off')
set(handles.Gtemp,'Enable','off')
set(handles.Load,'Enable','off')
set(handles.Ucurve,'Enable','off')
set(handles.BreakOp,'Enable','off')
set(handles.safetyM,'Enable','off')
set(handles.Res,'Enable','off')
set(handles.JointSelect,'Enable','off')
set(handles.StartButton,'Enable','off')
set(handles.jointsData,'Enable','off')
set(handles.ageData,'Enable','off')
% enable the stop button
set(handles.StopButton,'Enable','on');
% end - StartButton_Callback()


% this function executes when the Depth of Lay is modified by user.
function DepthOfLay_Callback(hObject, eventdata, handles)

% Initialise global variables that are shared between this and other functions
global  depth ...        % burial depth of cable (used to determine if air is shown)
        Layout;          % layout matrix containing different integer for each material

% read in value and if it is out of the range, colour the text red so user is aware
in1 = str2num(get(handles.DepthOfLay,'string'))/1000;        % read in value
if in1 >= 0 && in1 <= 2                                       % check if value is valid
    set(handles.DepthOfLay, 'ForegroundColor', [0,0,0]);     % text black
    depth = str2num(get(handles.DepthOfLay,'string'))/1000; % depth of cable lay
else                                                          % data not valid
    set(handles.DepthOfLay, 'ForegroundColor', [1,0,0]);     % make red invalid
end
checkValid(handles)          % call function to check if run button can be shown

% remove the fields for air temperature if the depth of lay means no air is in system
if Layout ~= 7          % if no entries are equal to 7 (no air)
    set(handles.text29, 'Visible', 'Off');                    % hide Atemp info
    set(handles.text32, 'Visible', 'Off');                    % hide Atemp info
    set(handles.Atemp, 'Visible', 'Off');                     % hide Atemp info
else                    % if no entries are equal to 7 (air)
   set(handles.text29, 'Visible', 'On');                      % make Atemp info visible
   set(handles.text32, 'Visible', 'On');                      % make Atemp info visible
   set(handles.Atemp, 'Visible', 'On');                       % make Atemp info visible
end
% end - DepthOfLay_Callback()


% this function executes when the cable separation distance is modified by user.
function Separation_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global separation;      % distance between conductors (when multiple conductors)

% read in value and if it is out of the range, colour the text red so user is aware
in2 = str2num(get(handles.Separation,'string'))/1000;        % read in value
if in2 >= 0 && in2 <= 0.2                                     % check if value is valid
    set(handles.Separation, 'ForegroundColor', [0,0,0]);     % text black
    separation = str2num(get(handles.Separation,'string'))/1000; % cable separation
else                                                          % data not valid
    set(handles.Separation, 'ForegroundColor', [1,0,0]);     % text red
end
```

```
checkValid(handles)          % call function to check if run button can be shown
% end - Separation_Callback()


% this function executes when the cable cross-section dropdown box is modified.
function xsection_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  Joint ...       % integer set by user to determine the Joint situation
        shield ...      % thickness of cable shield
        pvc ...         % thickness of pvc
        insul;          % insulation thickness of the cable

selection = get(hObject,'Value');      % extract default value for the system

% joint analysis is only available for cable cross-section of less than 400mm2
if selection <= 4 % less than 400mm2
    % enable joint configuration
    set(handles.JointText, 'Visible', 'On');     % show joint selection field
    set(handles.JointSelect, 'Visible', 'On');   % show joint text

else
    % disable joint configuration
    set(handles.JointSelect,'Value',1);          % force joint to 1 (no joint)
    set(handles.JointText, 'Visible', 'Off');    % hide joint selection field
    set(handles.JointSelect, 'Visible', 'Off'); % hide joint text
    hideJointFields(handles)

    % enable input functionality
    set(handles.condMat,'Enable','on'); % user can now change the conductor material
    set(handles.XLPE,'Enable','on');    % user can now change the XLPE thickness
    set(handles.Shield,'Enable','on');  % user can now change the Shield thickness
    set(handles.PVC,'Enable','on');     % user can now change the PVC

    % restore the values of the cable
    insul = str2num(get(handles.XLPE,'string'))/1000;      % thickness of XLPE
    shield = str2num(get(handles.Shield,'string'))/1000;   % thickness of Shield
    pvc = str2num(get(handles.PVC,'string'))/1000;         % thickness of PVC
end
Joint = get(handles.JointSelect,'Value');        % update Joint to what is configured

% call function to convert selection to actual cross-section value
getXsection(selection);                % call the function to define cross section data
axes(handles.axes3);                   % set active plot to axis 3
layoutMatrix();                        % update the graphical representation of system
checkValid(handles)                    % call function to check run button can be shown
% end - xsection_Callback()


% this function executes when the system configuration dropdown box is modified.
function configuration_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  system;         % cable system configuration

system = get(hObject,'Value');      % extract and save global value of selected system

% hide the data fields for separation unless required
if system == 1 || system == 2
    set(handles.Separation, 'Visible', 'Off');  % hide cable separation field
    set(handles.text6, 'Visible', 'Off');       % hide cable separation field
```

```
    set(handles.text9, 'Visible', 'Off');       % hide cable separation field
else
    set(handles.Separation, 'Visible', 'On');   % show cable separation field
    set(handles.text6, 'Visible', 'On');        % show cable separation field
    set(handles.text9, 'Visible', 'On');        % show cable separation field
end

axes(handles.axes3);            % set active plot to axis 3
layoutMatrix();                 % update the graphical representation of the system
checkValid(handles);            % call function to check if run button can be shown
% end - configuration_Callback()


% this function executes when the Stop button is pressed.
function StopButton_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  run ...         % flag used to determine if user has stopped the simulation
        StatusString;   % string that is output on gui to give status messages to user

run = 0;                                        % clear the run flag
set(handles.StopButton,'Enable','off');         % grey out stop button
set(handles.boundaryButton, 'Visible', 'Off');  % disable radio button

% update status for the user
StatusString = 'User stopped simulation. Close GUI and re-run Master file.';
set(handles.Status, 'String', StatusString) % update status on gui

% print out to command window
fprintf('Simulation stopped by user\n')     % output to command window
% end - StopButton_Callback()


% this function executes when the User closes the gui window.
function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  run ...         % flag used to determine if user has stopped the simulation
        wait;           % flag used to determine when the user inputs are complete

run = 0;                                % clear the run flag
wait = 0;                               % abort wait for user inputs
delete(hObject);                        % close gui figure
close all;                              % close all windows
fprintf('User closed GUI window\n')     % output to command window
% end - figure1_CloseRequestFcn()


% this function executes when the conductor material dropdown box is modified.
function condMat_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  conductor;      % integer set by user to determine if conductor is Cu or Al

conductor = get(hObject,'Value');     % extract value of the selected cable system
axes(handles.axes3);        % make active plot axis 3
layoutMatrix();             % update the graphical representation of the system
checkValid(handles)         % call function to check if run button can be shown
selection = get(handles.xsection,'Value');     % extract setting for cross-section
getXsection(selection);     % call function to re-determine cross-section resistance
% end - condMat_Callback()
```

```


% this function executes when the XLPE thickness is modified by user.
function XLPE_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  insul;          % insulation thickness of the cable

in4 = str2num(get(handles.XLPE,'string'))/1000;             % read in value
if in4 >= 0.001 && in4 <= 0.04                              % check if value is valid
    set(handles.XLPE, 'ForegroundColor', [0,0,0]);          % text black
    insul = str2num(get(handles.XLPE,'string'))/1000;       % thickness of XLPE
else                                                        % data is invalid
    set(handles.XLPE, 'ForegroundColor', [1,0,0]);          % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - XLPE_Callback()


% this function executes when the Shield thickness is modified by user.
function Shield_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global shield;          % thickness of cable shield

in5 = str2num(get(handles.Shield,'string'))/1000;           % read in value
if in5 >= 0 && in5 <= 0.02                                  % check if value is valid
    set(handles.Shield, 'ForegroundColor', [0,0,0]);        % text black
    shield = str2num(get(handles.Shield,'string'))/1000;    % thickness of Shield
else                                                        % data is invalid
    set(handles.Shield, 'ForegroundColor', [1,0,0]);        % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - Shield_Callback()


% this function executes when the PVC thickness is modified by user.
function PVC_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global pvc;             % thickness of pvc

in6 = str2num(get(handles.PVC,'string'))/1000;              % read in value
if in6 >= 0.001 && in6 <= 0.05                              % check if value is valid
    set(handles.PVC, 'ForegroundColor', [0,0,0]);           % text black
    pvc = str2num(get(handles.PVC,'string'))/1000;          % thickness of PVC
else                                                        % data is invalid
    set(handles.PVC, 'ForegroundColor', [1,0,0]);           % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - PVC_Callback()


% this function executes when the Air Temperature field is modified by user.
function Atemp_Callback(hObject, eventdata, handles)
in7 = str2num(get(handles.Atemp,'string'));                 % read in value
if in7 >= -40 && in7 <= 60                                  % check if value is valid
    set(handles.Atemp, 'ForegroundColor', [0,0,0]);         % text black
else                                                        % data is invalid
    set(handles.Atemp, 'ForegroundColor', [1,0,0]);         % text red
end
```

```
checkValid(handles)         % call function to check if run button can be shown
% end - Atemp_Callback()


% this function executes when the Ground Temperature field is modified by user.
function Gtemp_Callback(hObject, eventdata, handles)
in8 = str2num(get(handles.Gtemp,'string'));                 % read in value
if in8 >= -40 && in8 <= 60                                  % check if value is valid
    set(handles.Gtemp, 'ForegroundColor', [0,0,0]);         % text black
else                                                        % data is invalid
    set(handles.Gtemp, 'ForegroundColor', [1,0,0]);         % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - Gtemp_Callback()


% this function executes when the Bedding thickness field is modified by user.
function Bedding_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  bedding;        % thickness of bedding sand surrounding the cable
in9 = str2num(get(handles.Bedding,'string'))/1000;          % read in value
if in9 >= 0 && in9 <= 0.2                                   % check if value is valid
    set(handles.Bedding, 'ForegroundColor', [0,0,0]);       % text black
    bedding = str2num(get(handles.Bedding,'string'))/1000;  % thickness of bedding
else                                                        % data is invalid
    set(handles.Bedding, 'ForegroundColor', [1,0,0]);       % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - Bedding_Callback()


% this function executes when the Bedding thickness field is modified by user.
function Load_Callback(hObject, eventdata, handles)
in10 = str2num(get(handles.Load,'string'));                 % read in value
if in10 >= 1 && in10 <= 50e3                                % check if value is valid
    set(handles.Load, 'ForegroundColor', [0,0,0]);          % text black
else                                                        % data is invalid
    set(handles.Load, 'ForegroundColor', [1,0,0]);          % text red
end
checkValid(handles)         % call function to check if run button can be shown
% end - Load_Callback()


% this function executes when the user defined IDMT curve dropdown box is modified.
function Ucurve_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  curveU ...      % user defined curve
        TMSet ...       % time multiplier setting for best-fit curve
        TMSetUser ...   % time multiplier setting for user defined curve
        curveUser ...   % text string for user defined curve
        PickUpSet;      % pickup setting value considering safety margin and breakerOp

curveU = get(hObject,'Value');      % extract value of the selected cable system

if TMSet                            % if results exist for IDMT curve

    breakcurve();                                       % recalculate break curve
    set(handles.TMSU, 'String', TMSetUser)              % update TMSetUser result
    set(handles.CurveU, 'String', curveUser);           % update curveUser result
```

```

    if curveU == 1                  % if no curve selected by user
        % hide user curve result text
        set(handles.UserHeading, 'Visible', 'Off');     % show heading for user curve
        set(handles.TMStextU, 'Visible', 'Off');        % hide result text
        set(handles.CurveUtext, 'Visible', 'Off');      % hide result text
        set(handles.PickUpTextU, 'Visible', 'Off');     % hide result text
        set(handles.TMSU, 'Visible', 'Off');            % make field invisible
        set(handles.CurveU, 'Visible', 'Off');          % make field invisible
        set(handles.PickUpU, 'Visible', 'Off');         % make field invisible

    else                            % if curve selected by user
        % update user results
        set(handles.UserHeading, 'Visible', 'On');      % show heading
        set(handles.PickUpTextU, 'Visible', 'On');      % show heading
        set(handles.PickUpU, 'Visible', 'On');          % make field visible
        set(handles.TMStextU, 'Visible', 'On');         % show heading
        set(handles.TMSU, 'Visible', 'On');             % make field visible

        if TMSetUser == 0           % if no valid result has been found for user curve
            set(handles.PickUpU, 'String', 'Invalid Curve') % update result
            set(handles.TMSU, 'String', 'Invalid Curve')    % update result
        else                        % if valid result has been found for user curve
            set(handles.PickUpU, 'String', PickUpSet)   % update result
            set(handles.TMSU, 'String', TMSetUser)      % update result
        end

        set(handles.CurveUtext, 'Visible', 'On');       % show heading
        set(handles.CurveU, 'Visible', 'On');           % make field visible
        set(handles.CurveU, 'String', curveUser);       % update result

    end
end
% end - Ucurve_Callback()


% this function executes when the joint condition dropdown box is modified.
function JointSelect_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  Joint ...       % integer set by user to determine the Joint situation
        pvc ...         % thickness of pvc
        insul ...       % insulation thickness of the cable
        conductor ...   % integer set by user to determine if conductor is Cu or Al
        shield ...      % thickness of cable shield
        Rjoint;         % contact resistance value for cable joint

Joint = get(hObject,'Value');       % extract value of the selected cable system

if Joint == 1           % joint is not present, revert to normal configuration
    % enable input functionality
    set(handles.XLPE,'Enable','on');     % user can now change the XLPE thickness
    set(handles.Shield,'Enable','on');   % user can now change the Shield thickness
    set(handles.PVC,'Enable','on');      % user can now change the PVC
    set(handles.condMat,'Enable','on'); % user can now change the conductor material
    % restore the values of the cable
    insul = str2num(get(handles.XLPE,'string'))/1000;       % thickness of XLPE
    shield = str2num(get(handles.Shield,'string'))/1000;    % thickness of Shield
    pvc = str2num(get(handles.PVC,'string'))/1000;          % thickness of PVC
    Rjoint = [];                        % clear R joint (only required for Joint == 4)
    hideJointFields(handles)            % hide statistical fields (for Joint == 4)
```

```

elseif Joint <= 3       % if joint exists, Joint is 2 (good) or 3 (poor)

    % force dimensions to that of the cable joint
    insul = 10/1000;                   % new thickness of XLPE
    shield = 1/1000;                   % new thickness of Shield
    pvc = 10/1000;                     % new thickness of PVC
    conductor = 2;                     % set the conductor material to be aluminium
    set(handles.condMat,'Value',2);    % set the conductor material to be aluminium
    % disable input functionality of variables overridden by the joint in the system
    set(handles.XLPE,'Enable','off');   % user can no longer change XLPE thickness
    set(handles.Shield,'Enable','off'); % user can no longer change Shield thickness
    set(handles.PVC,'Enable','off');    % user can no longer change PVC thickness
    set(handles.condMat,'Enable','off');% user can no longer the conductor material
    Rjoint = [];                       % clear R joint (only required for Joint == 4)
    hideJointFields(handles)           % hide statistical fields (for Joint == 4)

else                    % if statistical analysis has been selected

    % force dimensions to that of the cable joint
    insul = 10/1000;                   % new thickness of XLPE
    shield = 1/1000;                   % new thickness of Shield
    pvc = 10/1000;                     % new thickness of PVC
    conductor = 2;                     % set the conductor material to be aluminium
    set(handles.condMat,'Value',2);    % set the conductor material to be aluminium
    % disable input functionality of variables overridden by the joint in the system
    set(handles.XLPE,'Enable','off');   % user can no longer change XLPE thickness
    set(handles.Shield,'Enable','off'); % user can no longer change Shield thickness
    set(handles.PVC,'Enable','off');    % user can no longer change PVC thickness
    set(handles.condMat,'Enable','off');% user can no longer the conductor material

    solveProbability(handles);   % call function to solve statistical joint information

    % made data fields visible
    set(handles.ageText, 'Visible', 'On');       % show age text
    set(handles.jointsText, 'Visible', 'On');    % show joint text
    set(handles.yearsText, 'Visible', 'On');     % show years text
    set(handles.probText, 'Visible', 'On');      % show probability text
    set(handles.resText, 'Visible', 'On');       % show resistance text
    set(handles.jointProb, 'Visible', 'On');     % show probability field
    set(handles.jointRes, 'Visible', 'On');      % show resistance field
    set(handles.statHeader, 'Visible', 'On');    % show joint heading
    set(handles.ageData, 'Visible', 'On');       % show age data field
    set(handles.jointsData, 'Visible', 'On');    % show joints data field

end

selection = get(handles.xsection,'Value');      % extract setting for cross-section

getXsection(selection);      % call function to convert to actual cross-section value
axes(handles.axes3);         % set active plot to axis 1
layoutMatrix();              % update the graphical representation of the system
checkValid(handles);         % call function to check if run button can be shown
% end - JointSelect_Callback()


% this function executes when the breaker operating time is modified by user.
function BreakOp_Callback(hObject, eventdata, handles)
in11 = str2num(get(handles.BreakOp,'string'))/1000;         % read in value
if in11 >= 0 && in11 <= 0.5                                 % check if value is valid
    set(handles.BreakOp, 'ForegroundColor', [0,0,0]);       % text black
```

```
else                                                        % data is invalid
    set(handles.BreakOp, 'ForegroundColor', [1,0,0]);       % text red
end
checkValid(handles);          % call function to check if run button can be shown
% end - BreakOp_Callback()


% this function executes when the safety margin is modified by user.
function safetyM_Callback(hObject, eventdata, handles)
in12 = str2num(get(handles.safetyM,'string'));          % read in value
if in12 >= 0 && in12 <= 99                              % check if value is valid
    set(handles.safetyM, 'ForegroundColor', [0,0,0]);   % text black
else                                                    % data is invalid
    set(handles.safetyM, 'ForegroundColor', [1,0,0]);   % text red
end
checkValid(handles);         % call function to check if run button can be shown
% end - safetyM_Callback()


% this function executes when the resolution dropdown box is modified.
function Res_Callback(hObject, eventdata, handles)

% Initialise global variables that will be used by other functions
global  resolution;          % integer that determines the size of each finite element

resolution = get(hObject,'Value');      % save the dropdown box value of resolution
axes(handles.axes3);         % set active plot to axis 3
layoutMatrix();              % update the graphical representation of the system
checkValid(handles);         % call function to check if run button can be shown
% end - Res_Callback()


% this function executes when the age of the system is modified by user.
function ageData_Callback(hObject, eventdata, handles)
in13 = str2num(get(handles.ageData,'string'));              % read in value
if in13 >= 1 && in13 <= 70                                  % check if value is valid
    set(handles.ageData, 'ForegroundColor', [0,0,0]);       % text black
else                                                        % data is invalid
    set(handles.ageData, 'ForegroundColor', [1,0,0]);       % text red
end
solveProbability(handles);    % call function to solve statistical joint information
% end - ageData_Callback()


% this function executes when the number of joints is modified by user.
function jointsData_Callback(hObject, eventdata, handles)
in12 = str2num(get(handles.jointsData,'string'));           % read in value
if in12 >= 1 && in12 <= 1000                                % check if value is valid
    set(handles.jointsData, 'ForegroundColor', [0,0,0]);    % text black
else                                                        % data is invalid
    set(handles.jointsData, 'ForegroundColor', [1,0,0]);    % text red
end
solveProbability(handles);    % call function to solve statistical joint information
% end - jointsData_Callback()


% this function executes when the radio button is toggled by user.
function boundaryButton_Callback(hObject, eventdata, handles)
% end - boundaryButton_Callback()


```

```
%% the following functions are in-line functions that are used throughout gui.m

% function to check if all the user inputs are within the required values and the
% Start button can therefore be displayed, if not, make it inactive
function checkValid(handles)

% Initialise global variables that will be used by other functions
global  Layout ...       % layout matrix containing different integer for each mate
        wait ...         % flag used to determine when the user inputs are complete
        StatusString;    % string that is output on gui to give status messages to user

% read in values from the user input fields
in1 = str2double(get(handles.DepthOfLay,'string'))/1000;    % depth of cable lay (m)
in2 = str2double(get(handles.Separation,'string'))/1000;    % cable separation (m)
in3 = str2double(get(handles.safetyM,'string'));            % safety margin
in4 = str2double(get(handles.XLPE,'string'))/1000;          % thickness of XLPE (m)
in5 = str2double(get(handles.Shield,'string'))/1000;        % thickness of Shield (m)
in6 = str2double(get(handles.PVC,'string'))/1000;           % thickness of PVC (m)
in7 = str2double(get(handles.Atemp,'string'));              % Ambient Air Temp.
in8 = str2double(get(handles.Gtemp,'string'));              % Ambient Ground Temp.
in9 = str2double(get(handles.Bedding,'string'))/1000;       % bedding thickness (m)
in10 = str2double(get(handles.Load,'string'));              % Load current (A)
in11 = str2double(get(handles.BreakOp,'string'))/1000;      % Breaker open time (s)

% check if all user specified values are within the correct range
if in1 >= 0 && in1 <= 1 && in2 >= 0 && in2 <= 0.2 && in3 >= 0 && in3 <= 99  && ...
       in4 >= 0.001 && in4 <= 0.04 && in5 >= 0 && in5 <= 0.02 && in6 >= 0.001 && ...
       in6 <= 0.05 && in8 >= -40 && in7 <= 60 && in7 >= -40 && in8 <= 60 && ...
       in9 >= 0 && in9 <= 0.2 && in10 >= 1 && in10 <= 50e3 && in11 >= 0 && in11 <= 0.5

    if wait == 1            % if system is waiting to start, update the layout plot
        layoutMatrix()      % update the graphical representation of the system
    end

    if Layout ~= 1          % if no entries in layout map to the conductor, alert user
        set(handles.StartButton,'Enable','off');     % make start button unavailable
        % update status at the bottom of the gui to alert user
        StatusString = 'Warning: Resolution too low for selected cable cross-section';
        set(handles.Status, 'String', StatusString) % update gui status

    elseif wait == 1        % else, check if user is still configuring the layout
        set(handles.StartButton,'Enable','on');      % make start button available
        axes(handles.axes3);                         % set active plot to axis 3
        StatusString = 'Simulation ready to run.';  % update status for the user
        set(handles.Status, 'String', StatusString) % update gui status

    end

else                        % if any of the user inputs are not within range

    set(handles.StartButton,'Enable','off');        % disable start pushbutton
    % update status at the bottom of the gui to alert user
    StatusString = 'Warning: Please adjust red text to within the valid range';
    set(handles.Status, 'String', StatusString)     % update gui status

end
% end - checkValid()
```

```matlab
% this function is called to assign information on the cross section of the conductor
% and the joints of the cable system
function getXsection(selection)

% Initialise global variables that will be used by other functions
global  conductor ...   % integer set by user to determine if conductor is Cu or Al
        xsection ...    % cross-section of conductor as specified by user
        R20R90 ...      % conductor resistance at 20 and 90 deg used to interpolate
        Joint ...       % integer set by user to determine joint status (1 = no joint)
        JointL;         % length of joint, if applicable

if Joint == 1   % if analysing cable only

    % determine conductor properties
    data = xlsread('Parameters','conductor'); % load the data set from .xls
    xsection = data(selection,5)/1e6;         % conductor cross-section (m2)
    % determine resistance values depending on cross-section and material
    if conductor == 1                         % if conductor is copper
        R20R90 = data(selection,1:2);         % Resistance/km points for interpolation
    else                                      % conductor is aluminium
        R20R90 = data(selection,3:4);         % Resistance/km points for interpolation
    end

else            % cable joint needs to be considered, override cable properties

    data = xlsread('Parameters','joint');     % load the data set
    JointD = data(1) / 1000;                  % diameter of cable joint (m)
    xsection = pi * (JointD/2) ^ 2;           % cross section area of joint diameter
    JointL = data(2) / 1000;                  % length of cable joint (m)

end
% end - getXsection()


% this function is called to hide the statistical information about the joint analysis
function hideJointFields(handles)
    set(handles.ageText, 'Visible', 'Off');      % hide age text
    set(handles.jointsText, 'Visible', 'Off');   % hide joint text
    set(handles.yearsText, 'Visible', 'Off');    % hide years text
    set(handles.probText, 'Visible', 'Off');     % hide probability text
    set(handles.resText, 'Visible', 'Off');      % hide resistance text
    set(handles.jointProb, 'Visible', 'Off');    % hide probability field
    set(handles.jointRes, 'Visible', 'Off');     % hide resistance field
    set(handles.statHeader, 'Visible', 'Off');   % hide joint heading
    set(handles.ageData, 'Visible', 'Off');      % hide age data field
    set(handles.jointsData, 'Visible', 'Off');   % hide joints data field
% end - hideJointFields()

% this function is called to solve the statistical information about the joint analysis
% using the 2-parameter Weibull distribution
function solveProbability(handles)

% Initialise global variables that will be used by other functions
global  Rjoint;         % contact resistance value for cable joint

    % read user defined values
    year = str2num(get(handles.ageData,'string'));          % age of system
    joints = str2num(get(handles.jointsData,'string'));     % no. of joints in system

    % Weibull values taken from Mehairjan  (2010, p. 73)
    X = 1:200;                          % data range of distribution
```

```
    A = 52.3925;                    % scale parameter of Weibull distribution
    B = 4.4791;                     % shape parameter of Weibull distribution

    f = wblpdf(X,A,B);              % probability density function - Weibull 2P
    R = exp(-(X/A).^B);             % reliability curve
    F = f./R;                       % failure rate function of failure

    ProbOne = F(year);              % probability of one failure
    ProbAll = ProbOne*joints;       % probability of one failure within system

    data = xlsread('Parameters','joint');    % load the data set for joint resistance
    jointData = data(3:4) / 1e6;             % gather data points for good & bad joints

    % linearly interpolate/extrapolate the expected resistance of the joint between
    % good and bad joint contact resistance values. Probability values are 0 to 1
    % which map to a good joint and a bad joint
    Rjoint = interp1([0 1],jointData,ProbAll,'linear','extrap');

    set(handles.jointProb, 'String', roundn(ProbAll,-5))  % update failure probability
    set(handles.jointRes, 'String', roundn(Rjoint*1e6,-1))% update joint resistance
% end - solveProbability()

% ------------------------------------ End - gui.m ---------------------------------- %
```

```matlab
%% layoutMatrix.m ---------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependent IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       12/09/14    Initial release to supervisor for partial review
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function creates a matrix, the same size as the resolution of the F.E. system.
% Each entry to the matrix is represented by an integer which maps to different
% materials. This allows data on material properties to be maintained and updated
% against the integer in the F.E. location, rather than re-defining the F.E. material
% every iteration. This function is also responsible for the 'colour by numbers' plot
% that allows the user to see the system layout prior to starting the simulation.
% ------------------------------------------------------------------------------------

function layoutMatrix()

% Initialise global variables that will be used by other functions
global  rows ...        % number of rows in system  matrices
        cols ...        % number of columns in system matrices
        x ...           % array containing all x axis values for cable cross-section
        y ...           % array containing all y axis values for cable cross-section
        Layout ...      % layout matrix containing different integer for each material
        cabrad ...      % radius of the conductor of the cable
        insul ...       % insulation thickness of the cable
        k ...           % vertical and horizontal step size (delta x or y)
        width ...       % width of simulation cross-section
        height ...      % height of simulation cross-section
        depth ...       % burial depth of cable (used to determine if air is shown)
        shield ...      % thickness of cable shield
        pvc ...         % thickness of pvc
        system ...      % cable system configuration
        separation ...  % distance between conductors (when multiple conductors)
        dt ...          % simulation time step at each iteration
        Days ...        % No. of days simulation should run for to reach steady state
        conductor ...   % integer set by user to determine if conductor is Cu or Al
        bedding ...     % thickness of bedding sand surrounding the cable
        xsection ...    % cross-section of conductor as specified by user
        resolution;     % integer that determines the size of each finite element

%% Define the simulation time steps, F.E. size and tolerance based on user defined
```

```
% resolution. The timestep must be reduced for higher resolution to ensure lambda
% values do not exceed a value that makes the system unstable. The total size of the
% cross section is also dependent on the resolution to improve the time required to
% solve the simulation. Also, because the system is smaller, the days simulated can be
% reduced as the system will reach steady state sooner.
switch resolution           % use switch for the value of resolution

    case 1                  % user has selected low resolution
        dt = 0.2;           % simulation timestep (s)
        k = 0.01;           % horizontal step size (m)
        height = 1;         % height of system (m)
        Days = 5;           % number of days simulated to achieve steady state temp.

    case 2                  % user has selected mid resolution
        dt = 0.03;          % simulation timestep (s)
        k = 0.004;          % horizontal step size (m)
        height = 0.8;       % height of system (m)
        Days = 5;           % number of days simulated to achieve steady state temp.

    case 3                  % user has selected high resolution
        dt = 0.008;         % simulation timestep (s)
        k = 0.002;          % horizontal step size (m)
        height = 0.6;       % height of system (m)
        Days = 4;           % number of days simulated to achieve steady state temp.

end


%% define the simulation space for the cross-section simulation
% also define all global and local variables used to reference locations in the system
h = k;                      % set vertical step size to be the same as the horizontal
width = height;             % width of system (m)
x = 0:k:width;              % create an array of all the x axis values
y = 0:h:height;             % create an array of all the y axis values
rows = height/h + 1;        % number of vertical blocks
cols = width/k + 1;         % number of horizontal blocks
mid = ((rows)/2);           % mid point of the matrix
cabrad = sqrt(xsection/pi); % conductor radius


%% generate a matrix that represents the different materials in the system
% this places an integer against each F.E. to represent the material located at that
% F.E. The material at each location will depend on the system selected by the user
% and the size of the components within the system.
% The materials that map to each number are:
%   1 -> Conductor       2 -> XLPE           3 -> Shield          4 -> PVC
%   5 -> Bedding sand    6 -> Soil           7 -> Air

if system == 1                      % if user has specified a single phase system

    Layout = ones(rows,cols)*6;     % default all points to soil

    % loop through each F.E. point and assign it to the relevant material depending on
    % the location within the system. A 'for' loop is not too slow as this is only
    % executed when the user is specifying the system.
    for i = 1:rows                  % for each row of simulation space
        for j = 1:cols              % for each column of simulation space

            % calculate radius from centre to the current point using Pythagoras
            rad = sqrt((abs(mid-i)*k)^2 + (abs(mid-j)*k)^2);

```

```
            % calculate the outer radius of the cable
            cabR = cabrad+insul+shield+pvc-k/2;    % outer radius of cable

            % determine if point is in conductor zone
            if (rad <= (cabrad-k/2))
                Layout(i,j) = 1;     % conductor

            % or, is point in insulator zone
            elseif (rad <= (cabrad+insul-k/2))
                Layout(i,j) = 2;     % insulator

            % or is point in shield zone
            elseif (rad <= (cabrad+insul+shield-k/2))
                Layout(i,j) = 3;     % shield

            % or is point in pvc zone
            elseif (rad <= (cabrad+insul+shield+pvc-k/2))
                Layout(i,j) = 4;     % pvc

            % or is point above ground in air zone
            elseif ((height/2 - i*k) > (depth-k))
                Layout(i,j) = 7;     % air

            % or is point in bedding sand zone
            elseif ( (abs(width/2 - j*k) < cabR + bedding) && ...  % within horizontal
                     (abs(height/2 - i*k) < cabR + bedding)     )  % & within vertical

                % if user has specified the system has bedding sand
                if bedding > 0          % if user has specified the system has bedding
                    Layout(i,j) = 5;    % bedding sand
                end

            end
        end
    end


elseif system == 2 || system == 4       % if user has specified a trefoil system

    Layout = ones(rows,cols)*6;         % default all points to soil

    % determine central points for each cable core, the layout for one centralised
    % trefoil cable is executed first. If the system is for a dual trefoil system, the
    % layout of the cable is then shifted and moved to represent each of the trefoils.
    L = (cabrad+insul+shield)*2;        % distance between circle centres
    h = sqrt(L^2 - (L/2)^2);            % height of triangle made by circles
    h1 = L/sqrt(3);                     % centre line to top circle origin
    h2 = h - h1;                        % centre line to bottom circle origin
    cabR = (h1+L/2)+shield+pvc-k/2;     % outer radius of the cable
    P1v = h1/(k);                       % central vertical point of phase 1
    P1h = 0;                            % central horizontal point of phase 1
    P2v = -h2/(k);                      % central vertical point of phase 2
    P2h = (cabrad+insul+shield)/(k);    % central horizontal point of phase 2
    P3v = -h2/(k);                      % central vertical point of phase 3
    P3h = -(cabrad+insul+shield)/(k);   % central horizontal point of phase 3

    % loop through each F.E. point and assign it to the relevant material depending on
    % the location within the system. A 'for' loop is not too slow as this is only
    % executed when the user is specifying the system.
    for i = 1:rows     % for each row
        for j = 1:cols     % for each column
```

```
            % calculate radius from centre to the current point using Pythagoras
            rad = sqrt((abs(mid-i)*k)^2 + (abs(mid-j)*k)^2);

            % if point is in internal PVC zone (PVC used as filler between cores)
            if (rad <= ((h1+L/2)-k/2))
                Layout(i,j) = 4;     % PVC

            % if point is in shield zone (external cable shield)
            elseif (rad <= ((h1+L/2)+shield-k/2))
                Layout(i,j) = 3;     % shield.

            % if point is in external PVC zone
            elseif (rad <= cabR)
                Layout(i,j) = 4;     % PVC

            % if point is above ground in air zone
            elseif ((height/2 - i*k) > (depth-k))
                Layout(i,j) = 7;     % air

            % if point is in bedding sand zone
            elseif ( (abs(width/2 - j*k) < cabR + bedding) && (abs(height/2 - i*k) < cabR + bedding) )

                % if user has specified the system has bedding sand
                if bedding > 0          % bedding sand specified
                    Layout(i,j) = 5;    % bedding sand
                end

            end

            % Determine insulation, conductor and shield associated with top core.
            % Calculate the radius of point with reference to the centre of the top
            % core using Pythagoras.
            rad = sqrt((abs(mid-P1v-i)*k)^2 + (abs(mid-P1h-j)*k)^2);

            % determine if point is in conductor zone
            if (rad <= (cabrad-k/2))
                Layout(i,j) = 1;     % conductor

            % or, is point in insulator zone
            elseif (rad <= (cabrad+insul-k/2))
                Layout(i,j) = 2;     % insulator

            % or, is point in xlpe shield zone
            elseif (rad <= (cabrad+insul+shield-k/2))
                Layout(i,j) = 3;     % internal shield
            end

            % Determine insulation, conductor and shield associated with left core.
            % Calculate the radius of point with reference to the centre of the left
            % core using Pythagoras.
            rad = sqrt((abs(mid-P2v-i)*k)^2 + (abs(mid-P2h-j)*k)^2);

            % determine if point is in conductor zone
            if (rad <= (cabrad-k/2))
                Layout(i,j) = 1;     % conductor

            % or, is point in insulator zone
            elseif (rad <= (cabrad+insul-k/2))
                Layout(i,j) = 2;     % insulator
```

```

            % or, is point in xlpe shield zone
            elseif (rad <= (cabrad+insul+shield-k/2))
                Layout(i,j) = 3;    % internal shield
            end

            % Determine insulation, conductor and shield associated with right core.
            % Calculate the radius of point with reference to the centre of the right
            % core using Pythagoras.
            rad = sqrt((abs(mid-P3v-i)*k)^2 + (abs(mid-P3h-j)*k)^2);

            % determine if point is in conductor zone
            if (rad <= (cabrad-k/2))
                Layout(i,j) = 1;    % conductor

            % or, is point in insulator zone
            elseif (rad <= (cabrad+insul-k/2))
                Layout(i,j) = 2;    % insulator

            % or, is point in xlpe shield zone
            elseif (rad <= (cabrad+insul+shield-k/2))
                Layout(i,j) = 3;    % internal shield
            end
        end
    end

    % if system is a dual trefoil system, use the above layout as a stencil for each
    % of the trefoil cables
    if system == 4              % dual trefoil system

        Lsave = Layout;         % save the Layout from above

        % determine the outer diameter of trefoil cable
        outerRad = h1 + L/2 + shield + pvc;

        % determine the number of columns required to shift based on the radius of
        % each cable and the separation between the cables as defined by the user
        shift = ceil((outerRad + separation) / k);  % columns to shift
        centre = ceil(mid);         % centre position of layout matrix

        % create left half of layout matrix by shifting original layout matrix left
        a = (shift);                % start of shift column values
        b = (shift+centre-1);       % end of shift column values
        left = a:b;                 % array of shift column values
        % new left half of the Layout matrix
        Layout(1:end,1:centre) = Lsave(1:end,left);

        % create left half of layout matrix by shifting original layout matrix right
        a = (centre-shift+2);       % start of shift column values
        b = (cols-shift+1);         % end of shift column values
        right = a:b;                % array of shift column values
        % new right half of the Layout matrix
        Layout(1:end,centre+1:end) = Lsave(1:end,right);

    end


elseif system == 3                  % if user has specified a single phase system

    Layout = ones(rows,cols)*6;     % default all points to soil

```

```
    Bed = zeros(rows,cols);          % default all points to zero for bedding matrix

    % determine central points of cables
    L = (cabrad+insul+shield+pvc)*2+separation; % distance between cable centres
    cabR = cabrad+insul+shield+pvc-k/2;         % cable outer radius
    P1h = L/k;                       % central horizontal point of phase 1
    P2h = 0;                         % central horizontal point of phase 2
    P3h = -L/k;                      % central horizontal point of phase 3

    % loop through each F.E. point and assign it to the relevant material depending on
    % the location within the system. A 'for' loop is not too slow as this is only
    % executed when the user is specifying the system. Also, loop through each of the
    % cable centre locations to allow creation of each of the cables.
    for Ph = [P1h P2h P3h]   % repeat layout for each cable centre
        for i = 1:rows       % for each row
            for j = 1:cols   % for each column

                % calculate radius of point from cable centre using Pythagoras
                rad = sqrt((abs(mid-i)*k)^2 + (abs(mid-Ph-j)*k)^2);

                % determine if point is in conductor zone
                if (rad <= (cabrad-k/2))
                    Layout(i,j) = 1;     % conductor

                % or, is point in insulator zone
                elseif (rad <= (cabrad+insul-k/2))
                    Layout(i,j) = 2;     % insulator

                % or is point in shield zone
                elseif (rad <= (cabrad+insul+shield-k/2))
                    Layout(i,j) = 3;     % shield

                % or is point in pvc zone
                elseif (rad <= (cabrad+insul+shield+pvc-k/2))
                    Layout(i,j) = 4;     % pvc

                % or is point above ground and not part of cable, then it is air zone
                elseif ((height/2 - i*k) > (depth-k)) &&  Layout(i,j) > 4
                    Layout(i,j) = 7;     % air

                % create a matrix of ones for all points within bedding sand zone
                elseif ( (abs(width/2 - j*k) < cabR + L + bedding) ... % in horizontal
                        && (abs(height/2 - i*k) < cabR + bedding) )    % & in vertical
                    Bed(i,j) = 1;        % bedding sand
                end   % end if
            end       % end for (cols)
        end           % end for (rows)
    end               % end for (centres)

    % where bedding has been solved to 1 in Bed matrix, replace all Layout values
    % currently set to be solid with number representing bedding
    Layout((Layout == 6) & (Bed == 1) & (bedding > 0)) = 5;

end                   % end if (system type)

%% plot the colour by numbers to show the user how they have configured the system
% specify the RBG values to be mapped to the numbers within Layout
cmap = [
    192/255 192/255 192/255     % aluminium
    255/255 255/255 255/255     % xlpe
    204/255 102/255   0/255     % copper shield
```

```
     32/255  32/255  32/255     % pvc
    255/255 255/255 153/255     % bedding
    218/255 192/255 133/255     % soil
    153/255 204/255 255/255     % air
    ];

if conductor == 1               % overwrite aluminium RBG if conductor is copper
    cmap(1,:) = [204/255 102/255   0/255];       % copper
end

% remove the colour for air if the depth of lay means no air is shown in the layout
if Layout ~= 7                  % if no entries are equal to 7 (air)
   cmap = cmap(1:6,:);          % trim off bottom row for air
end

% plot the layout of the cable system
colormap(cmap);         % force the colours to be those defined by cmap above
imagesc(x,y,Layout);    % plot the colours as an image
axis square             % ensure axis is square
title('System configuration') % title
xlabel('Height (m)')    % x axis label
ylabel('Width (m)')     % y axis label

% ------------------------------- End - layoutMatrix.m ----------------------------- %
```

```
%% materialProperties.m ---------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependant IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       06/09/14    Initial release to supervisor for partial review
% 1.1       30/09/14    Updated to include statistical analysis of cable joint
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function uses the layout matrix to define the thermal diffusivity properties of
% each Finite Element. These values change depending, not only on the material
% represented by the F.E. but also by depending on the temperature of the F.E.
% The alpha matrix, Amat, can then be used by tempCalc.m to determine the rate at
% which heat is transferred between neighbouring F.E.
% -------------------------------------------------------------------------------------

function materialProperties()

% Initialise global variables shared between MATLAB files
global  c1 ...          % specific heat capacity of conductor material (J/(g.K))
        r1 ...          % mass density of conductor material (g/m3)
        Rpm ...         % resistance per meter of cable
        Layout ...      % layout matrix containing different integer for each material
        conductor ...   % integer set by user to determine if conductor is Cu or Al
        Amat ...        % matrix containing Alpha (diffusivity) values of each F.E.
        TempMat ...     % temperature matrix containing temperature of each F.E.
        R20R90 ...      % conductor resistance at 20 and 90 deg used to interpolate
        Joint ...       % integer set by user to determine joint status (1 = no joint)
        JointL ...      % length of joint, if applicable
        kBed ...        % lookup value from Properties for bedding thermal conductivity
        cBed ...        % lookup value from Properties for bedding specific heat
        rBed ...        % lookup value from Properties for bedding mass density
        kSoil ...       % lookup value from Properties for soil thermal conductivity
        cSoil ...       % lookup value from Properties for soil bedding specific heat
        rSoil ...       % lookup value from Properties for soil mass density
        Rjoint;         % contact resistance value for cable joint

% determine the resistance of the conductor material at the current temperature
if Joint == 1  % if user has configured system to have no joints
    maxT = max(max(TempMat));  % maximum temp in the system
    T = [20 90];     % temp values for cable resistance variance at different temps
    % interpolate to get the resistance value of the conductor at maxT
```

```
    Rpm = interp1(T,R20R90,maxT,'linear','extrap') / 1000;

% for joints, use the specified resistance values found by (Fournier & Amyon, 2001)
elseif Joint == 2                % healthy joint

    if Rjoint                    % skip if Rjoint has been defined already
    else                         % if not, read in data from .xls file
        data = xlsread('Parameters','joint');    % load the data set
        Rjoint = data(3) / 1e6; % resistance value for healthy joint (microOhms)
        Rpm = Rjoint / JointL;  % equivalent Rpm if considered to be 1m long
    end

elseif Joint == 3                % unhealthy joint

    if Rjoint                    % skip if Rjoint has been defined already
    else                         % if not, read in data from .xls file
        data = xlsread('Parameters','joint');    % load the data set
        Rjoint = data(4) / 1e6; % resistance value for healthy joint (microOhms)
        Rpm = Rjoint / JointL;  % equivalent Rpm if considered to be 1m long
    end

elseif Joint == 4                % statistical analysis of joint joint

    Rpm = Rjoint / JointL;       % equivalent Rpm if considered to be 1m long

end

%% determine the alpha value of each point in the FE Matrix

% conductor materials (Layout = 1)
Tk = [ 25 125 225 ]; % reference temperature for interpolation of k
if conductor == 1 % copper
    k = [ 401 400 398 ];     % thermal conductivity values that map to Tk (W/(m.K)
    c1 = 0.385;              % specific heat capacity  J/(g.K)
    r1 = 8940e3;             % mass density g/m3
else % aluminium
    k = [ 205 215 250 ];     % thermal conductivity values at Tk (W/(m.K)
    c1 = 0.897;              % specific heat capacity  J/(g.K)
    r1 = 2712e3;             % mass density g/m3
end
% interpolation to get temperature specific values
K1 = interp1(Tk,k,TempMat,'linear','extrap');
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A1 = (Layout == 1) .* K1/ (c1*r1); % solve Alpha value of each conductor F.E.


% insulation materials XLPE (Layout = 2)
Tk = [ 19 20 55 90 91 ];    % reference temperature for interpolation of k
k = [ 0.223 0.223 0.267 0.280 0.280 ];  % thermal conductivity values at Tk (W/(m.K)
Tc = [19 20 40 60 70 90 91]; % reference temperature for interpolation of c
c =  [2.0 2.0 2.2 3.0 3.0 4.0 4.0] ;  % specific heat capacity maps to Tc J/(g.K)
r = 929e3;             % mass density g/m3
% interpolation to get temperature specific values
K2 = interp1(Tk,k,TempMat,'linear','extrap');
C2 = interp1(Tc,c,TempMat,'linear','extrap');
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A2 = (Layout == 2) .* K2./ (C2*r); % Alpha value of all XLPE Finite Elements



% shield materials (Layout = 3)
```

```
Tk = [ 25 125 225 ]; % reference temperature for interpolation of k
k = [ 205 215 250 ]; % thermal conductivity values at Tk (W/(m.K)
c = 0.385;           % specific heat capacity  J/(g.K)
r = 8940e3;          % mass density g/m3
% interpolation to get temperature specific values
K3 = interp1(Tk,k,TempMat,'linear','extrap');
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A3 = (Layout == 3) .* K3/ (c*r); % Alpha value of all Shield Finite Elements


% PVC materials (Layout = 4)
k = 0.19;     % thermal conductivities (W/(m.K))
c = 1.005;        % specific heat capacity  J/(g.K)
r = 801e3;    % mass density g/m3
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A4 = (Layout == 4) * k/ (c*r); % Alpha value of all PVC Finite Elements


% Bedding materials (Layout = 5)
if kBed                    % skip if kBed has been defined already
else                       % if not, read in data from .xls file
    data = xlsread('Parameters','soil');     % load the data set
    kBed = data(1);                          % thermal conductivities (W/(m.K))
    cBed = data(2);                          % specific heat capacity  J/(g.K)
    rBed = data(3);                          % mass density g/m3
end
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A5 = (Layout == 5) * kBed/ (cBed*rBed); % Alpha value of all Bedding Finite Elements


% Soil materials (Layout = 6)
if kSoil                   % skip if kSoil has been defined already
else                       % if not, read in data from .xls file
    data = xlsread('Parameters','soil');     % load the data set
    kSoil = data(4);                         % thermal conductivities (W/(m.K))
    cSoil = data(5);                         % specific heat capacity  J/(g.K)
    rSoil = data(6);                         % mass density g/m3
end
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A6 = (Layout == 6) * kSoil/ (cSoil*rSoil); % Alpha value of all Soil Finite Elements


% Air (Layout = 7) may not be used, depends on the buried depth of cable
k = 0.024;    % thermal conductivities (W/(m.K))
c = 1.005;    % specific heat capacity  J/(g.K)
r = 1.2e3;    % mass density g/m3
% thermal diffusivity m2/s (Layout used to only keep relevant values)
A7 = (Layout == 7) * k/ (c*r); % Alpha value of all Air Finite Elements


% combine all Alphas to get the system's Alpha matrix
Amat = A1 + A2 + A3 + A4 + A5 + A6 + A7;

% ---------------------------- End - materialProperties.m -------------------------- %
```

```
%% tempPlot.m ------------------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependent IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       12/09/14    Initial release to supervisor for partial review
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function solves the thermal matrix for each future time step as the simulation
% advances through time.
% This file also contains in-line function getQandL() which dynamically determines the
% heating and Lambda properties of the system as it changes with temperature.
% ------------------------------------------------------------------------------------

function [T,time] = tempCalc(T,dt,I)

% Initialise global variables that will be used by other functions
global  stepsize ...     % used to determine the time between plot updates in the gui
        run ...          % flag used to determine if user has stopped the simulation
        SimStep ...      % integer that reflects which stage the simulation is up to
        StatusString... % string that is output on gui to give status messages to user
        TempMat ...      % temperature matrix containing temperature of each F.E.
        St2Percent ...   % used to output the percentage of stage 2 completed
        PickupTmax ...   % maximum time relay will count for. i.e. to trip at pick-up
        simTime ...      % value for the simulated time
        Iplot ...        % value of the current being simulated
        Qmat ...         % F.E heat contribution due to current flow (Ohm heating)
        Plot ...         % flag used to inform gui when a new thermal plot is ready
        Days ...         % No. of days simulation should run for to reach steady state
        Tupdate;         % Temp that when exceeded, will update the material properties

% set local variables from global data to retain global data
Iplot = I;  % save globally for the plot title

% call the getQandL function which generate the matrices of Q (heating) and L, lambda
% values for each individual finite element. These vary with temperature so it is
% important to update regularly to ensure the system is as accurate as possible.
% Lambda values are produced in 4 different matrices for matrix calculation of each
% future T matrix.
[Lu Ld Ll Lr] = getQandL(dt,I);     % solve Qmat and the heat transfer matrix, Lambda


```

```
%% solve difference equation using Gauss-Seidel iterative approach

% initialise local timers and counters
iteration = 0;          % counter for the number of iterations
time = 0;               % reset local timer to zero
Step3Plot = 0;          % flag for outputting thermal values during fault simulations

while run == 1          % always loop until break or user stops

    % create shifted matrices for faster calculation of the future T values. This
    % will allow all future F.E. T values to be calculated with one matrix command,
    % rather than using for loops to solve for the next time step. The matrix must be
    % shifted up, down, left and right as well as maintained to allow for the
    % calculation to work. Because the outside edges are boundary conditions, these do
    % not need to be calculated and the matrix edges can be cut so the size of what
    % will be used is 2 less than the size of the original matrix T.
    T0 = T(2:end-1,2:end-1);    % no shift (previous T value)
    T1 = T(1:end-2,2:end-1);    % shift down (previous upper T value)
    T2 = T(3:end,2:end-1);      % shift up (previous lower T value)
    T3 = T(2:end-1,1:end-2);    % shift right (previous left T value)
    T4 = T(2:end-1,3:end);      % shift left (previous right T value)

    % use shifted matrices to perform faster calculation of diff equations
    T(2:end-1,2:end-1) = ...        % internal values of the new T matrix =
        Lu.*T1 + ...                % lambda with upper F.E. * upper temperature
        Ld.*T2 + ...                % lambda with lower F.E. * lower temperature
        Ll.*T3 + ...                % lambda with left F.E. * left temperature
        Lr.*T4 + ...                % lambda with right F.E. * right temperature
        (1-(Lu+Ld+Ll+Lr)).*T0 ...   % 1 - lambda to all F.E. * previous temperature
        + Qmat;                     % addition of ohmic heating from current flow


    %% check if a steady state temperature has been found
    % check if system has been simulated for more than the defined no. of days and
    % user has not aborted (run == 1)
    if ( time > Days*(60*60*24) ) && run == 1

        Plot = 1;           % plot flag used to trigger gui plot update
        TempMat = T;        % global matrix to plot temp. profile in gui window
        simTime = time/60;  % global value of simulation time to update in gui window

        if SimStep == 1     % if operating in the first stage of the simulation
            % status update to the user to show that step 1 is complete
            StatusString = 'Simulation running ... Step 1/3: Solving for system''s steady state temperature profile ... Done';
        end

        gui();              % call graphical user interface (gui) to update status
        Tupdate = [];       % clear Tupdate to have no value
        break               % exit from the while loop

    end


    %% check the recommended operating temp temperature has been exceeded

    maxT = max(max(T));     % maximum temp in the system

    % check if system can not handle the system load current during the first stage,
    % determining of the steady state temperature profile.
    if (maxT > 90 && run == 1 && SimStep == 1)
```

```
        run = 0;            % clear the run flag to abort the simulation

        StatusString = ...  % update status to the user to say system has failed
            'Simulation stopped. System can not handle the load current.';

        % also print out to the command window
        fprintf('Simulation stopped. System can not handle the load current.\n');

        gui();              % call graphical user interface (gui) to update status
        break               % exit from the while loop

    end


    % check if long term or short term cable ratings have been exceeded during
    % simulation stage 3. This is used to determine the trip time at the simulated
    % current value.
    if ((maxT >= 90 && time > 5) || maxT >= 250) && run == 1 && SimStep == 3

        Plot = 1;           % plot flag used to trigger gui plot update
        TempMat = T;        % global matrix to plot temp. profile in gui window
        simTime = time/60;  % global value of simulation time to update in gui window

        gui();              % call graphical user interface (gui) to update status
        Tupdate = [];       % clear Tupdate to have no value
        break               % exit from the while loop
    end

    % When determining the pick up current and hence the first point of the trip
    % curve, break after the time defined as the maximum pickup time, which is a
    % theoretical maximum measurement of a protection relay. If, the max time has been
    % exceeded and simulation is currently performing step 2 or the simulation,
    if SimStep == 2 && time >= PickupTmax

        Plot = 1;           % plot flag used to trigger gui plot update
        TempMat = T;        % global matrix to plot temp. profile in gui window
        simTime = time/60;  % global value of simulation time to update in gui window

        gui();              % call graphical user interface (gui) to update status
        Tupdate = [];       % clear Tupdate to have no value
        break               % exit from the while loop
    end


    %% periodically update the output plots and material properties

    % Every time the iteration counter is a multiple of the stepsize as defined in the
    % master file or during initialisation, is half of the step size to smooth the
    % initial temperature jump, do the following:
    % - update the temperature plots
    % - update the progress to the user at the bottom of the gui window
    % - re-solve all the thermal properties of the system as the temperature changes
    if (rem(iteration,stepsize) == 0 || iteration == stepsize/2) && run == 1

        if SimStep == 1     % if operating in the first stage of the simulation

            % progress is the simulation time / the maximum simulation time, days
            Progress = time / (Days*(60*60*24)) * 100;  % progress percentage
            % update the status to be displayed at the bottom of the gui window
            StatusString = sprintf('Simulation running ... Step 1/3: Solving for
```

```
system''s steady state temperature profile ... %0.1f %%',Progress);

        elseif SimStep == 2 % if operating in the second stage of the simulation

            % progress percentage iteration/max iterations = n/5 = n*20 percent
            Progress = St2Percent + time / PickupTmax * 20;
            % update the status to be displayed at the bottom of the gui window
            StatusString = sprintf('Simulation running ... Step 2/3: Optimising system''s pick-up value ... %0.1f %% (may be less)',Progress);

        end

        Plot = 1;           % plot flag used to trigger gui plot update
        TempMat = T;        % global matrix to plot temp. profile in gui window
        simTime = time/60;  % global value of simulation time to update in gui window
        gui();              % call graphical user interface (gui) to update status

        % update the material Properties of each F.E. with respect to the F.E. temp
        materialProperties();       % call materialProperties function (below)

        % update the lambda values of each of the F.E. and also the internal heating
        % matrix as resistance and therefore the power will change with temperature
        [Lu Ld Ll Lr] = getQandL(dt,I);     % solve for Qmat and Lambda

    end

    % as the maximum system temperature increases by 1 deg, update the material
    % properties to ensure the simulation operates with relevant material properties.
    % 0.1 degree for stage 3 of the plot because system changes very quickly
    if Tupdate     % if Tupdate has been  set

        % update the thermal plots more frequently during stage 3 so user can see the
        % progress, especially at high fault currents when the simulation solves in a
        % few seconds
        if maxT > Tupdate       % check if the update temp threshold has been exceeded

            if SimStep == 3     % if operating in step 3 of the simulation

                Tupdate = Tupdate + 0.1;     % set next temp. threshold for update
                Step3Plot = Step3Plot + 1;  % increment counter

                % update plot everytime the max temp increases by 5 degrees
                if Step3Plot >= 50  % if 50 increments, or 5 degrees

                    Plot = 1;           % plot flag used to trigger gui plot update
                    TempMat = T;        % global matrix to plot temp. profile in gui
                    simTime = time/60;  % global value of simulation time for gui
                    gui();              % call gui to update status
                    Step3Plot = 0;      % clear counter for plot update (stage 3 only)

                end

            else                        % operating in step 2 or 3 of the simulation

                Tupdate = Tupdate + 1;  % set next temp. threshold for update

            end

            % update the material Properties of each F.E. with respect to the temp.
            materialProperties();       % call materialProperties function (below)

```

```
            % update the lambda values of each F.E. and also the internal heating
            % matrix as resistance and therefore the power will change with temp
            [Lu Ld Ll Lr] = getQandL(dt,I);     % solve for Qmat and Lambda

        end

    else    % if Tupdate has not been set, initialise it as the maximum system temp.
        Tupdate = maxT;    % initialise Tupdate
    end


    %% maintain the simulation time (seconds since start)
    time = (time + dt);         % global time is incremented by dt, every 'while' loop
    iteration = iteration + 1;  % keep count of the iterations

end         % end of the while loop


%% Calculate the heat generated in one phase of the cable system
% This in-line function getQandL() which dynamically determines the heating and Lambda
% properties of the system as it changes with temperature.
function [Lu Ld Ll Lr] = getQandL(dt,I)

% Initialise global variables that will be used by other functions
global  Amat ...        % matrix containing Alpha (diffusivity) values of each F.E.
        c1 ...          % specific heat capacity of conductor material (J/(g.K))
        r1 ...          % mass density of conductor material (g/m3)
        Rpm ...         % resistance per meter of cable
        Layout ...      % layout matrix containing different integer for each material
        k ...           % vertical and horizontal step size (delta x or y)
        Qmat ...        % F.E heat contribution due to current flow (Ohm heating)
        system;         % cable system configuration

%% create matrix Qmat representing temp rise of each F.E.
% Heat is only generated inside the conductor finite elements. The amount of power is
% determined using the resistance and current. The temperature increase is dependent
% on the material properties of the conductor material.

% all measurement assume that the length of the cable or F.E. is 1 metre
len = 1;                            % length = 1m

% determine the total power generated in the system, this varies depending on the
% cable configuration
if system == 1                      % for single phase system
    P = I^2*Rpm*len;                % power per metre of cable

elseif system == 2 || system == 3   % for three phase system
    P1 = I^2*Rpm*len;               % power per metre of cable of one phase
    P = P1*3;                       % 3 cores carrying equal load, 3x power

elseif system == 4                  % for parallel three phase system
    I = I/2;                        % current is shared because parallel conductors
    P1 = I^2*Rpm*len;               % power per metre of only one phase
    P = P1*6;                       % 6 cores carrying equal load, 6x power
end

% the total power is shared amongst the finite elements that are mapped to conductor
pieces = sum(sum(Layout == 1));     % number of conductor finite elements
p = P / pieces;                     % power generated in each F.E. (W)

% the work done on each finite element is determined by the time the power is exerted
```

```matlab
q = p*dt;                                % work done in each F.E since last time sample (J)

% the temperature increase due to the work done on the F.E. is dependent on the
% specific heat of the material and the mass of the material within the F.E. which is
% dependent on the density of the material.
v = k^2*len;                             % volume of each F.E. (m3)
m = v*r1;                                % mass of each conductor F.E. (g)
qdot = q/(c1*m);                         % temperature increase of each F.E., delta T

% create a matrix representing the delta T, or qdot, of each F.E., this is done by
% substituting the heat contribution qdot into each conductor F.E.
Qmat = (Layout(2:end-1,2:end-1) == 1) * qdot; % substitute qdot into conductor F.E.


%% build Lambda matrices for matrix multiplication
% calculate lambda, should be << 0.5 to ensure F.E. remains stable
Lambda = Amat*(dt/k^2);               % calculate Lambda value for each F.E.
Lmax = max(max(Lambda));              % find the maximum value within the Lambda matrix
if (Lmax > 0.25)                     % check if lambda is outside a stable range
    fprintf('Lambda %.3f is too big \n',Lmax) % printout warning, system is not stable
end

% shift matrix and cut irrelevant edges of matrix. The lambda values are averaged
% with the F.E. that shares the boundary to ensure reasonable values are used when
% different materials share a F.E. boundary.
L0 = Lambda(2:end-1,2:end-1);            % F.E.'s value of Lambda
L1 = Lambda(1:end-2,2:end-1);            % upper F.E.'s value of Lambda
L2 = Lambda(3:end,2:end-1);              % lower F.E.'s value of Lambda
L3 = Lambda(2:end-1,1:end-2);            % left F.E.'s value of Lambda
L4 = Lambda(2:end-1,3:end);              % right F.E.'s value of Lambda
Lu = (L0+L1)./2;                         % average between local and upper Lambda
Ld = (L0+L2)./2;                         % average between local and lower Lambda
Ll = (L0+L3)./2;                         % average between local and left Lambda
Lr = (L0+L4)./2;                         % average between local and right Lambda

% end in-line function - getQandL()

% --------------------------------- End - tempCalc.m ------------------------------- %
```

```matlab
%% tempPlot.m ------------------------------------------------------------------------
%
% Author        Greg Nagel - 0061025127
% Project       System dependant IDMT overcurrent settings for underground cables
%
% This file has been created by Greg Nagel for a final year research project to be
% submitted to the University of Southern Queensland for courses, ENG4111/4112. It is
% theoretical only and should not be used as the basis for decisions made on actual
% power system applications.
%
% Release   Date        Comments
% 1.0       09/9/14     Initial release to supervisor for partial review
% 1.1       30/9/14     Include radio button for removing material boundaries from plot
% 2.0       05/10/14    Code finalised and prepared for submission
%
% This file is a function require to support the file Master.m as part of the
% simulation software developed by Greg Nagel to be used as a guideline for
% determining the IDMT protection settings for underground power cables.
%
% Supporting files required the same directory as Master.m
% breakcurve.m              function to fit and plot protection curves
% gui.fig                   graphical file for user interface
% gui.m                     function to execute graphical user interface
% layoutMatrix.m            function to create colour by numbers matrix
% materialProperties.m      function to dynamically update material properties
% tempCalc.m                function to solve the simulation through time
% tempPlot.m                function to output plot of thermal profile
%
% This function creates the coloured thermal plot shown in the graphical user
% interface. The plots also include circles to show the barriers between each of the
% material properties.
% -------------------------------------------------------------------------------------

function tempPlot()

% Initialise global variables shared between MATLAB files
global  x ...           % array containing all x axis values for cable cross-section
        y ...           % array containing all y axis values for cable cross-section
        Layout ...      % layout matrix containing different integer for each material
        cabrad ...      % radius of the conductor of the cable
        insul ...       % insulation thickness of the cable
        k ...           % vertical and horizontal step size (delta x or y)
        width ...       % width of simulation cross-section
        height ...      % height of simulation cross-section
        shield ...      % thickness of cable shield
        pvc ...         % thickness of pvc
        system ...      % cable system configuration
        TmaxSave ...    % contains fault temperatures and times for stage 1 temp plot
        TFmaxSave ...   % contains fault temperatures and times for stage 2 temp plot
        SimStep ...     % integer that reflects which stage the simulation is up to
        loops ...       % incremented to count the number of iterations
        separation ...  % distance between conductors (when multiple conductors)
        simTime ...     % value for the simulated time
        Iplot ...       % value of the current being simulated
        TempMat ...     % temperature matrix containing temperature of each F.E.
        Button;         % radio button status for showing the boundary circles on plot


% set local variables from global data to retain global data
T = TempMat;                % matrix of F.E. temp. values for use within this function
mins = rem(simTime,60);     % remainder of simulation time hours in minutes
```

```
hours = floor(simTime/60);  % simulation time rounded down to the nearest hour

colormap('default');        % clear the colours in colormap from layout plot

% find the maximum temperature within the cable (conductor, insulator, shield and PVC)
Tmax = max(max(T(Layout <= 4)));

% invert T and trim by one row for correct visual representation this is required as
% the matrix is calculated upside down compared to the layout of the system.
Tflip = [flipud(T(1:end-1,:)) ; T(end,:)];  % flip upside-down function

%% plot the colour profile of the temperature value of each finite element
pcolor(x,y,Tflip);              % plot temperature distribution
shading flat                    % shade each square as a single colour
axis square                     % force the axis to be square
colorbar                        % show temperature colour scale to the side of plot
ylabel('height of system (m)')  % label the y axis of the colour plot
xlabel('width of system (m)')   % label the x axis of the colour plot
title(sprintf( ...              % title of the plot with current and simulation time
 'Temperature profile after Time = %.0fh %.0fm at current = %.0f A',hours,mins,Iplot))
hold on                         % hold plot data so circle boundaries can be added

%% add circles for the boundaries of the conductor materials

% find centre point of the plot to reference the origins of the circles when plotting
xc = width/2;                       % horizontal centre location
yc = height/2;                      % vertical centre location

% create an array of radian values to be used to solve the x,y points of the circles
ang = 0:0.01:2*pi;                  % array of radian values from 0 - 2 pi


if system == 1 && Button == 1       % if user has specified a single phase system

    % add circle for outside boundary of conductor
    rad1 = cabrad;                  % radius of circle
    xp=rad1*cos(ang);               % x values of circle points
    yp=rad1*sin(ang);               % y values of circle points
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with circle origin at xc,yc

    % add circle for outside boundary of insulator
    rad2 = cabrad+insul;            % radius of circle
    xp=rad2*cos(ang);               % x values of circle points
    yp=rad2*sin(ang);               % y values of circle points
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with circle origin at xc,yc

    % add circle for outside boundary of shield
    rad3 = cabrad+insul+shield;     % radius of circle
    xp=rad3*cos(ang);               % x values of circle points
    yp=rad3*sin(ang);               % y values of circle points
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with circle origin at xc,yc

    % add circle for outside boundary of pvc
    rad4 = cabrad+insul+shield+pvc;   % radius of circle
    xp=rad4*cos(ang);               % x values of circle points
    yp=rad4*sin(ang);               % y values of circle points
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with circle origin at xc,yc


elseif system == 2 && Button == 1   % if user has specified a single trefoil system

```

```
    % determine central points for cables using Pythagoras' theorem
    L = (cabrad+insul+shield)*2;     % distance between trefoil circle centres
    h = sqrt(L^2 - (L/2)^2);         % height of triangle made by trefoil circles
    h1 = L/sqrt(3);                  % distance from centre to top circle origin
    h2 = h - h1;                     % distance from centre to bottom circle origin

    % add circle for the inside of shield
    rad1 = h1+L/2;                   % radius of circle
    xp=rad1*cos(ang);                % x values of circle points
    yp=rad1*sin(ang);                % y values of circle points
    plot(xc+xp,yc+yp,'w');           % plot the x,y points with circle origin at xc,yc

    % add circle for outside of shield
    rad2 = h1+L/2+shield;            % radius of circle
    xp=rad2*cos(ang);                % x values of circle points
    yp=rad2*sin(ang);                % y values of circle points
    plot(xc+xp,yc+yp,'w');           % plot the x,y points with circle origin at xc,yc

    % add circle for outside of pvc
    rad3 = h1+L/2+shield+pvc;        % radius of circle
    xp=rad3*cos(ang);                % x values of circle points
    yp=rad3*sin(ang);                % y values of circle points
    plot(xc+xp,yc+yp,'w');           % plot the x,y points with circle origin at xc,yc

    % add circle for the conductor of each cable core
    rad4 = cabrad;                   % radius of circle
    xp=rad4*cos(ang);                % x values of circle points
    yp=rad4*sin(ang);                % y values of circle points
    % add circle for the conductor of top core
    plot(xc+xp,yc+yp+h1,'w');       % plot the x,y points with origin of top core
    % add circle for the conductor of right core
    plot(xc+xp+L/2,yc+yp-h2,'w');   % plot the x,y points with origin of right core
    % add circle for the conductor of left core
    plot(xc+xp-L/2,yc+yp-h2,'w');   % plot the x,y points with origin of left core

    % add circle for the insulation of each cable core
    rad5 = cabrad+insul;             % radius of circle
    xp=rad5*cos(ang);                % x values of circle points
    yp=rad5*sin(ang);                % y values of circle points
    % add circle for the insulation of top core
    plot(xc+xp,yc+yp+h1,'w');        % plot the x,y points with origin of top core
    % add circle for the insulation of right core
    plot(xc+xp+L/2,yc+yp-h2,'w');   % plot the x,y points with origin of right core
    % add circle for the insulation of left core
    plot(xc+xp-L/2,yc+yp-h2,'w');   % plot the x,y points with origin of left core

    % add circle for the individual shields of each cable core
    rad6 = cabrad+insul+shield;      % radius of circle
    xp=rad6*cos(ang);                % x values of circle points
    yp=rad6*sin(ang);                % y values of circle points
    % add circle for the shield of top core
    plot(xc+xp,yc+yp+h1,'w');        % plot the x,y points with origin of top core
    % add circle for the shield of right core
    plot(xc+xp+L/2,yc+yp-h2,'w');   % plot the x,y points with origin of right core
    % add circle for the shield of left core
    plot(xc+xp-L/2,yc+yp-h2,'w');   % plot the x,y points with origin of left core


elseif system == 3 && Button == 1   % if user has specified 3 single-phase cable system
    % determine distance between cable centres
    L = (cabrad+insul+shield+pvc)*2+separation; % distance between cable centres
```

```

    % add circle for outside of conductor of the three cables
    rad1 = cabrad;                  % radius of circle
    xp=rad1*cos(ang);               % x values of circle points
    yp=rad1*sin(ang);               % y values of circle points
    % add circle for the outside of the conductor of the middle cable
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with origin of centre cable
    % add circle for the outside of the conductor of the right cable
    plot(xc+xp+L,yc+yp,'w');        % plot the x,y points with origin of right cable
    % add circle for the outside of the conductor of the left cable
    plot(xc+xp-L,yc+yp,'w');        % plot the x,y points with origin of left cable

    % add circle for outside of the insulation of the three cables
    rad2 = cabrad+insul;            % radius of circle
    xp=rad2*cos(ang);               % x values of circle points
    yp=rad2*sin(ang);               % y values of circle points
    % add circle for the outside of the insulation of the middle cable
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with origin of centre cable
    % add circle for the outside of the insulation of the right cable
    plot(xc+xp+L,yc+yp,'w');        % plot the x,y points with origin of right cable
    % add circle for the outside of the insulation of the left cable
    plot(xc+xp-L,yc+yp,'w');        % plot the x,y points with origin of left cable

    % add circle for outside of the shield of the three cables
    rad3 = cabrad+insul+shield;     % radius of circle
    xp=rad3*cos(ang);               % x values of circle points
    yp=rad3*sin(ang);               % y values of circle points
    % add circle for the outside of the shield of the middle cable
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with origin of centre cable
    % add circle for the outside of the shield of the right cable
    plot(xc+xp+L,yc+yp,'w');        % plot the x,y points with origin of right cable
    % add circle for the outside of the shield of the left cable
    plot(xc+xp-L,yc+yp,'w');        % plot the x,y points with origin of left cable

    % add circle for outside of the PVC of the three cables
    rad4 = cabrad+insul+shield+pvc; % radius of circle
    xp=rad4*cos(ang);               % x values of circle points
    yp=rad4*sin(ang);               % y values of circle points
    % add circle for the outside of the PVC of the middle cable
    plot(xc+xp,yc+yp,'w');          % plot the x,y points with origin of centre cable
    % add circle for the outside of the PVC of the right cable
    plot(xc+xp+L,yc+yp,'w');        % plot the x,y points with origin of right cable
    % add circle for the outside of the PVC of the left cable
    plot(xc+xp-L,yc+yp,'w');        % plot the x,y points with origin of left cable

elseif system == 4 && Button == 1   % if user has specified parallel trefoil system

    % determine central points for cables using Pythagoras' theorem
    L = (cabrad+insul+shield)*2;    % distance between trefoil circle centres
    h = sqrt(L^2 - (L/2)^2);        % height of triangle made by trefoil circles
    h1 = L/sqrt(3);                 % distance from centre to top circle origin
    h2 = h - h1;                    % distance from centre to bottom circle origin

    % determine the distance each cable will need to be shifted left or right from the
    % centre. This is dependant on the outer diameter of the cable and the separation
    % between the cables as defined by the user.
    shift = h1 + L/2 + shield + pvc + separation - k/2;

    % add circle for the inside of shield of each trefoil
    rad1 = h1+L/2;                  % radius of circle
    xp=rad1*cos(ang);               % x values of circle points
```

```
    yp=rad1*sin(ang);                % y values of circle points
    plot(xc+xp+shift,yc+yp,'w');     % plot the x,y points with origin of right trefoil
    plot(xc+xp-shift,yc+yp,'w');     % plot the x,y points with origin of left trefoil

    % add circle for outside of shield of each trefoil
    rad2 = h1+L/2+shield;            % radius of circle
    xp=rad2*cos(ang);                % x values of circle points
    yp=rad2*sin(ang);                % y values of circle points
    plot(xc+xp+shift,yc+yp,'w');     % plot the x,y points with origin of right trefoil
    plot(xc+xp-shift,yc+yp,'w');     % plot the x,y points with origin of left trefoil

    % add circle for outside of pvc of each trefoil
    rad3 = h1+L/2+shield+pvc;        % radius of circle
    xp=rad3*cos(ang);                % x values of circle points
    yp=rad3*sin(ang);                % y values of circle points
    plot(xc+xp+shift,yc+yp,'w');     % plot the x,y points with origin of right trefoil
    plot(xc+xp-shift,yc+yp,'w');     % plot the x,y points with origin of left trefoil

    % add circles for each of the conductors within the two trefoils
    rad4 = cabrad;                       % radius of circle
    xp=rad4*cos(ang);                    % x values of circle points
    yp=rad4*sin(ang);                    % y values of circle points
    % add circles for  each of the top trefoil conductors
    plot(xc+xp+shift,yc+yp+h1,'w');      % plot x,y, origin top core, right trefoil
    plot(xc+xp-shift,yc+yp+h1,'w');      % plot x,y, origin top core, left trefoil
    % add circles for  each of the right trefoil conductors
    plot(xc+xp+L/2+shift,yc+yp-h2,'w'); % plot x,y, origin right core, right trefoil
    plot(xc+xp+L/2-shift,yc+yp-h2,'w'); % plot x,y, origin right core, left trefoil
    % add circles for  each of the left trefoil conductors
    plot(xc+xp-L/2+shift,yc+yp-h2,'w'); % plot x,y, origin left core, right trefoil
    plot(xc+xp-L/2-shift,yc+yp-h2,'w'); % plot x,y, origin left core, left trefoil

    % add circles for the XLPE insulation of each core within the two trefoils
    rad5 = cabrad+insul;                 % radius of circle
    xp=rad5*cos(ang);                    % x values of circle points
    yp=rad5*sin(ang);                    % y values of circle points
    % add circles for  each of the top trefoil cores
    plot(xc+xp+shift,yc+yp+h1,'w');      % plot x,y, origin top core, right trefoil
    plot(xc+xp-shift,yc+yp+h1,'w');      % plot x,y, origin top core, left trefoil
    % add circles for  each of the right trefoil cores
    plot(xc+xp+L/2+shift,yc+yp-h2,'w'); % plot x,y, origin right core, right trefoil
    plot(xc+xp+L/2-shift,yc+yp-h2,'w'); % plot x,y, origin right core, left trefoil
    % add circles for  each of the left trefoil cores
    plot(xc+xp-L/2+shift,yc+yp-h2,'w'); % plot x,y, origin left core, right trefoil
    plot(xc+xp-L/2-shift,yc+yp-h2,'w'); % plot x,y, origin left core, left trefoil

    % add circles for the individual shields of each core within the two trefoils
    rad6 = cabrad+insul+shield;          % radius of circle
    xp=rad6*cos(ang);                    % x values of circle points
    yp=rad6*sin(ang);                    % y values of circle points
    % add circles for  each of the top trefoil cores
    plot(xc+xp+shift,yc+yp+h1,'w');      % plot x,y, origin top core, right trefoil
    plot(xc+xp-shift,yc+yp+h1,'w');      % plot x,y, origin top core, left trefoil
    % add circles for  each of the right trefoil cores
    plot(xc+xp+L/2+shift,yc+yp-h2,'w'); % plot x,y, origin right core, right trefoil
    plot(xc+xp+L/2-shift,yc+yp-h2,'w'); % plot x,y, origin right core, left trefoil
    % add circles for  each of the left trefoil cores
    plot(xc+xp-L/2+shift,yc+yp-h2,'w'); % plot x,y, origin left core, right trefoil
    plot(xc+xp-L/2-shift,yc+yp-h2,'w'); % plot x,y, origin left core, left trefoil

end
```

```

hold off                         % release plot data so plot can be overwritten

% record the number of times this function has been called as the number of loops
loops = loops + 1;               % increment the value of loops

% save the values of maximum temp and simulation time for the bottom temp vs. time
% plot that will be updated as the simulation is running
if SimStep == 1      % if operating in stage 1 of the simulation (steady state heating)
    % save the orignal values to be plotted during stage 1 and stage 2
    TmaxSave(loops,1:2) = [Tmax simTime];

else                 % operating in stage 2/3 of the simulation (overload/fault)
    % don't overwrite the original values, save as TFmax for plots in stage 2 and 3
    TFmaxSave(loops-length(TmaxSave),1:2) = [Tmax simTime];

end

% --------------------------------- End - tempPlot.m ------------------------------- %
```