MASTER

Insulated aluminium sections exposed to fire

Thermal and mechanical finite element modelling of protected aluminium and steel sections exposed to fire loading, a comparison between columns and beams

van der Wurff, R.M.

Award date:
2019

Insulated aluminium sections exposed to fire

GRADUATION THESIS – Structural design TU/e

Thermal and mechanical finite element modelling of protected aluminium and steel sections exposed to fire loading, a comparison between columns and beams.

Name:	Renée van der Wurff
Adres:	Lijsterbesstraat 111 5616 LE Eindhoven The Netherlands
Email:	rmwurff@gmail.com
Phone:	+31 6 104 99 456
Identification:	0811984 – A2019.278
Date:	22/10/2019

Graduation supervision committee:

Prof. dr. ir. J. Maljaars
Eindhoven University of Technology

Dr. Ir. H. Hofmeyer
Eindhoven University of Technology

Ir. F. Pawiroredjo
Bayards B.V.

PREFACE

This is the graduation thesis for completion of the master phase of the specialization Structural Design in the master track Architecture, Building and Planning of Eindhoven University of Technology. The thesis goes in-depth on the thermal and mechanical behaviour of a protected aluminium beam under fire load in comparison to that of a similar column as to warrant full scale beam tests as prescribed in EN13381 to determine the thermal properties of insulating materials when working with aluminium members. A full overview is achieved through methodical finite element analysis of both aluminium and steel sections which resulted in a project of more than 9000 lines of code.

SUMMARY

New insulating materials to be used with steel must be tested to determine their thermal properties according to NEN-EN 13381, which prescribes twelve unloaded columns and two full-scale loaded beam tests. During tests time and temperatures of gas, surfaces and cavities are measured. In tests with loaded sections deflection limits describe failure. With the use of Fourier differential equation and the heat equation, and inputting the test data plus densities and specific heat, the thermal conductivity of the insulation can be expressed as a function of temperature with additional linear regression analyses.

The necessity of the full-scale loaded beam tests in EN13381 is due to the fact that steel is subject to larger deformations before failure. Due to this the insulation layer around the cross-section can be damaged and result in a more rapid heating of the beam. The question is however, if the same can be said in the case of aluminium cross-section in combination with insulation.

Eurocode 3 and 9 describes simplified equations, assuming the thermal conductivity to be infinite and thus the temperature constant over the cross-section. The thermal and mechanical material properties at elevated temperatures in the Eurocode are based on steady state tests, however literature argues that transient state tests are more appropriate due to creep, overaging and annealing. To describe the stress-strain relation at elevated temperatures the Ramberg-Osgood equation is commonly used. Creep strains can be described using the Dorn-Harmathy model or be implicitly incorporated for aluminium by adjusting the stress-strain relation.

Typically a beam is subject to a three-sided fire, incurring a thermal gradient over the cross-sectional height between the exposed and ambient sides of the beam. This can affect both thermal and mechanical properties. The thermal gradient causes a distribution of the strength and stiffness, causing a shift of the neutral axis. Additionally, lengthwise thermal expansion differs between these sides, causing a thermal bowing effect. Strain is thus comprised out of elastic part, thermal expansion part and creep part.

To evaluate the behaviour of (protected) steel and aluminium sections, a thermal analysis is followed by a mechanical analysis is performed within finite element environment Abaqus. Approximately a hundred scenarios were considered, ranging from a column exposed to elevated temperatures from all sides, to beam facing a fire from three sides, and an integrated beam with one exposed side. The thermal analysis also includes an approach to tackling intumescent paints, fibre blanket insulation and an evaluation of the thermal effects of different floor systems. The mechanical analysis includes both a look at a simply supported beam under an evenly distributed load and when subject to a four point bending test.

The results show that as to be expected, the thermal gradient in uninsulated is much lower than for insulated section. In addition, the same can be said for aluminium section in comparison to steel, considering the larger thermal conductivity, this fits with conventional understanding. Overall it can be concluded that insulation has a tremendous effect on the temperature increase over time and the implementation of insulation and floor system on a beam is determining for the temperature distribution in the cross-section. For aluminium the effect appears to cause the thermal gradient over the cross-

section to become more linear, while steel has an inherently larger gradient than aluminium given the fact that it has a lower thermal conductivity.

Comparing the strain development between column and loaded beams for steel confirms that steel loaded beam sections showcase significant sagging before failure. The strain in the case of an insulated IPE section in combination with a lightweight floor shows clear deviation from $400°C$ onwards before failure at circa 600 degrees. In contrast, for aluminium, the slope of the strain is similar up until failure.

This leads to the conclusion that the deformation of a protected aluminium beam exposed to a fire load does **not** differ to any great extent from that of a similar column in such a manner that the protective insulation layer may be damaged prior to failure, and the heating of the beam would be affected. Following the results in chapter 7, there is a positive argument for the omission of full scale loaded beam tests for fire testing with new insulation materials in combination with aluminium. Considering the limit values in EN 13381 and the temperature from which the strain of the beam deviates from the column, to omit the beam test an additional safety margin of $25°C$ on the critical temperature for insulated, loaded structures is a recommended. To absolve the need for the loaded aluminium beam test completely however, additional testing is advised to determine if the model fits with an actual fire test.

CONTENT

NOMENCLATURE

Abbreviations

FEM Finite element method

ULS Ultimate limit state

Symbols

$f_{0,2}, f_y$ stress at 2‰ strain [N/mm^2]

f_u ultimate stress of aluminium [N/mm^2]

E_{mod} Young's modulus [N/mm^2]

L Span of the specimen [m]

d distance between extreme fibres of member [mm]

D, δ deflection [mm]

q or h' heat flux

k, λ thermal conductivity

k_{sh} shadow effects

A/V section factor, area over volume

∇ Laplace operator

α_c heat transfer coefficient for convection, in W/m^{2}K

θ_g, T_g gas temperature from nominal fire curve, in °C

θ_m surface temperature of the member following from the material standard, in °C

Φ sight-factor, unless otherwise specified equal to 1.0

ε_m emission factor of the surface of the member; unless otherwise specified equal to 0.8

ε_f Emission factor of a fire, generally equal to 1.0

σ Stephan Boltzmann constant (= $5{,}67 \cdot 10^{-8}$ W/m^{2}K^4)

θ_r effective radiation temperature of the fire compartment, in °C

η reduction factor of loading in case of extreme condition compared to ultimate limit state

c specific heat

ρ density

λ_{rel} relative slenderness ratio

σ, ε stress and strain

α_L linear expansion coefficient

n strain hardenings factor

Subscripts

Lim represents a limiting value of the quantity

θ quantity at elevated temperature

al property of aluminium

st property of steel

p property of insulation

m property of member

c convection parameter

r radiation parameter

net netto value

el elastic

th thermal

cr creep

1. INTRODUCTION

The probability of a fire occurring within a dwelling is one in sixty-seven on a yearly bases according to CBS data on 2016 [1], proving it is one of the most common disasters to occur and thus to guard structures against. In general structures deteriorate during a fire, the reason why fire safety design is part of building design and implemented into the Eurocode standards. Within the standard it is presented as a minimal time period, depending on utility type, over which the element must retain its functionality as to allow for evacuation. It states that the fire resistance of a structure can either be determined through physical testing, by following standardized calculation methods or more advanced numerical models [2].

The calculation methods to describe mechanical behaviour as presented in the Eurocodes, such as the strength reduction method, are dependent on member temperature. Temperatures are described using thermodynamic theory, specifically Fourier's equation. This equation describes the heat flux as the product of thermal conductivity and the dimensional temperature gradient. To calculate the temperature of the member [2] the equation could be simplified by making assumptions regarding material properties and thermal processes. The Eurocode reduces Fourier's equation to describe the heat flux as attributed to convection and radiation [2] and for aluminium and steel, assumes a constant temperature gradient over the cross-section due to the relatively high heat conductivity of the material [3][4]. The relation between thermal and mechanical behaviour of materials in this context is often presented in the form of reduction factors. These reductions factors with which the mechanical strength and/or stiffness value of a material is multiplied at a certain elevated member temperature, is often presented in table format [4]. The reduction factors for a material are based on experimental data, as found from mechanical stress tests while the sample is exposed to certain constant elevated temperature [3][4]. This displays an inherent overlap between theory based and empirical approaches to fire safety design.

Given the fact that metals have a high heat conductivity, heating of a metal section occurs more quickly compared to concrete and timber. Such sections must be protected when the fire resistance would otherwise prove insufficient. Elevated temperatures affect the material properties of the metal and cause a rapid drop of the Young's modulus and a sustained reduction of the load carrying capacity of the structure as the proof and ultimate strength limits, $f_{0,2}$ and f_u are diminished [2]. This can lead to large deformations and eventual collapse. Aluminium elements are more vulnerable in comparison to steel, showing the onset of deterioration at temperatures as low as 175℃ and an ultimate temperature of 600℃ [5].

To ensure structural elements meet the time requirement as set by the Eurocode, most cases require additional protective material. There are several factors tying into the behaviour of structural elements under fire load. Factors as thermal expansion, temperature, exposure to fire load, protective cover, loading, creep, and connections are expected to be influencing the behaviour of the structure and the performance of insulating material [6]. New insulating materials must be tested to determine their material properties such as heat conductivity. EN 13381 prescribes fire testing methods to determine these properties for use with structural members. The current setup for standard fire tests for aluminium elements is taken from the prescribed European standard for steel. However, it is unclear if the steel setup is representative when working with aluminium as a full evaluation has not yet been developed.

This thesis examines the behaviour of insulated steel and aluminium sections exposed to elevated temperatures. The aim of this study is to gain insight into the deformation of insulated aluminium members exposed to fire and to determine a possible alternative fire test dedicated to aluminium members. Therefore, the chapters in this thesis represent the steps taken to achieve this, namely a literary review and a transient non-linear finite element analysis in Abaqus. The entire simulation comprises of a transient thermal analysis based on Fourier equation and a non-linear mechanical analysis – with transient state test material properties for aluminium and steel, IPE and RHS sections in loaded and unloaded scenario's.

2. PROBLEM DESCRIPTION

2.1 Problem introduction

Even though the materials steel and aluminium are similar superficially – considering slenderness, thermal conductivity, ductility – there are still significant differences in properties and subsequent behaviour when exposed to elevated temperatures, such as the magnitude of thermal and creep deformations. This may imply that the materials require their own tests to acquire representative data on thermal conductivity and specific heat for different insulation materials. However, there is a European standard for steel fire test EN1363 and EN 13381 in which setup is described, while there is no such standard for aluminium to calculate fire resistance properties [7]. Currently the steel test requires twelve unloaded columns and two full scale loaded beam tests, from which the performance of the insulation material can be obtained [7]. The need for both column and loaded beam tests lays with the fact that a steel beam experiences significant sagging or larger deflections, potentially affecting the insulating material [8]. This can result in damage to the insulation layer or even a complete separation. Due to this loss of protection against elevated temperature, a steel section can heat more quickly at this stage. Hence the behaviour of protected sections can differ between that of a column and a beam. Full scale beam test are relatively expensive and considering the rate of return on investment in a smaller aluminium market compared to steel, it is of interest to determine if the loaded beam test is necessary considering the material behaviour. The question thus becomes, if the insulation performance of an unloaded aluminium column differs from that of a loaded beam as is with steel?

Given the material properties of aluminium, it is to be expected that other failure mechanisms occur before excessive sagging compromises the protective layer. Herein the effect of creep under elevated temperatures is of significant interest [9]. This would imply that the protected beam would reach a critical internal temperature of approximately 200-400℃, depending on the utilization ratio, that is load divided by resistance, before sagging damages the protective covering. Thereby negatively affecting the heating rate of the aluminium section. Expectations are that the deflection of the aluminium beam compared to steel are more favourable in a sense that creep occurs faster, possibly even omitting the need for the fire beam test altogether.

In addition to creep, aluminium has a higher thermal conductivity than steel. It is to be expected that the thermal gradient in this case is therefore lower, which carries into the effect of thermal bowing. Complementary, aluminium also has about a twice as large thermal expansion. All three aspect will come to light during the thesis.

2.2 Problem statement

The situation gives rise to the question whether the thermal and mechanical behaviour of a protected aluminium beam under fire load differs to that of a similar column. It is to be judged, if a protective insulating layer is negatively affected and a change in the gradual heating of the member is observed, as to warrant full scale beam tests as prescribed in EN13381 to determine the thermal properties of insulating materials when working with aluminium members.

There is therefore a practical need for a more lucrative alternative to the fire test setup when working with aluminium, leading to the following research question:

Does the deformation of a protected aluminium beam under fire load differ to that of a similar column, in such a way that the protective layer is affected and a change in the gradual heating of the beam is observed?

To tackle this subject, a set of sub questions have been formulated to serve as a starting base. Herein a distinction can be made between geometric, material and mechanical specific questions.

Geometric

a) *What are the specifications of the standard steel fire test EN1363 concerning support, connections, beam size, length, and protective covering?*
b) *What aluminium profiles are used in practice?*
c) *What are the geometric specifications of comparable beam segments of steel and aluminium for FEM analysis?*

Material specifics

d) *What material properties are subject to change during a fire?*
e) *What are determining factors for the fire resistance of a protected beam section that are to be considered or expected to occur, and are these coupled or sequential phenomenon?*

Mechanical

f) *What are appropriate failure criteria of the protective layer and beam section?*

FEM model

g) *What effect has a protective layer on the beam section heating over time and what are the protective layer equivalent properties within the FEM environment?*
h) *What is the mathematical equivalent for the FEM implementation of the fire load?*
i) *Is there comparative data available or attainable for verification of the model?*
j) *What recommendations/observations can be made for a standardized aluminium fire test proposal?*

2.3 Approach

To answer the main problem statement, the thesis is separated into two main parts, first that of a full literary study, and second a finite element analysis. The literary study is comprised out of evaluation of the data available in the ISO standards supplemented and evaluated with complementary research studies. All in all, this will set the basis for the theoretical background regarding the material properties at room and elevated temperatures for steel, aluminium and insulation materials in addition to the boundary conditions, available model techniques, failure mechanisms and validation possibilities.

The numerical model will serve to determine if the deformations that occur during a numerical fire, are significantly larger for a loaded aluminium beam compared to that of an unloaded column, requiring a standard fire beam test as currently prescribed in EN1363. If the deformation rate is similar between the two, the need for a beam fire test can be omitted. Thus limiting the standard fire test for aluminium to 12 unloaded columns to determine the properties of insulating materials.

The thermal and mechanical (creep and thermal expansion) analysis should be doable in sequential order, wherein the thermal analysis is input for the mechanical analysis. Combining the literary review, a numerical analysis and the critical review and improvement of the current standard EN1363 for aluminium, will comprise the complete thesis.

3. LITERATURE STUDY & THEORETICAL BACKGROUND

3.1 Normative texts

To test new insulating materials there are several standardized setups available that concern furnace specifications and the test specimen. The standards EN1363 and EN1365 respectively, describe the furnace specifications including air pressure, measuring sensors and equipment settings; and element specifications regarding material and usage typology. EN 13381 describes testing methods to determine the fire resistance of structural members due to protective measures such as insulation. Part 4 and 8 of standard EN13381 describe the setup of respectively passive and active fire protective measures with structural steel members. It is prescribed that for cases with steel structural members, it is necessary to test twelve unloaded columns and two full-scale loaded beams [7] to ascertain the properties of insulating materials. The need for two full-scale beam tests is based on the fact that steel beams experience significant sagging before failure[8], in comparison to columns. This behaviour implies that due to the deformation, the protective cover can be torn, cracked, fall away or be otherwise damaged and thus allow for more rapid heating of the section [7].

3.1.1 Measured parameters

Given that fire tests are performed to classify the insulation material, a difference is made between active and passive systems. An active system would be a reactive foaming coating for instance. What is known a priori of the insulation is the thickness of the applied layer, which is a parameter used for calculation at a later stage [8]. NEN-EN 13501-2:2016 Annex B prescribes that individual and mean temperatures of surfaces – both of member and outside insulation – and cavities are measured during testing [8]. From the thermal analysis a series of material dependent tables are produced which sets certain fire resistance periods (of 15, 30, 45, 60, 120, 180, 240 minutes) against critical design temperatures (for steel ranging between 350 to 750℃ with 50℃ increments) for certain insulated section factors A_p/V [8]. An example of such a table is table B.2 in the standard NEN-EN13501. Such a material dependent table can then be cross-referenced with technical datasheets of insulation fabricators to get the required insulation thickness [10].

The values regarding the geometry of the cross-section, insulation thickness and the transient temperatures at the surfaces and cavities, as measured during testing, are used as input for the calculation of the material properties of the insulation [8]. The applied equations are based on Fourier Equations on thermodynamics. In basic form this would be $q = -k\nabla\theta$, describing the local heat flux density as the product of the negative thermal conductivity of all materials in the referenced space, multiplied by the spatial temperature gradient. By combining Fourier's differential equation with the heat equation, as further discussed in chapter 3, and inputting the specific heat and density values as obtained from other tests, the effective thermal conductivity of the insulation is calculated [8]. The thermal conductivity is deemed effective because its value is expressed in relation to the (steel or aluminium) member temperature as to fit simplified mathematical models instead of its true absolute value that would be related to the temperature of the insulation material at that exact location. Adjusting the thermal conductivity is a necessity because the value of the temperature of the insulation in the test is not measurable but only established relatively to material surface temperatures which implies circular (mathematical) dependencies. The standard states that the variation of the thermal conductivity is a function of temperature, and its values are found using the mentioned equations [6]. Subsequent, the temperature dependency of the thermal conductivity is found through linear regression analysis [11].

3.1.2 Fire test setup

The size of the test is dependent on the size of the furnace, which is ordinarily no larger than five by seven metres [12]. Columns are subjected to fire on all sides, while beam tests are setup to simulate a three-sided fire. When performing a fire test aerated concrete blocks are used to simulate the flooring in case of beams. These are placed on top of the beams and are highly insulating to simulate three-sided heating. Generally these blocks cover the entire length of the beam and are 600 mm wide and 120 mm high [13].

The full-scale beam test is mechanically loaded, in contrast to the column tests – represented as a simply supported four-point static bending test loaded with hydraulic jacks [13]. The load is constant during the test. Given that a fire is an extreme load situation [14], the load is significantly less in comparison to the fundamental load combination as expressed in EN 1990. In case of fire, the utilization – applied load divided by carrying capacity – of the cross-section is suggested to be 0.65 for steel [3]. This value is based on the reduction of the extreme load due to combination factors under fire conditions.

3.1.3 Failure criterion

Failure of beam elements under fire conditions is expressed in both a deflection and a rate of deflection limit, equations (*1*) and (*2*) respectively [6][13]. These limiting values are based on securing a representative data range given the type of structural member and an adequate safety level to prevent damaging the equipment. Sudden and uncontrolled failure of specimens can cause damage to the furnace and equipment used in the setup. As some irregularity can occur before stable conditions are reached – such as settling due to initial loading – the rate of deflection limit is not applied until a deflection equal to L/30 has been reached [7].

Deflection limit
$$D_{Lim} = \frac{L^2}{400d} \ mm \tag{1}$$

Deflection rate limit
$$\frac{dD}{dt_{Lim}} = \frac{L^2}{9000d} \ mm/min \tag{2}$$

As the column tests are unloaded, deflection is not a failure criterion. Thus, failure of the section becomes an integrity problem [6]. To measure an integrity failure of a beam or column member, a gap gauge can be used to measure whether the insulation layer can be penetrated either by a gap of 6mm running 150mm long, or by a gap of 25mm [6], the length of the specimen is not discussed. Other failure criteria relate to the critical temperature of the material [6].

In literature for mechanical FEM models as to determine the fire resistance, failure of beams in three or four point bending tests is defined at one calculation step before material fracture [13]. In case of loaded columns, failure is defined at flexural buckling [15] as to describe the fire resistance. However, the columns exposed to fire are unloaded, failure is therefore not due to buckling. The column is subject to thermal expansion and eventual melting of the material. No models were found as to attain the thermal properties of insulating materials. When insulation was applied, the properties of the material were known a-priori and used as input for the model.

To date, a normative setup specified for insulation fire tests with aluminium sections is not available.

3.2 Thermal analysis

As previously discussed, a fire test is performed to obtain data regarding the temperature of all possible surfaces and cavities of the test member, for which deflection values are determining for failure in case of beams. The result is then used to evaluate the material properties of the insulating material, that is the effective thermal conductivity [8]. The effective thermal conductivity is calculated using the differential equation method [8].

3.2.1 Calculating properties from test data

The calculations involved in finding the thermal conductivity are based on Fourier's equation on thermodynamics $q = -k\nabla\theta$ [8][16], describing the local heat flux density as the product of the negative thermal conductivity multiplied by the negative temperature gradient across the surface. This equation is combined with the heat differential equation $\partial\theta/\partial t - \nabla^2\theta = 0$ given the law of conservation of energy. ∇ denotes the Laplace operator for a three dimensional problem. The heat transfer is obtained by considering the difference between the gas temperature as generated by the fire, and the surface temperature of the member, attributed to convection and radiation [2]. The change in heat transfer per unit volume in the insulation is proportional to the change in member surface temperature multiplied by the specific heat and density of the insulation $\Delta Q = c_p \rho_p \Delta\theta$ [4]. The temperature gradient is three-dimensional and within the NEN standard [3][4] is simplified by assuming that the temperature is constant over the cross-section of the member, reducing it to a one-dimensional problem [3][4], $\partial q/\partial t = -kA\, \partial T/\partial x$. These assumptions and simplifications reduce the formula to (*3*) [17][16], which is a partial differential equation with one unknown, dependent on both t (time) and X (one-dimensional location).

$$\frac{\partial\theta_g}{\partial t} = \frac{k_p}{c_p \rho_p}\left(\frac{\partial^2\theta_m}{\partial x_p^2}\right) + Q \qquad (3)$$

Herein it is assumed that the effective thermal conductivity is constant over the thickness of the material (x) [8][18]. As expressed in paragraph 2.1 the effective thermal conductivity k_p found by solving this differential equation for a series of temperatures and then performing a linear regression analysis dependent on the material temperature [8][11]. The change of gas temperature over time $\partial\theta_g/\partial t$, is known as it is taken from experimental temperature measurements. The rate of change of the temperature of the material(s) over distrance x, and/or the spatial partial derivative of θ over x twice, is approximated by considering the measured surface temperatures. Q stands for the heat energy added or lost in the system, also known. Inputting these values, in addition to the values for c and ρ, into (*3*) leaves one unknown, the thermal conductivity of the insulation k_p. Note that k_p is not a constant, but a function of the member temperature due to method with which it is established. Inconsistencies between test-setups such as geometry and the number of fire exposed sides are accounted for by adjusting the spatial derivative in the equation [3][4][8].

In summary, the fire tests are performed to obtain data regarding the surface temperature of the insulated member and to then calculate the thermal conductivity of the material. To do the calculations, it is necessary to collect data regarding the specific heat and density of both the member and insulation, the gas temperature and member surface temperatures. These values are used as input for (*3*) to calculate k_p. In the thermal analysis a given k will be used to determine the member temperature (nodal temperatures within the material) [8], essentially performing the previously described calculation method in reverse order.

3.2.2 Simulated fire

During a standard fire test, the temperature development within the chamber follows that of the nominal fire curve [3][4][8]. This curve represents the environmental gas temperature due to a fire as described in Eurocode EN 1991-1-2, see equation (*4*) [2]. Herein t stands for the elapsed time in minutes and θ_0 is the initial gas temperature. The initial values are described by ambient conditions and at t is zero the initial temperature is equal to the gas temperature and the member temperature, thus 20°C.

$$\theta_g = 345\log(8t+1) + \theta_0 \qquad (4)$$

The same gas-temperature curve is applied in several studies [13][18]. However, alternatively to this fire curve, steady state experiments wherein the temperature is set at a constant value are performed to evaluate post-fire behaviour [19], creep behaviour [15][18] and buckling [5][8][14][19][20] of protected and unprotected sections. The temperature range of the material itself as used in these studies, is limited to 200-500°C given the aluminium melting temperature, which is lower than the temperature that might occur during a fire . Protection of the aluminium main load bearing structure is certainly required in these cases.

3.2.3 Member temperature

The calculation of the temperature of an uninsulated aluminium or steel member (*5*) in [3][4] is straightforward and follows from Fourier's equation. The change in member temperature $\Delta\theta_{al(t)}$ expressed as the multiplication of shadow effects k_{sh}, inverse of specific heat c times density ρ, the section factor $\frac{A_m}{V}$, heat flux h'_{net} and time increment Δt. In [3][4] the assumption is made that due to the relatively high thermal conductivity of aluminium the temperature over the cross section is constant, thus equalling k_m to infinity. In case not all sides of the member are exposed to the fire, the section factor is adjusted [3][4] as in the spatial derivative of (*3*). The heat flux is expressed as the result of both convection and radiation, taken as the difference between gas and member temperature multiplied by the convection coefficient and emissivity of the material [4][16]. However, the heat flux can be substituted with Fourier's law, as done in (*3*). Alternatively, the heat flux can be approximated considering the type of fuel for the fire after one hour of exposure. Cellulosic fuelled fire ($q=150kW/m^2$) and hydrocarbon fuelled fire ($q=200kW/m^2$) are generally used for testing of structural materials [16].

$$\Delta\theta_{al(t)} = k_{sh}\frac{1}{c_{al}\rho_{al}}\frac{A_m}{V}\,h'_{net}\Delta t \qquad (5)$$

Besides the thermal conductivity of the insulation, the effect on the temperature of the member due to the insulating layer is expressed with factor phi [3][4], taking the insulation specific heat and density over that of the aluminium properties multiplied by the thickness of the layer in comparison to the section factor [4]. The thermal conductivity of the metal is considered to be infinite and all material properties are assumed to be constant over the cross-section of the individual materials [4] as similarly done in (*5*). Given a parametric fire, heat energy is added to the system, making Q non-zero and expressed as the change in gas temperature [2]. Equation (*3*) can then be rewritten to express an approximation of the surface temperature of the metal integrated over both the one dimensional geometry of the cross-section and time, as expressed in (*6*)[3][4]. For insulating materials commercially available, the thermal properties are expressed to fit with this equation.

$$\Delta\theta_{al(t)} = \frac{\lambda_p/d_p}{c_{al}\rho_{al}}\frac{A_p}{V}\left[\frac{1}{1+\Phi/3}\right]\left(\theta_{g(t)}-\theta_{al(t)}\right)\Delta t-\left(e^{\phi/10}-1\right)\Delta\theta_{g(t)} \qquad (6)$$

$$\text{With } \Phi = \frac{c_p\rho_p}{c_{al}\rho_{al}}\,t_p\,\frac{A_p}{V}$$

The Eurocode approach to approximate the member temperature during a fire is generally reasonably accurate, though conservative [5][8][19] for insulation materials with low density and high thermal resistance in comparison to the exact solution found using Laplace transformation [13][18][5]. This is due to the adjustment of the spatial derivative, delayed thermal response through the exponent and the presumption that the thermal conductivity is infinite. The application of the exact solution is dependent on the complexity of the thermal parameters related to the thermal resistance of the materials [18].

3.2.4 Thermal material properties

The material properties that must be defined are the thermal conductivity (Figure 1), specific heat (Figure 2) and density values of the metal and insulation. These may be dependent on temperature and geometry.

3.2.4.1 Insulation

To fit the fire resistance requirements for structures, it may be necessary to provide metal members with insulating material. Generally there are three types of insulation, namely boards, spray mortar and coatings [23].

The fire tests as discussed previously, are used to gauge the material properties of insulating material. In advance to the test, the data on the material is just an indication or unknown. In practice the result is often translated to design tables with a relation between A/V, fire resistance time and fire design temperature [8]. Even though experiments show a dependence of the thermal conductivity on temperature, the effective value is calculated with the differential equation method using the variation of the thermal conductivity, specific heat and density of the material [8].

These values can be taken from technical datasheets. An example can also be found in [18] for a ceramic fibre blanket. Other fibre based materials show comparable thermal properties [24][25][26][27][28].

$$c_p = 820\,J/kgK\,;\rho_p = 96\,kg/m^3\,;k_p[W/mK] = \begin{cases} 0.033 - 1.443 \cdot 10^{-8} \cdot \theta_p + 2.875 \cdot 10^{-7} \cdot \theta_p^2 \\ 0.12\ mean\ value \end{cases}$$

3.2.4.2 Contact and cavities

At the cavity between the metal member and the insulation, some thermal resistance might occur. This thermal resistance is due to a lack of full contact between the materials. The contact resistance is determined by the roughness of the surfaces and the contact pressure between them [29]. However, this contact resistance is often neglected, due to the numerical difficulty of implementing its effect [29]. This would imply that the temperature on the inner surface of the insulation is the same as the temperature of the structural member, which would be a conservative assumption and is neglected in calculations for insulation materials [29]. Contact resistance of this nature has been evaluated between steel and concrete to be 200W/m^{2}K [23][24][32].

3.2.4.3 Steel

EN 1993-1-2 [3] has a well-established base line for the material properties of steel. Even though the thermal conductivity is assumed to be uniform in thickness direction when calculating the member temperature, the parameter is temperature dependent.

Density [3] $\rho_s = 7850\,kg/m^3$

Poisons ratio $\nu = 0.29 - 0.31$ between temperatures of 0-700°C [33].

Thermal conductivity [3][22]

$$\lambda_s[W/mK] = \begin{cases} 54\ if\ \theta_s < 20°C \\ 54 - 3.33 \cdot 10^{-2} \cdot \theta_s\ if\ 20°C \leq \theta_s < 800°C \\ 27.3\ \ if\ \theta_s \geq 800°C \end{cases} \tag{7}$$

Specific heat [3][13]

$$c_s[J/kgK] = \begin{cases} 425 + 7.73 \cdot 10^{-1} \cdot \theta_s - 1.69 \cdot 10^{-3} \cdot \theta_s^2 + 2.22 \cdot 10^{-6} \cdot \theta_s^3\ if\ 20°C \leq \theta_s < 600°C \\ 666 + \dfrac{13002}{738 - \theta_s}\ if\ 600°C \leq \theta_s < 735°C \\ 545 + \dfrac{17820}{\theta_s - 731}\ if\ 735°C \leq \theta_s < 900°C \\ 650\ if\ 900°C \leq \theta_s < 1200°C \end{cases} \tag{8}$$

Alternatively to [3], [18] proposes for the specific heat a different singular equation. This simplification is based on the conclusion in [34] where the accuracy of the specific heat has little effect on the steel temperature calculations. $c_s\ [J/kgK] = 472 + 3.8 \cdot 10^{-4} \cdot \theta_s^2 + 0.2 \cdot \theta_s$

3.2.4.4 Aluminium

Material properties as expressed in [4] are based on steady state tests. For the thermal conductivity and specific heat properties, the values are often the same between literature [18][13][35] and Eurocode.

Density [4] $\rho_{al} = 2700\,kg/m^3$

Poisons ratio [9] $\nu = \begin{cases} 0.33 - 0.40 \; for \; alloy \; 6060 - T66 \\ 0.33 - 0.43 \; for \; alloy \; 5083 - H111 \end{cases}$

Thermal conductivity [4][16]

$$\lambda_{al}[W/mK] = \begin{cases} 0.07 \cdot \theta_{al} + 190 \; for \; 0℃ \leq \theta_s < 500℃ \; alloy \; 6XXX \\ 0.1 \cdot \theta_{al} + 140 \; for \; 0℃ \leq \theta_s < 500℃ \; alloy \; 5XXX \end{cases} \quad (9)$$

Specific heat [4][16]

$$c_s[J/kgK] = 0.41 \cdot \theta_{al} + 903 \; for \; 0℃ \leq \theta_s < 500℃ \quad (10)$$

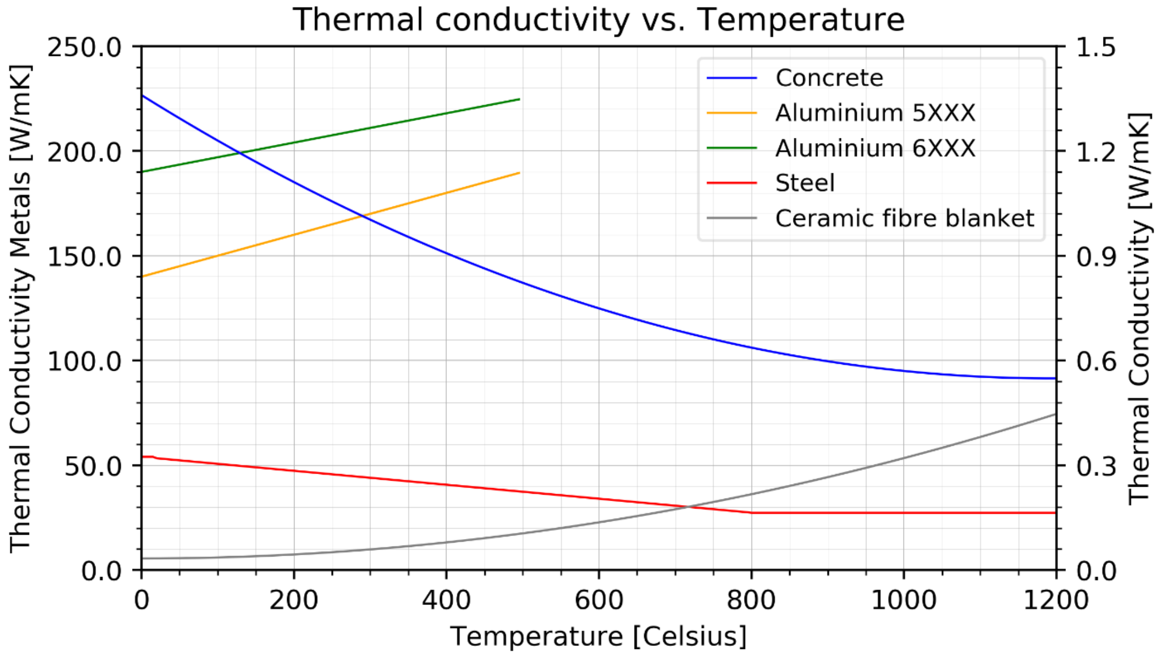

Figure 1 – Thermal conductivity of the materials aluminium, steel and insulation (ceramic fibre blanket) as specified in chapter 3.4. The grey and blue line refer to the right handed axis.

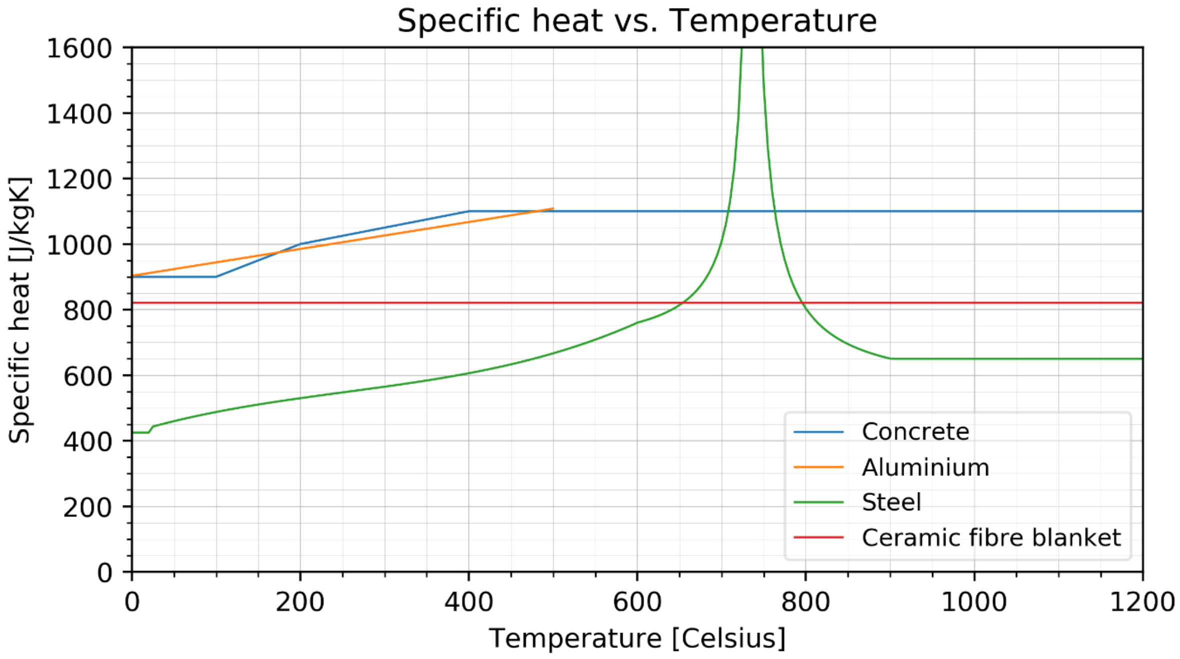

Figure 2 –Specific heat values of aluminium, steel and insulation (ceramic fibre blanket) according to chapter 3.4.

3.2.4.5 Concrete

To represent a floor system, the use of a concrete slab is a possibility. For that purpose, the thermal properties of concrete are taken as expressed in NEN-EN 1992-1-2.

Density

$$\rho_c[kg/m^3] = \begin{cases} \rho_c(20°C) = 2300\ for\ 20°C \leq \theta \leq 115°C \\ \rho_c(20°C) \cdot (1 - 0.02(\theta - 115)/85) for\ 115°C < \theta \leq 200°C \\ \rho_c(20°C) \cdot (0.98 - 0.03(\theta - 200)/200)\ for\ 200°C < \theta \leq 400°C \\ \rho_c(20°C) \cdot (0.95 - 0.07(\theta - 400)/800) for\ 400°C < \theta \leq 1200°C \end{cases} \quad (11)$$

Thermal conductivity

$$\lambda_c[W/mK] = \begin{cases} 2 - 0.2451(\theta/100) + 0.0107(\theta/100)^2 \quad Upper\ limit \\ 1.36 - 0.136(\theta/100) + 0.0057(\theta/100)^2 \quad Lower\ limit \end{cases} \quad (12)$$

Specific heat

$$c_c[J/kgK] = \begin{cases} 900\ for\ 20°C \leq \theta \leq 100°C \\ 900 + (\theta - 100) for\ 100°C < \theta \leq 200°C \\ 1000 + (\theta - 200)/2\ for\ 200°C < \theta \leq 400°C \\ 1100\ for\ 400°C < \theta \leq 1200°C \end{cases} \quad (13)$$

3.2.5 Thermal gradient

The Eurocode [3][4] makes the assumption that the temperature over the cross-section is uniform, calculated following (*6*). This is, however, not the reality [8][29] and might yield conservative results [5][8][19]. Considering a three-sided fire, the heat input differs between sides and as the thermal conductivity of the material is not infinite, as seen in equation (7) and (9), a thermal gradient exists over the cross-section [8][29][37]. In studies, the thermal gradient is most often considered to be linear [22][38] or quadratic [19] over the height of the cross-section. The resultant thermal gradient is dependent on the material properties, the heat input and the geometry of the section.

Due to the thermal gradient, both thermal and mechanical material properties of the member differ over the cross-section. This, in turn, affects the temperature distribution as the thermal conductivity and specific heat are temperature dependent.

The mechanical implications of the thermal gradient include a difference in E and $f_{0,2}$ between the 'cold' and 'hot' side of the member. The former causes a shift of the neutral axis to the colder side due to the higher stiffness [22]. This would induce an eccentricity and an additional moment [22] dependent on the axis of the applied load. Furthermore, the thermal gradient can induce a bowing effect by thermal expansion, given that the hotter flange would extend more causing an internal eccentricity from the neutral axis opposite to that due to stiffness [22].

Additionally, the effect of the thermal gradient on the critical temperature of the cross-section of a column can be argued. [39] demonstrated that the critical temperature for a column is higher than with a uniform temperature distribution, while considering the maximum occurring temperature [22]. When considering the average temperature of the thermal gradient [22], [34][35] found that the fire resistance is reduced while [36][37] found it to have a higher resistance. The Eurocode [3] allows for the consideration of a thermal gradient, but specifies that the E and $f_{0,2}$ values for the maximum temperature are to be used, to counterbalance the shift of the neutral axis [22].

3.2.6 Thermal simulation model

Within a finite element (FE) package such as DIANA or Abaqus it is possible to perform a heat transfer analysis. In literature, the model is often simulated as eight-node quadratic heat transfer elements DC3D8 [21][22] or twenty-node quadratic heat transfer bricks DC3D20 [19]. Given the time and temperature dependent material parameters, a transient, material non-linear FE analysis is a requirement [21][22][19] to obtain the nodal temperatures.

3.3 Mechanical analysis

As expressed in chapter 1, metal structural members show deterioration at elevated temperatures. The temperature range at which mechanical deterioration occurs is different for both aluminium and steel. For aluminium, the material properties are defined over the range 0-550℃ [4] and for steel this is 0-1200℃ [3]. At higher temperatures the material properties go beyond the scope of mechanical engineering. Within this range, the strength and stiffness of the material is reduced to zero and failure is definitive [3][4][18][28].

3.3.1 Material strength and Young's modulus

The Eurocode has formulated the material properties of aluminium based on steady state experiments. Herein, the specimen is subject to a constant temperature, a fixed strain rate and the stress is measured [9]. However, [15][18–22] argues that transient state tests are more appropriate in case of fire conditions. A difference may occur when considering transient state tests opposed to steady state tests, which are considered more appropriate to fire conditions [18][25][26]. This is attributed to creep, overaging and annealing [9]. Alternatively to a steady state test, in a transient state test the member is subject to a changing temperature, a certain stress and the strain is measured [9]. Comparing the result of the stress-strain relationship shows that for alloy 5083-H111 the proof stress found through steady state experiments as in [4] is 20 to 85 pct higher than found with transient state tests for a temperature range of 200-350℃ [9]. Contrarily, the proof stress of alloy 6060-T66 is found to be 5-40 pct lower in [4] than in transient state tests for the same temperature range [9].

In literature several options are used to base proof stress and Young's modulus data on. These range from Eurocode [13][22], Kaufman suggestion [15][9] as used in [44][29], transient state tests [19][18–20][37][45] or steady state tests [20][37][46][47]. In Figure 3 and Figure 4 the development of the E-modulus and proof stress respectively, are plotted against an increasing temperature.

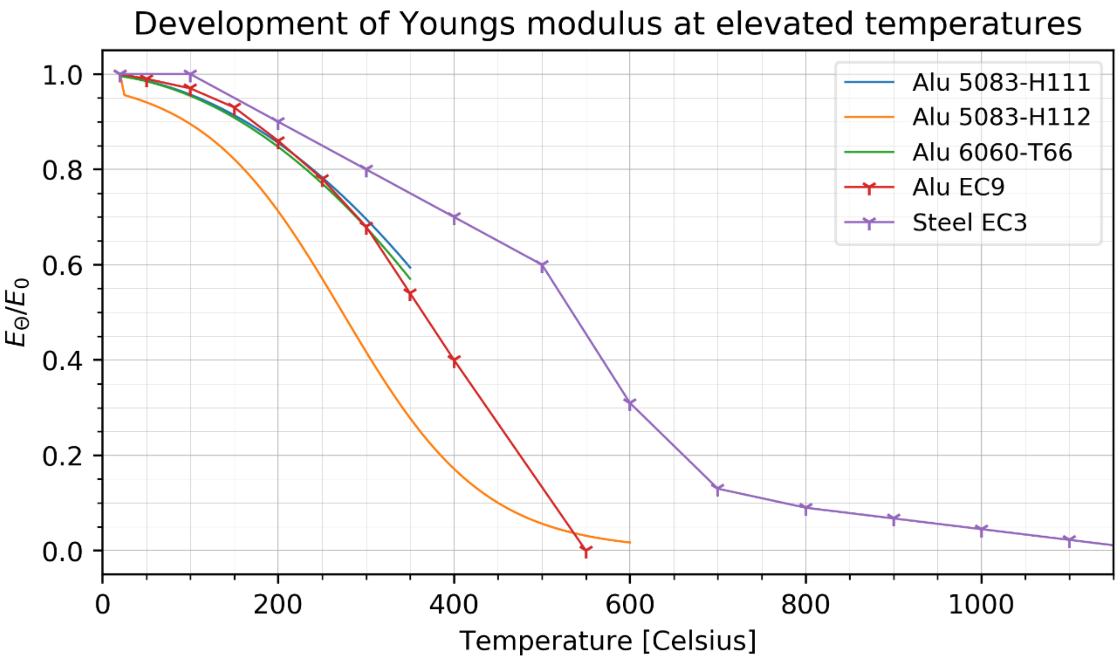

Figure 3 – Development of the Young's modulus at elevated temperatures compared to the nominal value at room temperature, as taken from different references [3][4][15][18].

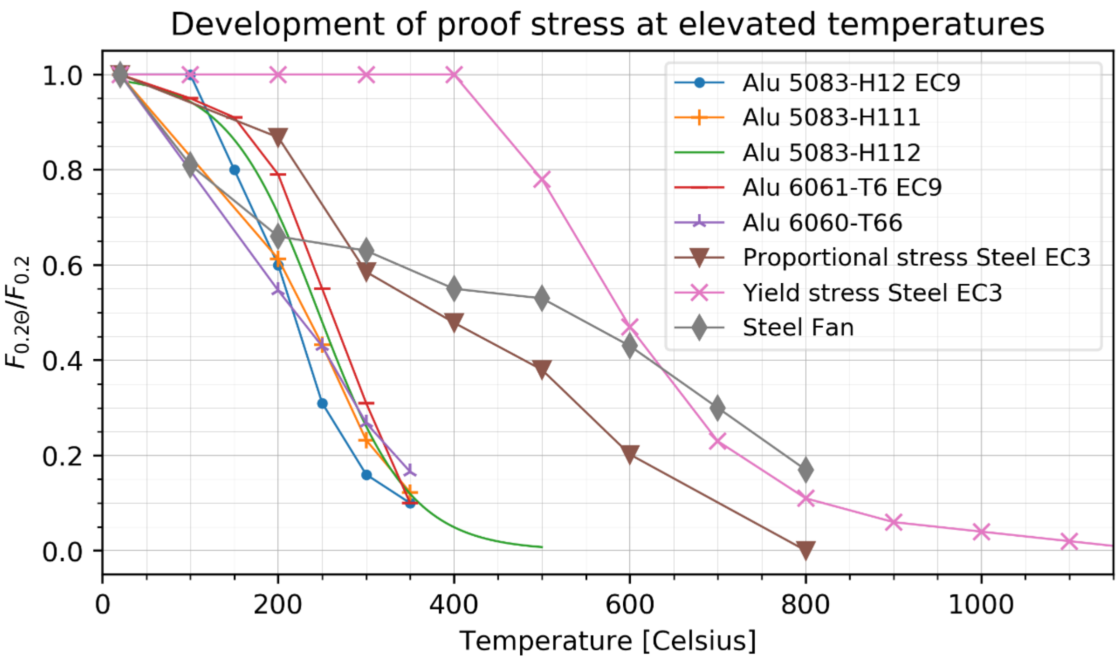

Figure 4 – Development of 0.2% stress at elevated temperatures compared to the nominal value at room temperature, as taken from different references [3][4][15][5][31][42] *for which the EC9 values for aluminium are based on steady state experiments.*

3.3.2 Strain

In general static mechanics, Hooke's Law $\varepsilon = \sigma/E$ is fundamental to describe the elastic relation between strain and stress [49]. or large strain, steel and aluminium show physical non-linear behaviour [49], as can be observed in Figure 5 as taken from [20].

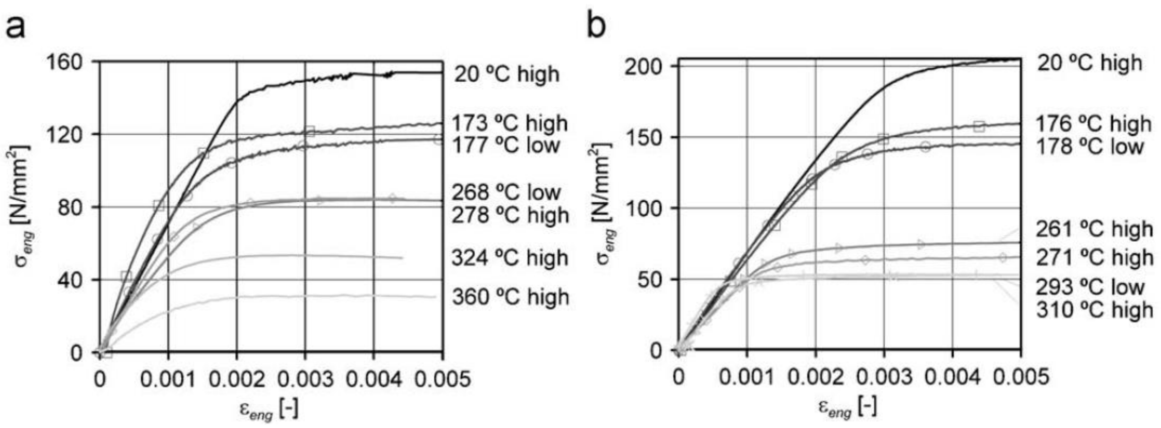

Figure 5 – Steady state stress-strain curves of (a) alloy 5083-H111 and (b) alloy 6060-T66 at elevated temperatures from [20].

3.3.2.1 Ramberg-Osgood relation

To describe the stress strain relation at elevated temperatures, the Ramberg-Osgood equation (*14*) is commonly used in literature [5], [15], [21]. This equation utilizes the corrected strength and stiffness parameters at elevated temperatures. [5] argues that this equation describes the stress-strain curves relatively well up to strain values of $\varepsilon = 0.01$, which would be adequate for structural applications as these are generally limited to small strains.

$$\varepsilon = \frac{\sigma}{E_\theta} + 0.002\left(\frac{\sigma}{f_{0.2;\theta}}\right)^n \tag{14}$$

Beyond strains of $\varepsilon = 0.01$ [5] and temperatures larger than half of the melting temperature (circa 150°C) [9][16] however, these stress-strain relations are no longer accurate. This is attributed to creep.

Alloys of 6XXX are less susceptible to creep strains than alloys in 5XXX series [35]. In addition to creep, thermal expansion affects the deflection of an element [16].

Considering these phenomena the total strain is a contingent of elastic strain, creep and that due to thermal expansion [16], which alludes to equation *(15)*.

$$\delta_{total} = \delta_{elastic} + \delta_{thermal\ expansion} + \delta_{creep\ I,II,III} \tag{15}$$

3.3.2.2 Thermal expansion

In case of statically indeterminate structures, thermal expansion due to elevated temperatures causes additional forces in the specimen [16]. Considering a thermal gradient over the cross-section of the beam, the amount of thermal expansion differs between the 'hot' and 'cold' side, causing a thermal bowing effect [22]. The eccentricity from the neutral axis due to this deformation causes an addition bending moment [22].

The lengthwise thermal expansion of steel is described as [3][22]

$$\frac{\Delta l}{l} = \begin{cases} 1.2 * 10^{-5} \cdot \theta_s + 0.4 \cdot 10^{-8} \cdot \theta_s^2 - 2.416 \cdot 10^{-4}\ if\ 20°\text{C} \le \theta_s < 750°\text{C} \\ 1.1 \cdot 10^{-2}\ if\ 750°\text{C} \le \theta_s \le 860°\text{C} \\ 2 \cdot 10^{-5} \cdot \theta_s - 6.2 \cdot 10^{-3}\ if\ 860°\text{C} < \theta_s \le 1200°\text{C} \end{cases} \tag{16}$$

The lengthwise thermal expansion of aluminium is described as [4][16]

$$\frac{\Delta l}{l} = 0.1 \cdot 10^{-7} \cdot \theta_{al}^2 + 22.5 \cdot 10^{-6} \cdot \theta_{al} - 4.5 \cdot 10^{-4}\ for\ 0°\text{C} \le \theta_{al} < 500°\text{C} \tag{17}$$

Thermal bowing can occur in both restrained and unrestrained sections, [50] expressed deformation of this kind for unloaded and unrestrained steel I-sections with a linear thermal gradient (ΔT) over the cross-section as equation *(19)*.

$$\delta = \frac{\alpha L^2 \Delta T}{8d} \tag{18}$$

3.3.2.3 Creep strain

In steady state experiments the effect of creep is typically underestimated. This phenomenon is of particular interest when temperatures exceed half the melting temperature [16][9][35]. In transient state tests the effects of creep, overaging and annealing is captured [9][20]. Ref. [9] considered creep implicitly by adapting the steady state stress-strain curves for alloy 6060-T66. Alloys in the 6XXX series are less susceptible to creep than those in the 5XXX series [35], and thus this method was applicable for alloy 6060-T66 [9].

Creep can be described in three stages, the primary stage in which the strain rate decreases, the secondary stage where the strain rate is constant and the tertiary stage in which the strain rate rapidly increases. These stages can be recognized in Figure 6 as taken from [16][9]. To take creep strains explicitly into account, the creep strain of the primary and secondary stage can be described with the Dorn-Harmathy model (*19*) [16][9]. Herein, $\dot{\varepsilon}_C$ stands for the strain rate of subscript I primary stage strain and II secondary stage creep strain, $\varepsilon_{C,I+II}$ for primary and secondary stage creep strain and $\varepsilon_{C,0}$ the projection of the secondary creep strain at time is zero. Equation (*19*) is explained in depth in [16][9].

$$\dot{\varepsilon}_{C,I+II}(t) = \dot{\varepsilon}_{C,II} \coth^2\left(\frac{\varepsilon_{C,I+II}}{\varepsilon_{C,0}}\right) \tag{19}$$

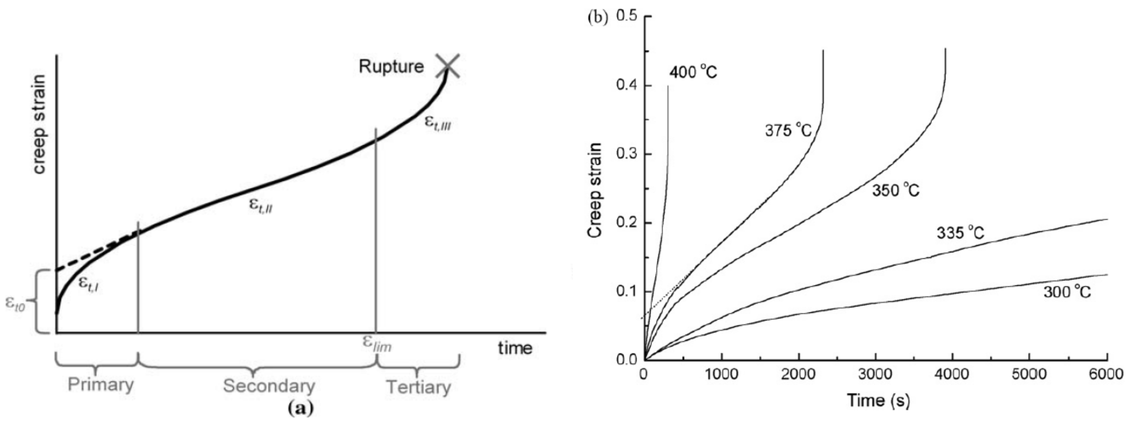

Figure 6 – Creep curve showcasing (a) primary, secondary and tertiary creep stage, source [9]*. And (b) creep curves at different temperatures with constant loading of 50 MPa, source* [16].

3.3.3 Loading

In structural design, fire design is part of ultimate limit state (ULS) as an accidental load case [51]. This implies that the loading - both permanent (G_k) and variable (Q_k) loads - can be adjusted in comparison to the fundamental load case [14]. For consequence class 2 (CC2) the safety factor with which to multiply permanent loads is $\gamma_G = 1.2$ and for variable loads it is $\gamma_Q = 1.5$ fundamentally[14]. For fire design, these safety factors (γ) are lower in addition to a reducing load combination factor Ψ_{fi}. The result is a load η times smaller than that of the fundamental load case, following equation (*20*) [4]. Thus, there is a certain degree of rest capacity, which is of crucial importance as the resistance of the member reduces under elevated temperatures, as can be observed in Figure 4[37].

$$\eta_{fi} = \frac{G_k + \Psi_{fi} Q_{k,1}}{\gamma_G G_k + \gamma_{Q,1} Q_{k,1}} \tag{20}$$

EN 1990 suggests that the load can initially be assumed 65% of the fundamental load case for steel [4]. This value is based on a conservative estimation of the lowered safety factors for fire design.

3.3.4 Failure mechanisms

A distinction can be made in failure type, that would be strength or resistance (R), stability (S) or integrity (I) failure [8][6]. Failure in literature is often described as the point at which rapid strain occurs without further adding to the load [15] such as buckling and necking[45]. Aluminium is predominantly used as slender plate or extruded material [5]. Given a compressive load, an aluminium member is therefore especially susceptible to out-of-plane buckling [22][14][19][20][39][40]. Under tension, aluminium primarily shows ductile failure. A 'neck' or thinning of the cross-section occurs and after extensive plastic deformation, fracture. Fracture is the point where the material starts to separate [47]. This mechanism can occur in members subjected to tension and bending. In numerical models, the time step right before fracture occurs is often formulated as the failure criterion [45][47]. For steel the same applies [13].

These phenomena can occur both at room temperatures and at elevated temperatures. However, strength and Young's modulus of both steel and aluminium drop with increasing temperature, as discussed previously. Given a certain load in the elastic strain range of aluminium under room temperature conditions, would normally be no cause for concern. Yet, when exposed to elevated temperatures, the same load would result in plastic behaviour as the yield and ultimate strength limits are much lower, see Figure 4 and Figure 5 for reference.

3.3.5 Mechanical simulation model

To simulate the deformation of the beam and column in the mechanical model, the thermal output is used as input, as discussed in Chapter 1. The nodal temperatures of the thermal heat transfer bricks discussed in *3.2.6 Thermal simulation* model determine the local strength and stiffness parameters. Considering a thermal gradient over the cross-section, E_{θ} and $f_{0.2\theta}$ variate accordingly [8][29]. As discussed previously, the fire resistance can be affected both negatively and favourably when comparing the actual parameters, against that of the average temperature or that of the maximum temperature [34][35][36][37]. The Eurocode [3] prescribes that E_{θ} and $f_{0.2\theta}$ should follow from the maximum temperature, to account for effects as restrained thermal bowing [22].

Within Abaqus, a finite element (FE) package, different element types have to be used given the type of analysis [21][22] thus heat transfer bricks for thermal analysis. Either analysis may be performed with a different number of elements and nodes, depending on their ability to describe the behaviour of the member accurately. Therefore, some interpolation might be necessary to generate the input data for intermediate nodes [19].

With FE package DIANA, [5] modelled the mechanical elements with eight node shell elements, type CQ40S and [21][22] also did so with reduced integration. Alternatively, [46][53] used elements of four node shells S4 in Abaqus. Other examples include [15] which applied linear, quadratic or cubic two node beam elements based on Timoshenko beam theory, while [19] proposes the use of fully integrated, solid, quadratic elements C3D20 as these can capture the linear stress gradient over the cross-section due to pure bending.

3.4 Aluminium section types

Aluminium is a material that can be extruded, such sections in practice are often designed to fit multiple purposes. An example is an Y-profile for offshore flight decks, which integrates an installation piping trench in the section, see Figure 8. A more general example is that of a decking element. Herein a slender aluminium member is designed similar to a truss in width direction with a solid circumference lengthwise. This setup makes for effective slender decking element Figure 7.

Figure 8 – Photograph Y-profile as made by Bayards B.V. with an integrated installation trench.

Figure 7 - Aluminium bridge decking [54]

As the beam test is focussed entirely on pure bending, an interesting section would be an I-section. The material distribution to the flanges of such a section lends itself well to bending. Beyond I-profiles and decking elements, there is an extensive amount of research done into the phenomenon of local buckling of aluminium sections, a most common evaluated profile would be that of a rectangular hollow section composed out of thin sheets for specific research purposes [22][19][15][5].

Noting that no premature failure of the fire test specimen is allowed, the section should not be subject to local buckling. Thus – following the Eurocode – the section class [55][56] of the both the steel and aluminium specimens should be limited to that of class 1 to 3, consequently designed slightly more compact [5]. Class distinction can be made by regarding the relative slenderness λ_{rel} of the plates constituting the section, expressed as the square root between the plastic resistance N_{pl} over the critical resistance of the section N_{cr}. This can be likened to the square root of the yield stress f_y over Youngs modulus E, as expressed in equation (*21*). To fit the class criteria this λ_{rel} should not be below the value of 0.4 for aluminium [56].

$$\lambda_{rel} = \sqrt{\frac{N_{pl}}{N_{cr}}} \cong \sqrt{\frac{f_y}{E}} \tag{21}$$

The Eurocode assumes that the section class of any member does not vary from room temperature conditions [4][5]. However, considering the normalised values of the Young's modulus and yield stress of steel, it can be observed that for steel the Young's modulus degrades faster than the yield stress does [3]. Following the relation of the relative slenderness as expressed in (*21*), this would result in a higher value at elevated temperatures and thus classify the section differently, that is of a higher order.

For aluminium, the normalised values for the Young's modulus and the 0.2 percent proof stress prove contrary to the normalised value for E and f_y of steel. Herein the stress value drops more quickly than the Young's modulus [4], resulting in more favourable ratio and thus a lower section class with higher temperatures. This phenomenon showcases that there is no need to design an aluminium member as that of class 1 to circumvent local buckling mechanisms, as the section becomes less susceptible at elevated temperatures [5].

4. FINITE ELEMENT THERMAL ANALYSIS

4.1 Model description

As discussed in Chapter 3 Literature study & theoretical background the material properties of steel, aluminium and the insulation is dependent on temperature. Temperature is in this case a transient variable. The dependency is at first glance assumed a one-way street. This implies that a thermal and mechanical analysis can be done in sequential order. However, as previously discussed, in experimental studies a steel beam fire test is a necessity because a steel beam shows significant sagging before failure. This means that the insulation can be damaged and the temperature of the steel member is affected. To determine whether this is a concern for aluminium members, it is necessary to perform an extra check. Should the strain of the loaded beam exceed a limit, coupled thermal-mechanical simulation is necessary. This limit is dependent on the bond between the metal member and the strain of the insulation and will be addressed in the mechanical analysis.

The full FEM analysis can be separated into two main parts: (1) thermal analysis and (2) non-linear mechanical analysis. Consequently, a check is performed to see if the final time step N is reached and if the strain limit for the insulation is not exceeded. If this limit is exceeded, an alternative analysis (3) is proposed. These two steps are repeated one after the other for step n, wherein n stands for the iteration step between 0 and N, until n=N, after which the process is terminated. As the thermal analysis is transient, the iteration between 0 and N is expressed in time, and n is thus a multitude of the time step.

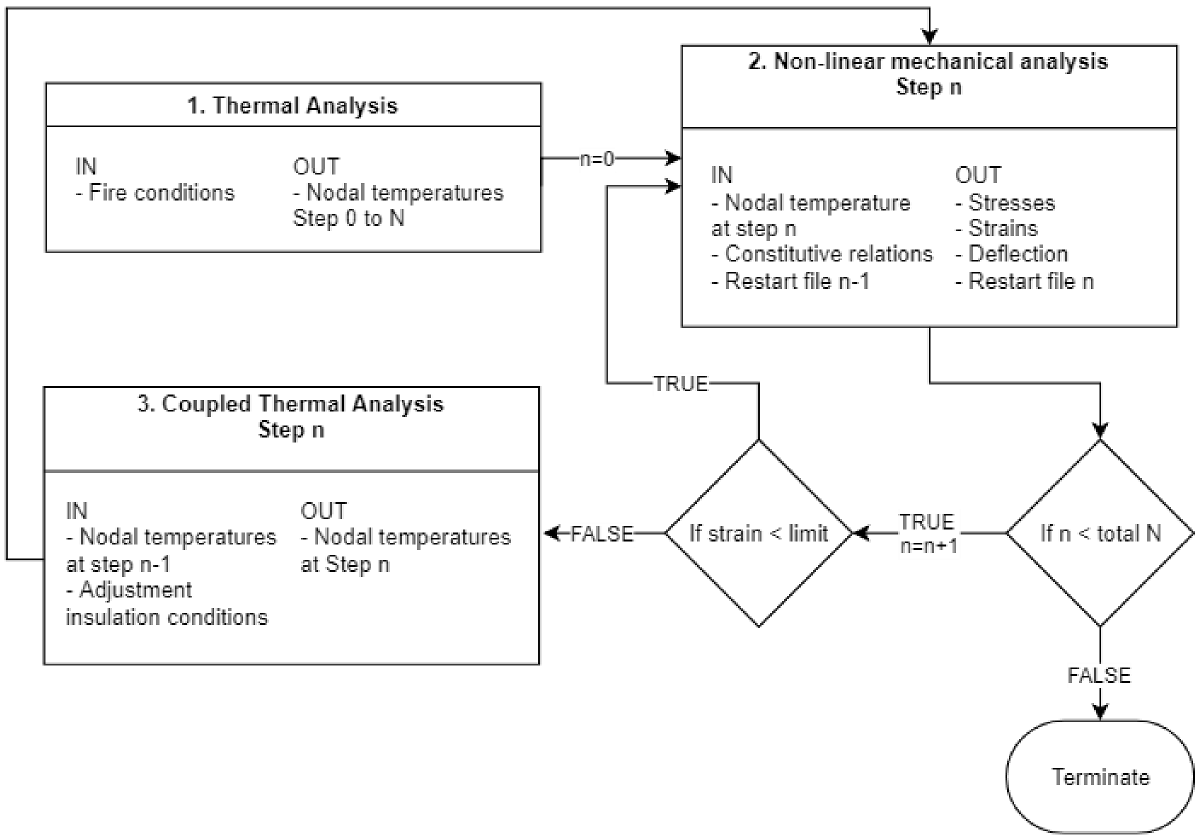

Figure 9 – FEM model set up, pertaining a thermal and mechanical analysis within the ABAQUS/CAE environment.

4.2 Thermal analysis

Initially it is assumed that the strain limit is not reached. Furthermore, the fire input for the member is constant over the length, in Z-direction. These assumptions present an opportunity for simplification of the FEM model. The nodal temperatures in the thermal analysis are found by modelling a two-dimensional (XY-field) deformable body subject to heat transfer. This is a possibility because the nodal temperature would be constant in Z-direction, only differing in X and Y direction depending on fire exposure.

Depending on the configuration, the metal member comes into contact with a floor system and the insulation. It is assumed that due to a lack of full contact between the surfaces, a thermal resistance between these elements exists. The value of this resistance between elements is set to be either 200 W/m^{2}K as discussed in Chapter 3 Literature study & theoretical background or set at unit value. This counts for all interactions between elements. Thermal contact can be simulated using surface to surface contact discretization. As the bodies do not move relative to each other, sliding can be formulated as being small, which is an approximation of the general contact master-slave algorithm and thus faster than finite sliding, which is just a formality. The metal member acts as the master surface and the insulation is the slave. Inaccuracies can occur due to the use of a coarse mesh, this will be further discussed in paragraph 4.2.1.3 Sensitivity analysis of mesh density.

The material properties of steel, aluminium and the insulating ceramic fibre blanket, namely thermal conductivity, density and specific heat, are prescribed in chapter 3 Literature study & theoretical background. Additional constants and conditions are as stated in Table 1 and Table 2. The convection coefficients are the same as stated in NEN-EN 1991-1-2 for fire loading. The transient temperature of the member, insulation and flooring is determinant by the temperature dependent properties as calculated in the thermal analysis. Three different geometries are considered, that would be a square hollow section, a typical I-section and a decking member as specified in Table 3. These measures are taken such that the section classes are specified as non-slender, thus < class 4. These geometries are then analysed in three different situations, that of a column, a beam with a floor on top and an integrated floor beam. These situations will be discussed more in-depth in the following paragraphs.

*Table 1 – Temperature description of FEM model attributes. *in accordance to a surface covered with soot during a fire.*

Attribute	**Fire side**	**Member**	**Insulation**	**Floor**	**Ambient**
Initial temperature [°C]	20	20	20	20	20
Transient temperature FEM [°C]	Eq. *(4)*	Dependent	Dependent	Dependent	20
Transient temperature EC [°C]	Eq. *(4)*	Eq. *(6)*	N/A	N/A	20
Emissivity (ε_m)	1.0	0.7*[9][13]	0.7[2]	0.9[3]	1.0

Table 2 – FEM model thermal constants.

Attribute	**Value**	**Unit**
Total time (T)	90	Minutes
Step time (t)	6	Seconds
Boltzmann constant	5.67e-11	W/m^{2}K
Convection coefficient ambient ($\alpha_{c,ambient}$)	4	W/m^{2}K
Convection coefficient fire side ($\alpha_{c,hot}$)	25	W/m^{2}K

Table 3 – Geometric specifications of cross-section, all measures are in mm unless otherwise specified.

	RHS		**IPE**		**Decking**	**Insulation**
	Steel	Aluminium	Steel	Aluminium	Aluminium	Fibre blanket
H eight	200	200	200	200	60	-
W idth	200	200	100	100	514	-
t hickness	9	6	-	-	-	-
tf flange	-	-	8.5	9	4.0	-
tw web	-	-	5.6	5	2.0	-
tp	-	-	-	-	-	20

4.2.1 Column: a four-sided fire simulation

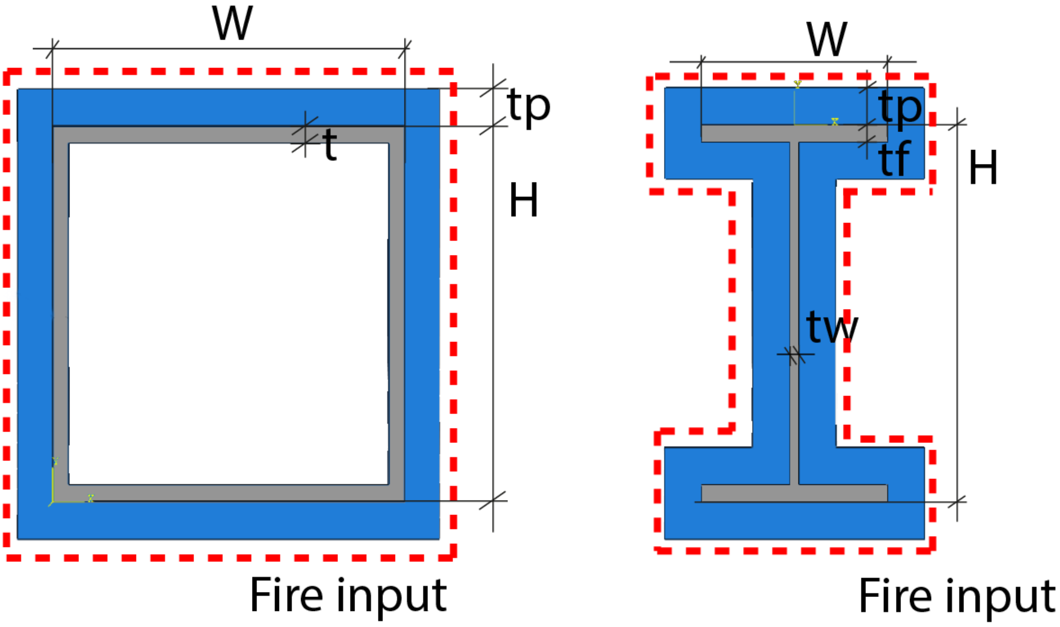

Figure 10 – Geometry of Rectangular Hollow Section and IPE cross-section respectively, measurements as given in Table 3.

Considering that in this four-sided fire simulation, the heat input is constant along the cross-section, shadow effects are disregarded for simplicity. Due to the constant heating from all sides and the high thermal conductivity of metals, the temperature distribution in the member is close to constant. Therefore it is possible to mesh these sections using two-dimensional 8-node heat transfer bricks. This computational simplification is supported in references [21][22]. An appropriate mesh size is discussed later.

4.2.1.1 Validation

Abaqus performs a transient heat transfer analysis with temperature dependent material properties following traditional Fourier's law differential equation *(3)*. In comparison to the Eurocode this should yield a more exact solution of the time-temperature curve of the aluminium and steel member. The Eurocode uses a simplified formula *(6)* to calculate the member temperature. The material properties of the insulation in this equation, are all dependent on the member temperature. This is opposite to the FEM analysis were the temperature in the insulation is calculated locally, dependent on local thermal properties.

For the FEM simulation, insulation data is based on that of a ceramic fibre blanket [18]. To determine whether the simulation is representative, the resultant member temperatures are compared to the outcome found with the simplified equation *(6)* for several commercial insulation types. These commercial types have their material properties tied to the temperature of the member, not local values as would be for a FEM analysis. The ceramic fibre blanket was used for the finite element analysis. The specifications for the thermal conductivity and specific heat is supplied in Table 4. The density of the blanket insulation types is 96kg/m^3.

Table 4 - Thermal conductivity and specific heat for several insulation types with the same density 96 kg/m3

θ	**Denka Blanket**[24]	**Marine blanket**[25]	**Kaowool blanket**[26]	**Ceramic fibre blanket**[18]	**Fyre wrap**[27]	**Coating**[28]
	EC	EC	EC	FEM	EC	350kg/m3
0				0,033		0,05
200		0,05	0,06	0,044	0,06	0,05
260	0,05			0,052		0,05
400		0,1	0,11	0,078	0,09	0,05
538	0,11			0,116		0,05
600		0,15	0,16	0,136	0,14	0,05
800		0,21	0,23	0,216	0,2	0,05
816	0,19			0,223		0,05
1000		0,29	0,32	0,320	0,29	0,05
1093	0,3			0,377		0,05
1371	0,44					0,05
c_p	1130	1130	1130	820	1130	1100

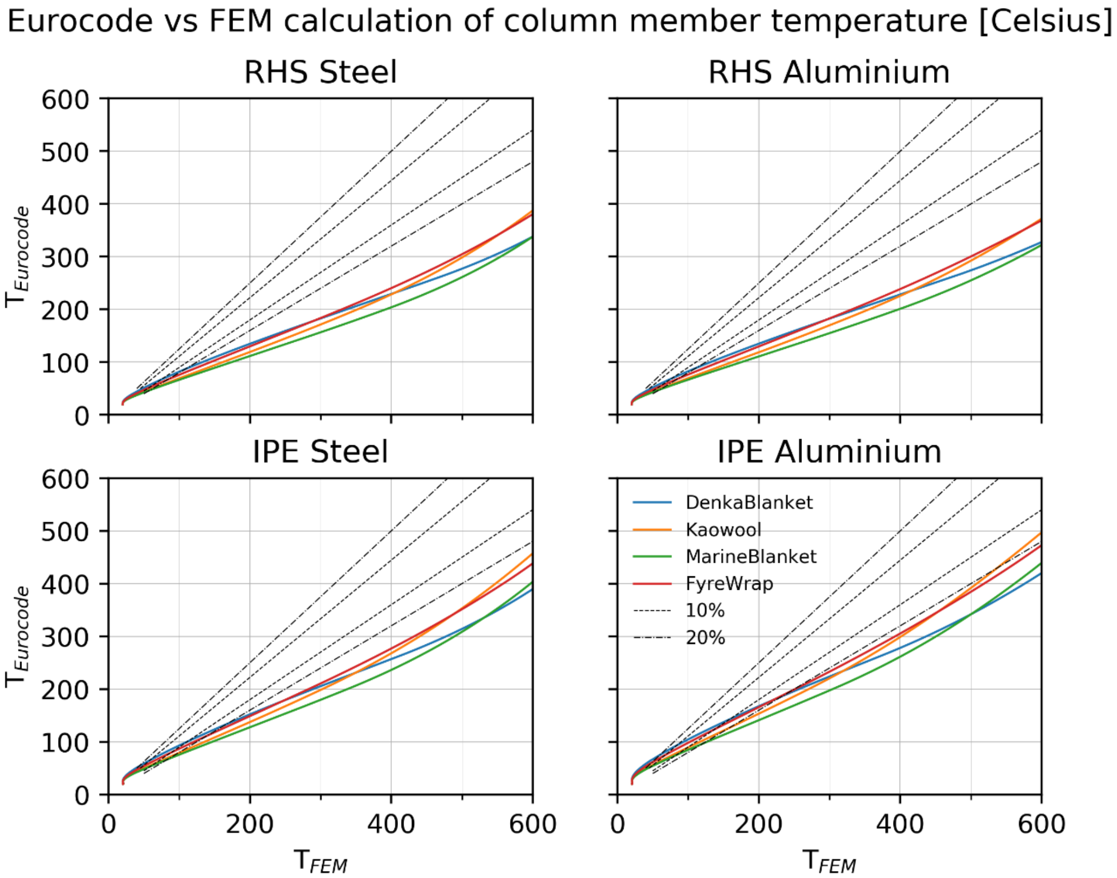

Figure 11 – Comparison of FEM simulation temperature results for cross-section with four-sided heating, with that of the simplified Eurocode equation for several insulation types. FEM temperatures versus EC found temperatures.

As is evident in Figure 11, the deviation between the FEM data and that of EC is significant, more than 20%. The cross-sections in the FEM analysis are heating at a faster rate than calculated according to the Eurocode. Given the material properties from reference [18] and the similarity in value to that of the commercial types, the fallacy of the FEM simulation lays with a discrepancy in the thermal conductivity and its temperature dependency. In specific, the temperature dependency of the thermal conductivity of the insulation is in relation to the member temperature as used in equation *(6)*, instead of the true local temperature at that particular FEM calculation node, as it should be.

As the deviation in these results do not sufficiently validate the thermal analysis, a literary reference is modelled and the results compared for additional confirmation. To achieve this, the model geometry is set to be a IPE of H140x100x6x6 with a 20mm thick spray-on coating, see last column in Table 4. The steel specimen in [28] was freely set in a furnace, thus heated from all sides, and the furnace temperature followed that of equation *(4)*. The steel thermal properties follow from NEN-EN 1993-1-2. In Figure 12 the black line represents the temperature data in the current FEM simulation as in Figure 13, which follows experimental and FEM ANSYS model of [28] quite closely. As the FEM model was setup in the same manner as previously discussed, and observing that the black line follows a similar trend in Figure 12, no full validation is achieved, though a certain level of correctness is observed. It is assumed that the model itself is representative and fault lies with incorrect insulation properties.

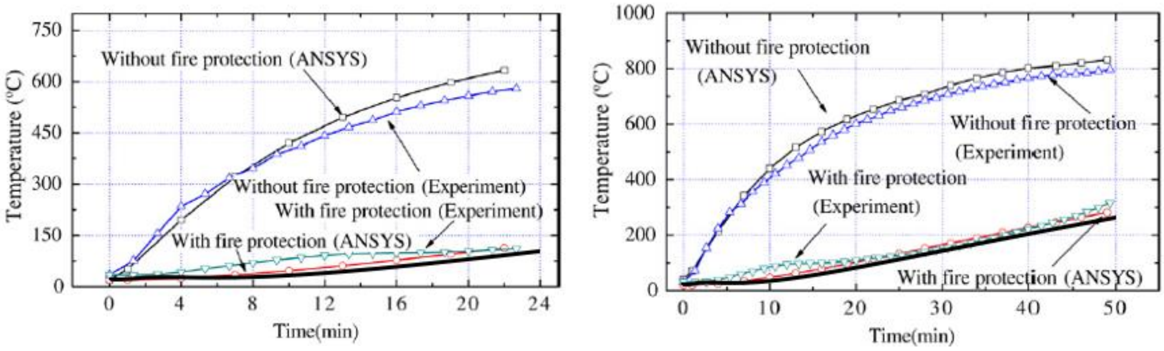

Figure 12 – Comparison of literature reference temperature data of an insulated steel IPE to that found in the ABAQUS FEM simulation. The result is the superimposed black line.

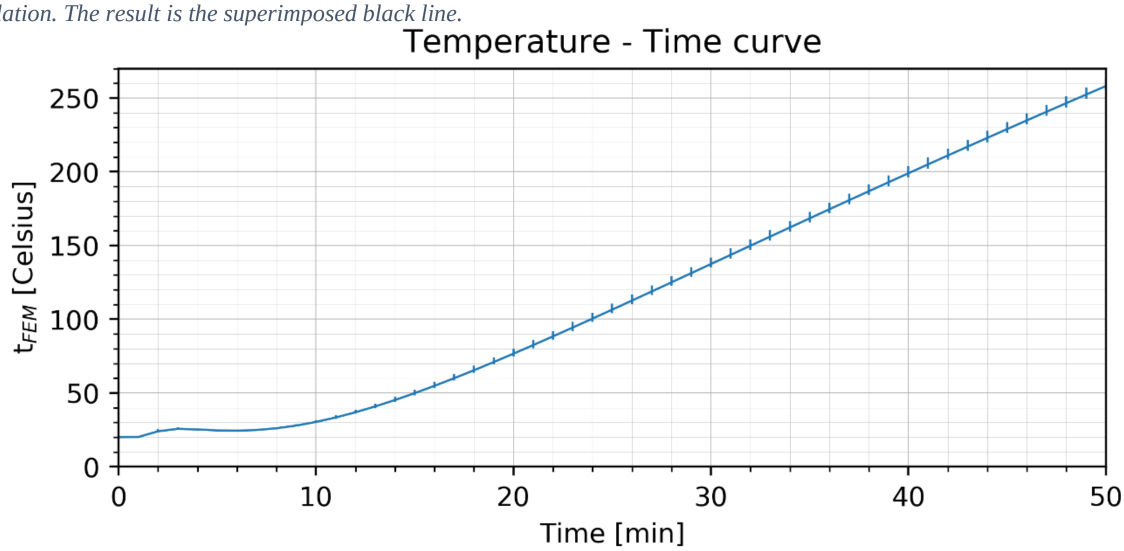

Figure 13 – Temperature - Time curve of thermal FEM analysis following the setup of the literary reference. [28]

After a thorough search, the necessary insulation properties were not found. To obtain more representative temperature data the temperatures are scaled to fit. This is achieved by dividing the thermal conductivity dataset by 1.5. In doing so, the temperature data for the IPE sections fits into the 10% deviation marker, see Figure 14. All figures in this chapter, with the exception of this paragraph, have been computed with the adjusted thermal conductivity.

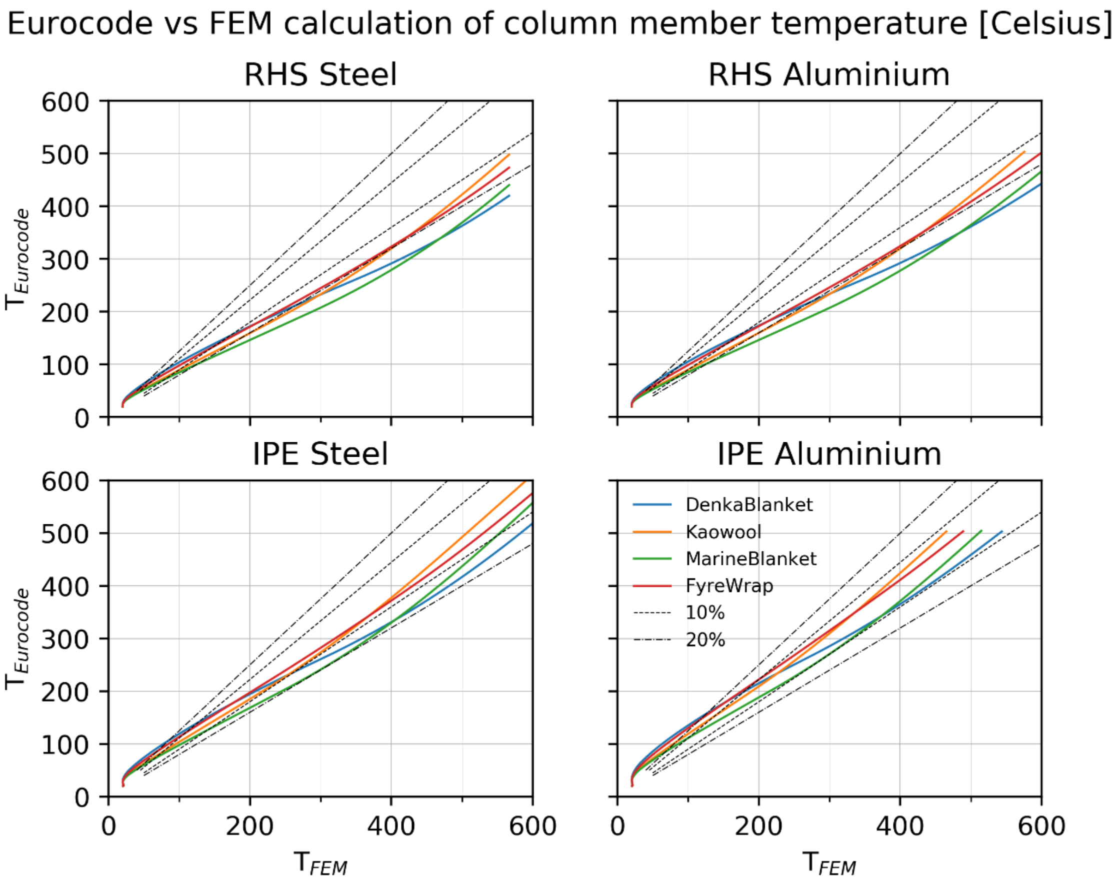

Figure 14 – Comparison of FEM simulation with adjusted thermal conductivity temperature results for cross-section with four-sided heating, with that of the simplified Eurocode equation for several insulation types. FEM temperatures versus EC found temperatures.

4.2.1.2 Thermal gradient

As is to be expected, the maximum occurring temperature in the I-section is halfway its height, in the centre of the web, see Figure 15. Any 'zigzagging' in this figure is due to the averaging of the temperature over the width of the cross-section at height y, which is only a post-processing plotting issue The minimum is found at the flange. This is the case for both aluminium and steel column sections, as well as for different contact resistances between insulation and metal in paragraph 4.2.1.4. The difference in slope of the thermal gradient over the height of the cross-section between aluminium and steel is due to the thermal properties, namely thermal conductivity and the product of specific heat and density. These properties are significantly larger for aluminium, reducing the slope, and thus having a more uniform thermal gradient. Figure 16 shows what the average temperature is over the cross-section and how the minimum and maximum occurring temperature deviate from the average. Note that the temperatures in the figure go beyond the melting temperature of the metals, this is because the FEM program does not consider such limitations during calculation.

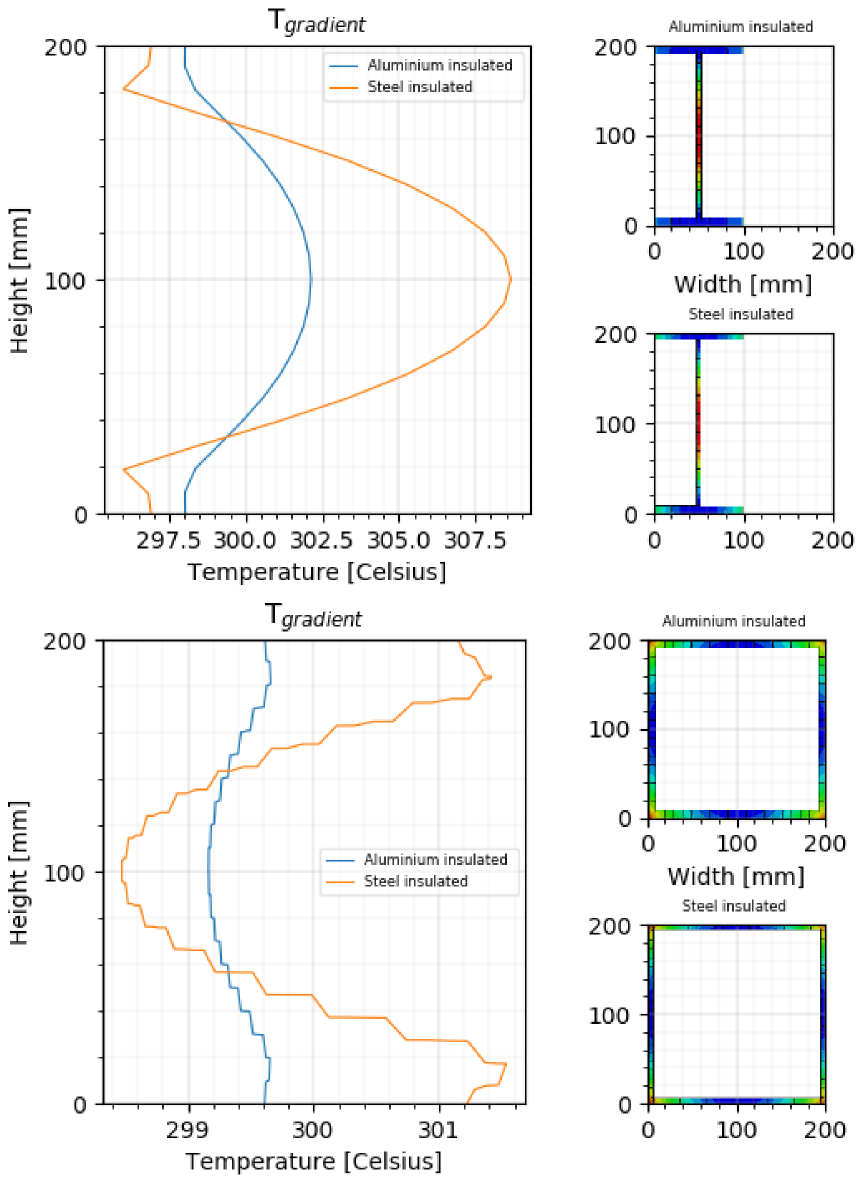

Figure 15 – Temperature gradient over cross-section when taking the mean over the width at height y for a column at overall mean cross-section temperature of 300℃, $t_{IPE,alu}$ = 30min, $t_{IPE,steel}$ = 40min, $t_{RHS,alu}$ = 50min, $t_{RHS,steel}$ = 50min.

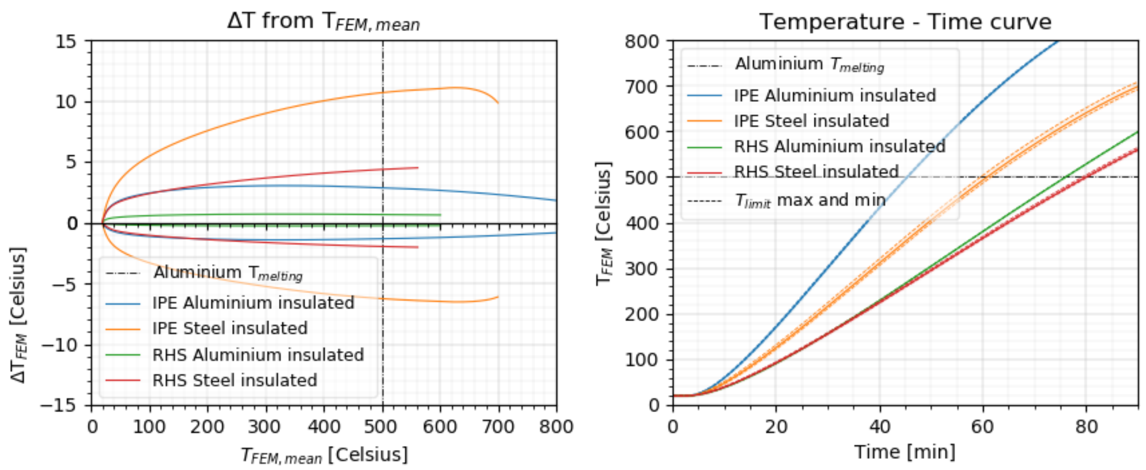

Figure 16 – Minimum and maximum absolute temperature deviation from transient average temperature in cross-section with contact resistance at 200W/m²K between metal and insulation. Left the absolute deviation from the average, right are the errorbars.

4.2.1.3 Sensitivity analysis of mesh density

For computational optimization a mesh density refinement study for the insulation is performed on the RHS section. The element size of the insulation mesh is ranged from 1mm (10%), 2mm (20%), 4mm(40%) to 10 mm(100%). The temperature outcomes are all compared to that found with the finest mesh (10%) to determine the accuracy with a coarser mesh. The deviation is calculated by dividing the result found with a coarser mesh by that at 10%-mesh density and examine the percentile difference. While comparing these values, all other settings are kept constant, such as the mesh density of the member.

As discussed, the metal member itself is compiled out of 8-node linear heat transfer bricks. The accuracy of this setup is evaluated by variating the mesh-density of the member between four different settings, namely 1, 2, 4 and 10 elements over thickness, respectively 5mm, 2.5mm, 1.25mm and 0.2mm. The resultant temperatures are compared by dividing them with the result found for 1 element over thickness (1-5mm). While comparing, all model settings are kept constant, such as the mesh density of the insulation.

Table 5 – Percentile deviation of member temperature from normalised set. For the member mesh compared with values found with a mesh of 1 element or 5mm thickness (coarsest). For the insulation the values are compared to those found with the finest mesh, 10% or 1mm.

	Member mesh			Insulation mesh		
Aluminium	**10**	**4**	**2**	**100%**	**40%**	**20%**
Average [%]	0,157	0,082	0,034	0,245	0,027	0,006
Minimum [%]	0,279	0,026	0,012	0,238	0,026	0,006
Maximum [%]	0,390	0,109	0,026	0,293	0,032	0,007
Steel	**10**	**4**	**2**	**100%**	**40%**	**20%**
Average [%]	0,473	0,228	0,081	2,771	0,083	0,013
Minimum [%]	0,062	0,074	0,020	2,572	0,071	0,011
Maximum [%]	1,410	0,347	0,085	3,570	0,189	0,030

Except for the maximum temperature for a 10-element mesh density over the member and that at 100% (10mm) insulation mesh, the deviation is below a half percent. Given this result, the mesh density of the insulation is set at 20%, which would be defined as 4*tp/2t or as 5 elements over the thickness. For the mesh density of the member a mesh density of one-element is deemed sufficiently accurate in comparison the aforementioned references. A visual of the insulation mesh sizes can be observed in Figure 17.

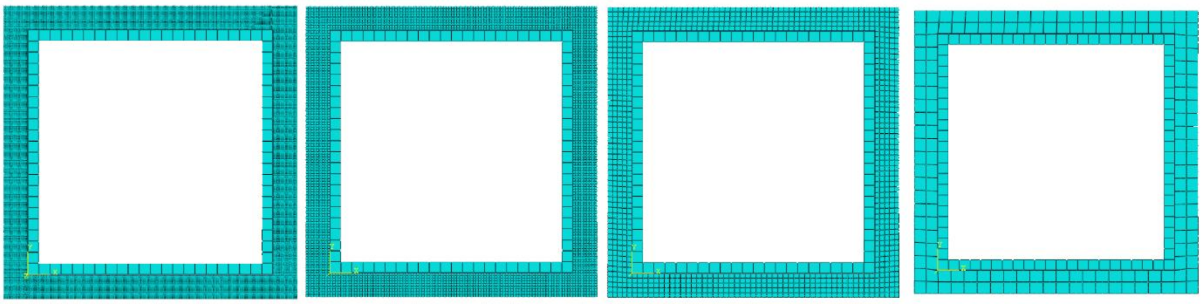

Figure 17 – Rectangular Hollow Section 200x200x9mm. From left to right insulation mesh at 10%, 20%, 40% and 100%.

For the mesh density of the member, an additional consideration must be made. This data has to be implemented into the mechanical analysis. In this case, the translation is done by superimposing the nodal temperatures of the cross-sectional contour on the mechanical shell-model and repeating the 2D temperatures along the length of the shell element for all nodes. Subsequently, Abaqus assigns the temperature data to the mechanical integration points through linear interpolation with the cross-section. In this manner a thermal gradient can be obtained over the height of the cross-section and all integration points in the mechanical analysis have a temperature value. To achieve this, the mesh size has to be the same between thermal and mechanical analysis as the element types differ. This point is addressed in the next chapter.

4.2.1.4 Sensitivity analysis of contact definition

As discussed before, contact between insulation and member is defined as contact through surface-to-surface discretization between master and slave surface. At this interface, heat transfer occurs between the insulation and the member. The thermal property at this interface is not specified in theory or literature references. Thermal resistance between materials is dependent on surface smoothness and the pressure between surfaces. This property has been evaluated between concrete and steel and been approximated at 200W/m^{2}K [23][24][32]. For simplicity sake, the thermal resistance at the interface can be taken at a unit value of 1. This would be an overestimation of the actual thermal resistance. To determine the effect of the thermal resistance, the unit value is compared to a situation where the interface with insulation is set to be 200W/m^{2}K. From this calculation it is evident that the difference in member temperature, as compared for both heat transfer resistance values, increases with increasing temperature over a range of 20-700℃. The maximum absolute percentile difference is expressed in Table 6, defined as the member temperature found with 200W/m^{2}K divided by the member temperature with unit value multiplied by a 100%. Hence a value closer to 0% means the member temperatures of the two cases are the same. The difference between the two thermal resistance values does not exceed 1.5%, therefore the thermal resistance is generally set at 200W/m^{2}K.

Table 6 – Maximum percentile difference of member temperature with variating thermal resistance between surfaces.

θ%-Deviation between 200W/m^{2}K / unit		**Minimum [%]**	**Maximum [%]**	**Average [%]**
Column	RHS Aluminium	1,175	1,181	1,176
	IPE Steel	1,426	1,267	1,376
	IPE Aluminium	1,281	1,258	1,280
Beam facing	RHS Aluminium	0,979	1,045	1,016
3 sides fire	IPE Steel	0,861	1,196	1,009
	IPE Aluminium	1,048	1,202	1,128
Beam facing	RHS Aluminium	0,788	0,830	0,806
1 side fire	IPE Steel	0,527	1,204	0,874
	IPE Aluminium	0,841	1,119	0,972

4.2.2 Beam: a three-sided fire simulation

In this case, the model is subject to three-sided heating. The geometry of the model is altered, as a concrete slab is simulated on top of the flange of the metal member and the insulation is adjusted to fit the remaining circumference, see Figure 18. For the cross-sections the same measures apply as in Table 3. The mesh size is set as in the previous paragraph. The material properties are as described in chapter 3 Literature study & theoretical background. Contact with the concrete parts is modelled with the aforementioned thermal resistance value of 200W/m^{2}K. For the insulated cases contact resistance between floor-member-insulation is set at 200W/m^{2}K.

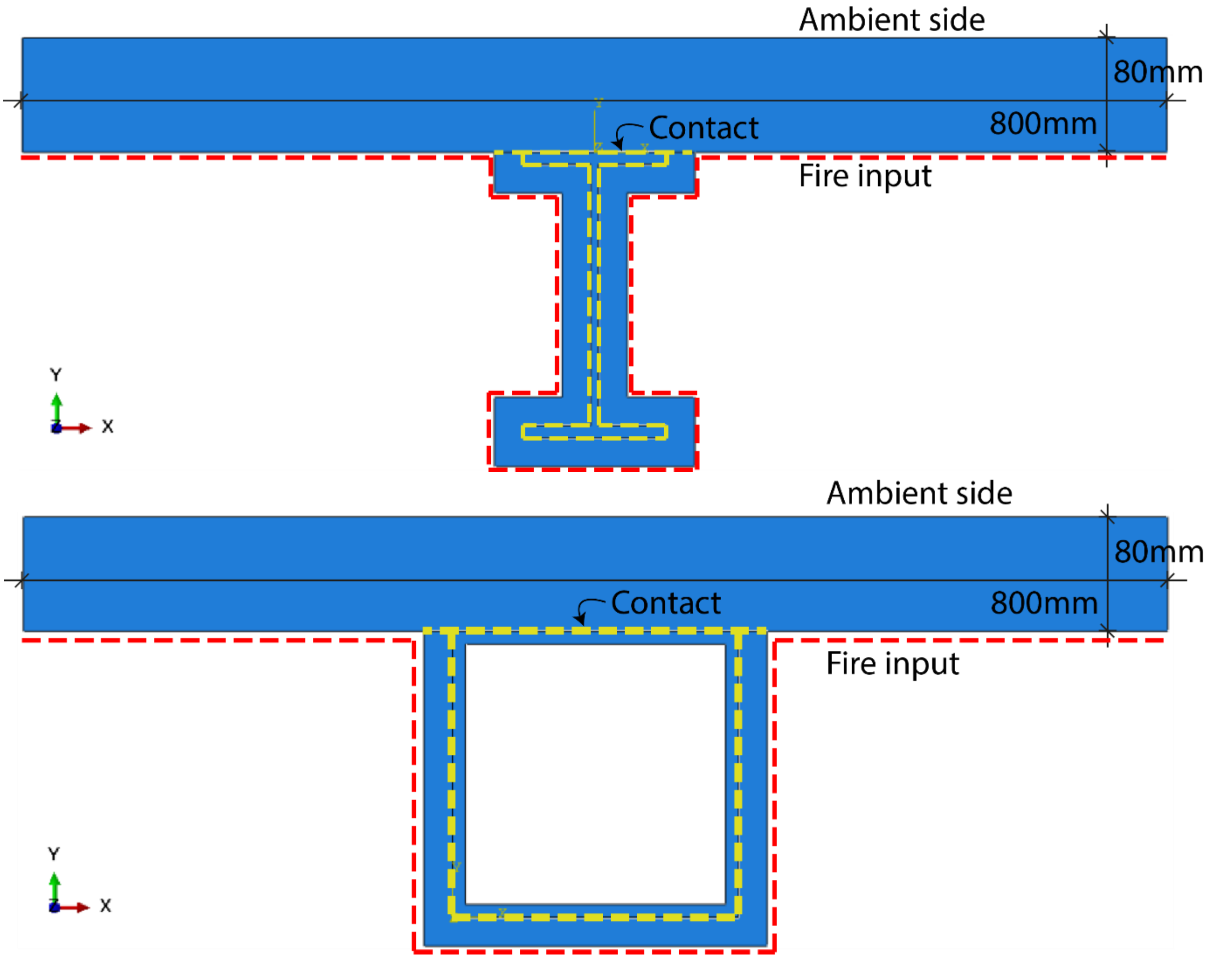

Figure 18 – Geometry of beam with a concrete floor slab on the top flange and 3-sided heating.

However, aluminium members are often used because of their light weight and slenderness attributes and a flooring system often shares these specifications. Such floors are not made of highly insulating concrete material, but more often consist of metal with a thin layer of concrete or other plate material to mechanically tie it together and fit comfort criteria [57].

Examples of lightweight flooring would have a density below 350kg/m^2 such as Slimline, IDES and Starframe systems [57]. Such systems are combinations between aired openings, insulating material, steel or aluminium sheets and beams, and a concrete layer. For simplicity, the properties of such a system is regarded as a composition of the mean value over the temperature range of the material property due to the percentile contribution of each material to the system. For the floor this results into the material properties as expressed in Table 7. Each material's percentile contribution to the systems make-up is considered, as to calculate a weighted material property. These values are input for an alternative to that of the concrete flooring with the same geometrical setup. This is a very simplified static rendition for a floor, the evident differences in the material properties and thus the resulting member temperatures are significant enough to relay the effect of a different system.

Table 7 – Approximation of the material properties of a lightweight floor system as a combination of the mean value of the individual material following its percentile makeup.

Mean values	**Percentile**	***λ* [W/mK]**	***ρ* [kg/m3]**	***c* [J/kgK]**
Air	10%	0,025	1,225	1006
Concrete	15%	0,87047	2176	1045
Aluminium	15%	207,325	2700	1005,5
Insulation	60%	0,12	96	820
Total	100%	31,30382	789,1225	900,175

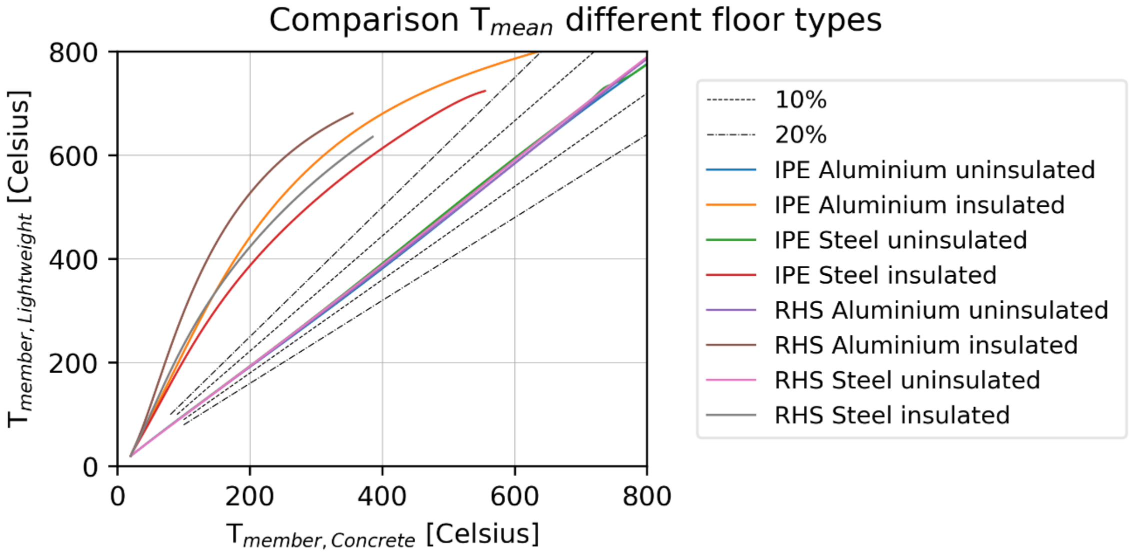

Figure 19 – Comparison of the average temperature of the aluminium member (either an RHS or IPE) with different floor types, on the X-axis a concrete floor system and on the Y-axis that of the lightweight floor system described in Table 7.

As is depicted in Figure 18, the insulation does not encompass the floor for the insulated cases. For non-insulated cross-sections it is apparent that heating of the member is practically identical for different floor systems, see the straight line in Figure 19. The shift however for a lightweight floor system with an insulated beam indicates, that for a higher value of the thermal conductivity of the floor, the temperature of the member is influenced. which is the situation for both insulated cases, and due to the higher thermal conductivity of the lightweight floor in comparison to the concrete floor, the metal member heats quicker. This effect is also evident when reviewing the minimum and maximum deviation from the average temperature of the member in the right errorbars of Figure 20 & Figure 21, which has a much larger range than in Figure 16. This is complemented by the fact that ambient conditions are applied on the non-heated side. Therefore, due to convection, a larger thermal gradient is possible. This is especially true for the steel members, which has a smaller $\frac{k}{c * \rho}$ factor than aluminium, thus having a larger difference between minimum and maximum temperatures.

What is most curious however, is that for the insulated cases – where only the member is insulated – the thermal gradient is thusly affected that the maximum temperature can occur at the top flange. Apparently in these cases, the floor heats much more quickly than the insulated member, therefore more heat is transferred through this way instead of from fire to insulation to member. This reveals a reversed thermal gradient, maximum at the ‘ambient’ side and minimum at the fire side in Figure 22 & Figure 23.

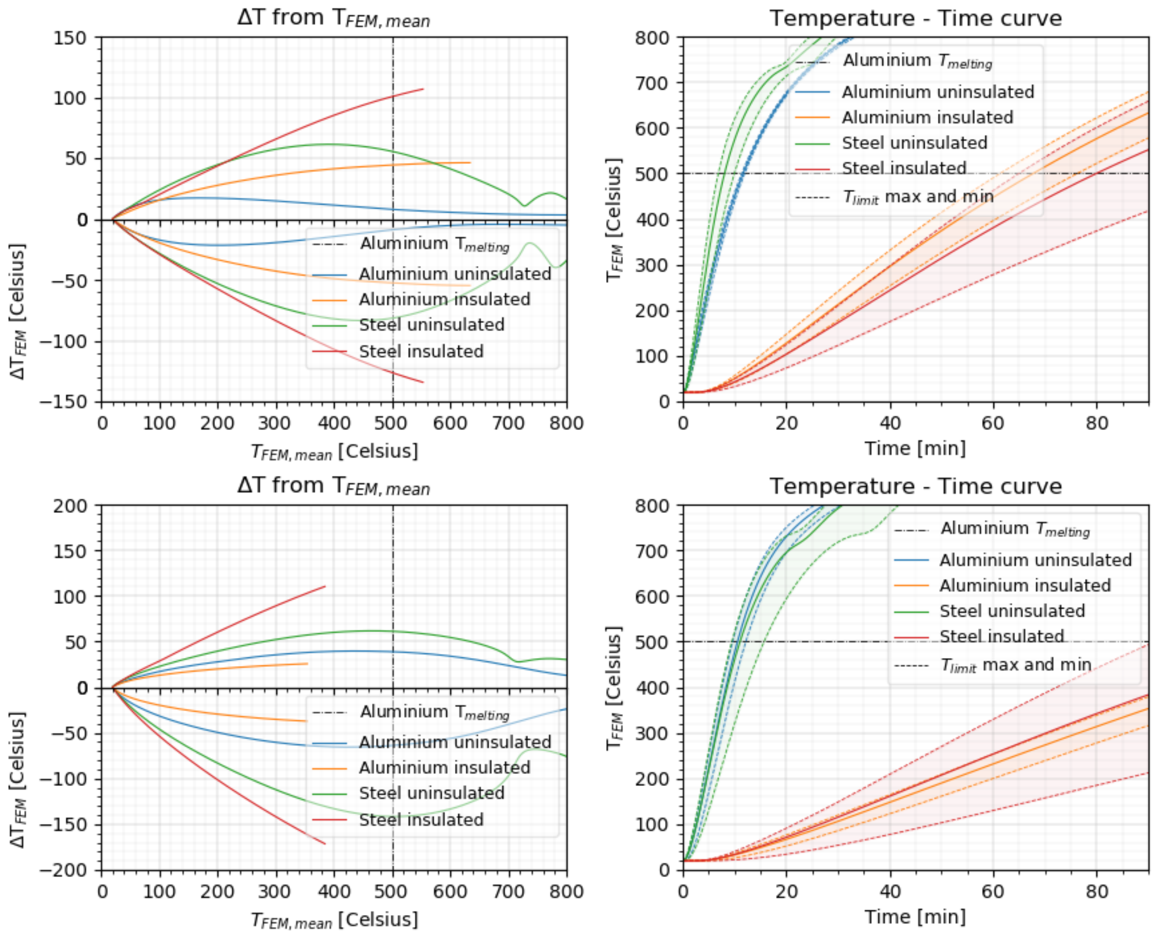

Figure 20 – Transient mean temperature curves for the three sided heated beam and the minimum, maximum deviation from that temperature occurring in the cross-section. Left the absolute deviation from the average, right are the errorbars. From top to bottom: IPE with concrete floor, RHS with concrete floor.

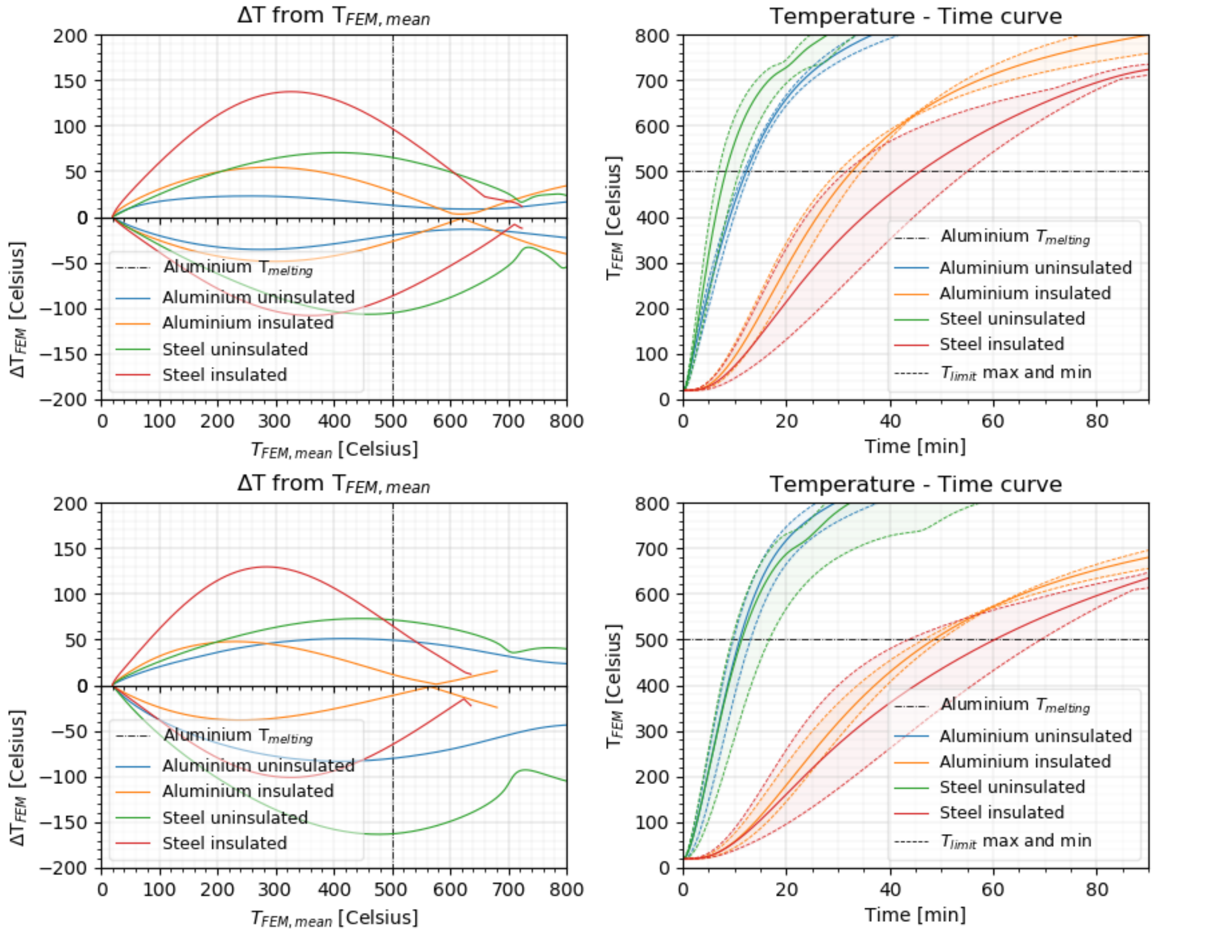

Figure 21 – Transient mean temperature curves for the three sided heated beam and the maximum and minimum temperature deviation. From top to bottom: IPE with lightweight floor, RHS with lightweight floor.

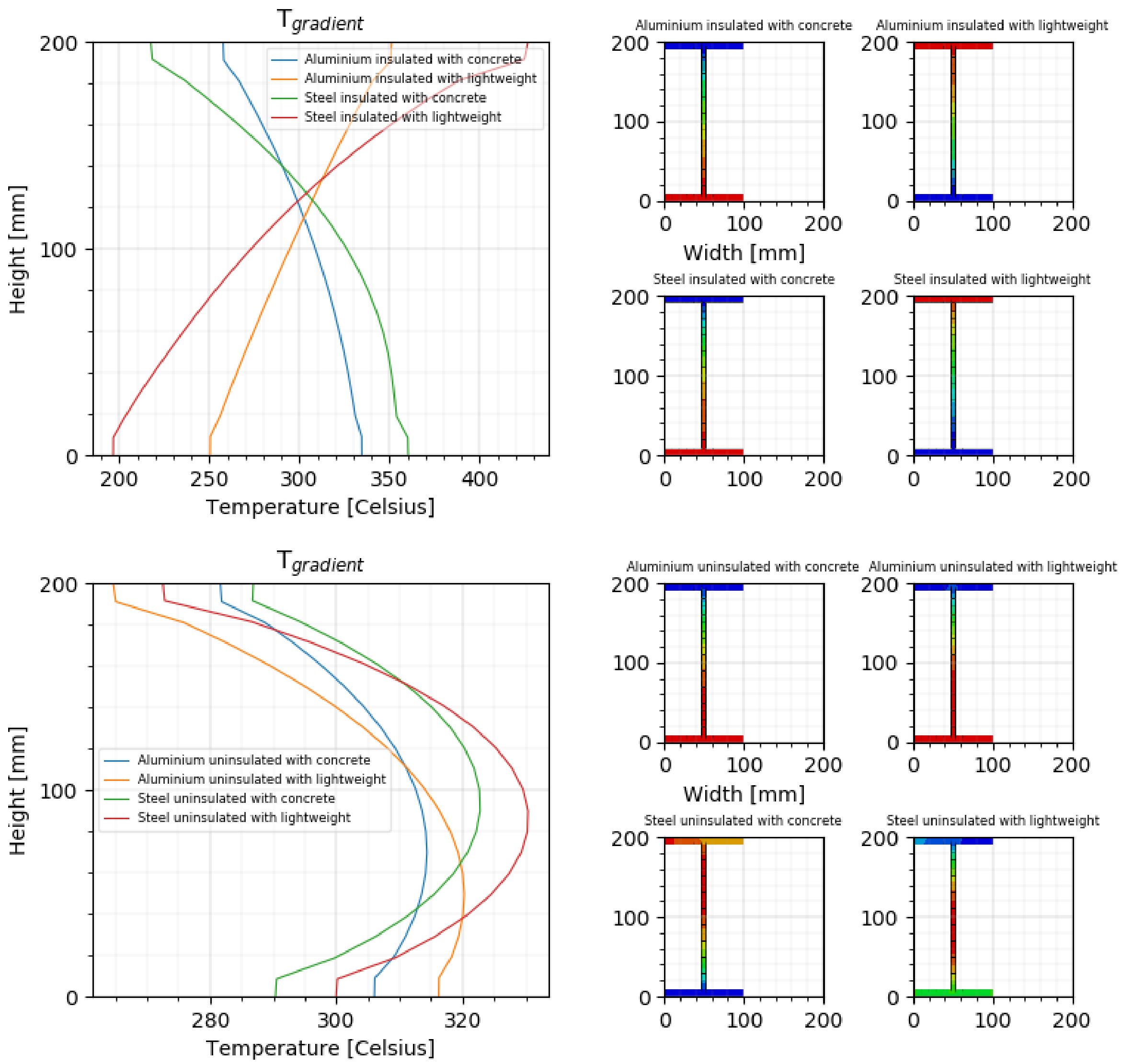

Figure 22 – Temperature gradient over three sided heated IPE beam section with flooring on top for insulated (top) and uninsulated (bottom) case as in Figure 18. $t_{uninsulated}$ = 7min, $t_{insulated,concrete}$ = 45min, $t_{insulated,lightweight}$ = 20-30min.

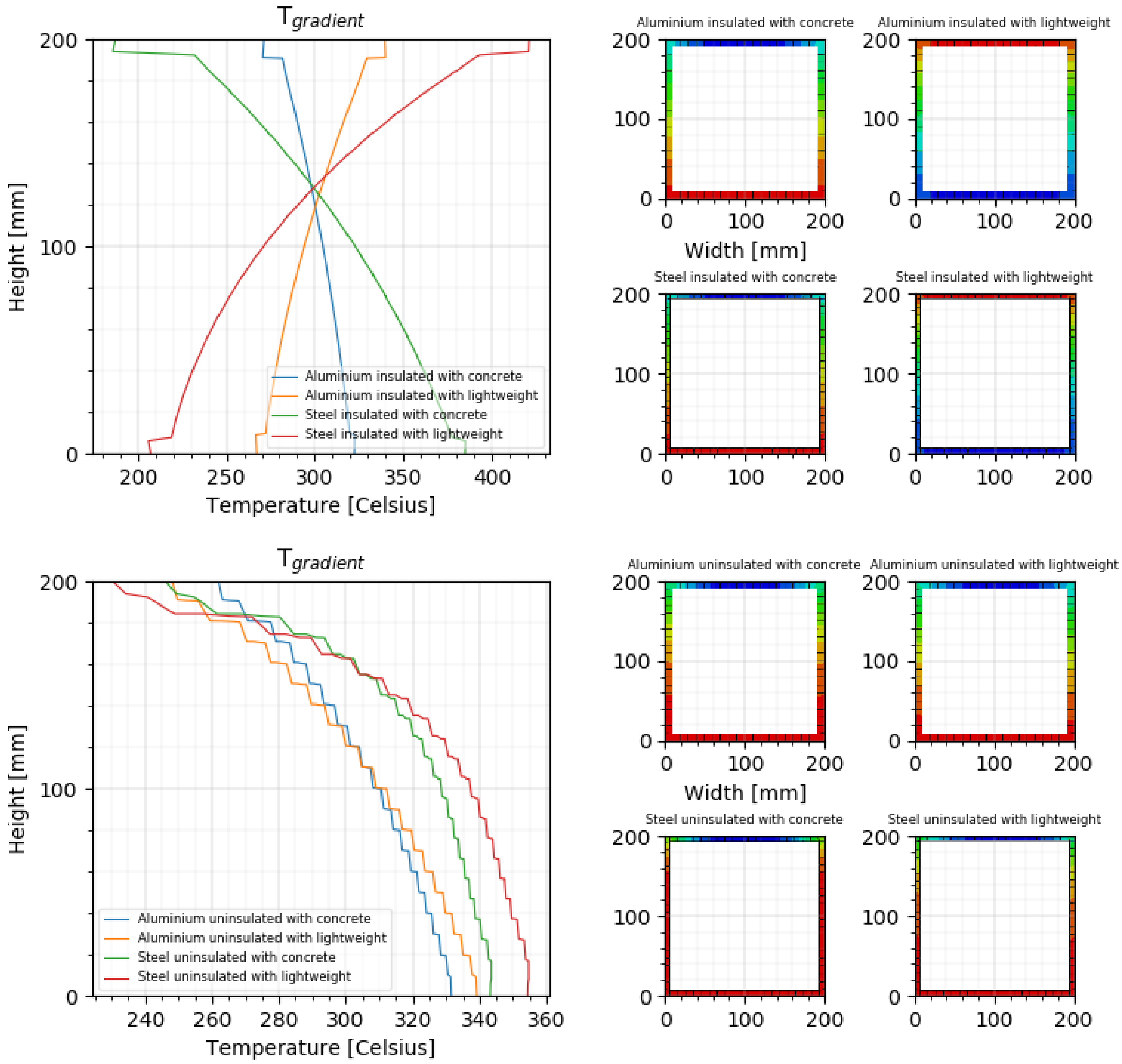

Figure 23 – Temperature gradient over three sided heated RHS beam section with flooring on top for insulated (top) and uninsulated (bottom) case as in Figure 18. $t_{uninsulated}$ = 7min, $t_{insulated,concrete}$ = 70min, $t_{insulated,lightweight}$ = 20-30min.

4.2.3 Integrated beam: a one-sided fire simulation

Given that aluminium sections are often applied in tandem with lightweight floor systems where the structural height is minimised by having floor and beams in the same layer, an additional model setup is considered. An alternative model is that of the integrated beam wherein only the bottom part of the cross-section of both the IPE and RHS would be exposed to elevated temperatures. Contact with other elements is specified as having a thermal resistance equal to 200W/m^2K, same as before.

In this case it is assumed that a floor slab is placed on the bottom flange of the geometry. For an IPE section this can be achieved in a straightforward fashion. For the RHS, the section is slightly altered as to have external ledges as bottom flange for the slab to lay on. These ledges are 16mm in length on either side of the RHS and make the total width 232mm. Such a change on the geometry would be most peculiar when working with steel but for aluminium, extrusion makes this a feasible adjustment. The model is specified as visible in Figure 24.

In this design there are several alternatives to consider. In Figure 24 an insulated cross-section with a concrete floor is visualized (variant 1), however in some cases similar sections would not be insulated (variant 2), and given that aluminium is a lightweight material a floor with the same attributes such as described Table 7 would be more appropriate (variant 3).

Exploring these variations reveals the effect that the floor system has on the heating of the member. Evidently a concrete floor is a capable insulator, which explains the relatively low aluminium member temperatures for the insulated case in comparison to that of the lightweight floor in Figure 25. This result is further supported by the relationships as sketched in Figure 26 & Figure 27. As expressed previously, the thermal gradient is in these cases even larger, Figure 28 & Figure 29. This fits with the amount of heated surface versus that with facing ambient convection and the respective thermal conductivity of the materials.

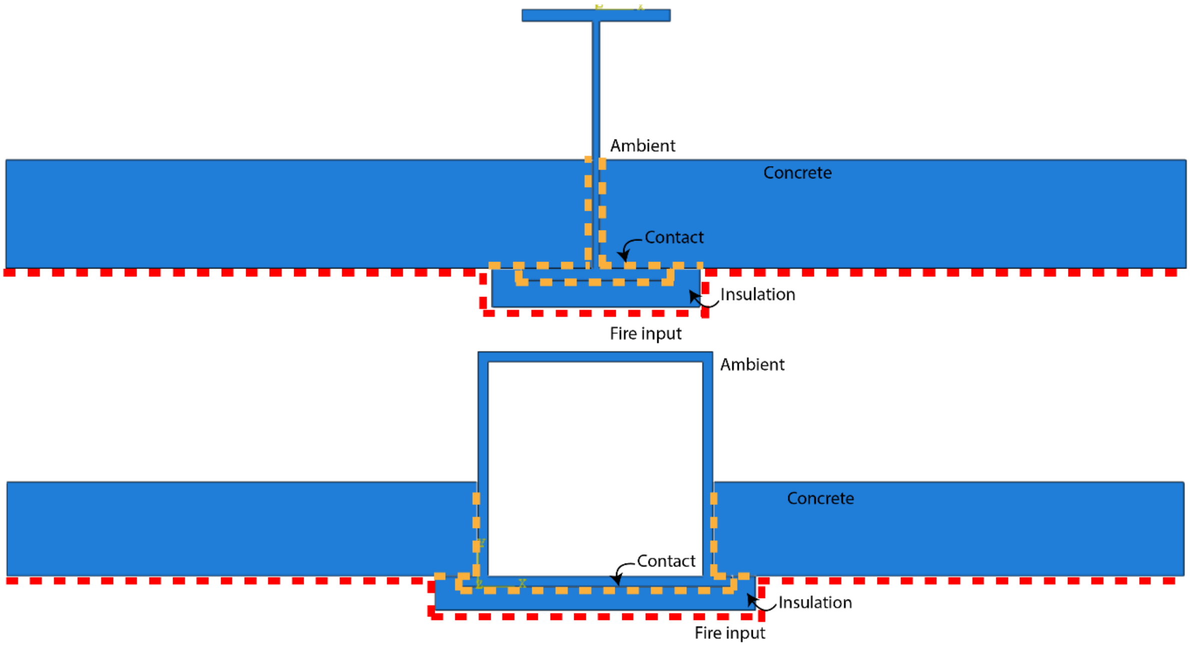

Figure 24 – Geometry of model subjected to a one-sided fire load, total width of model with IPE is 800mm for RHS is 1000mm.

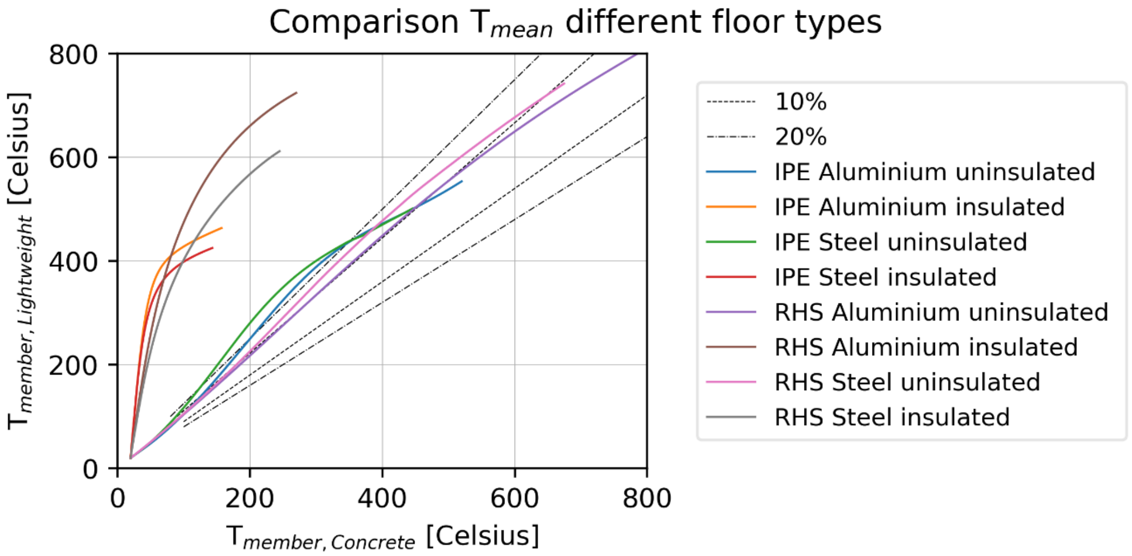

Figure 25 – Member temperatures for an integrated beam subject with a floor slab, concrete versus a lightweight floor system.

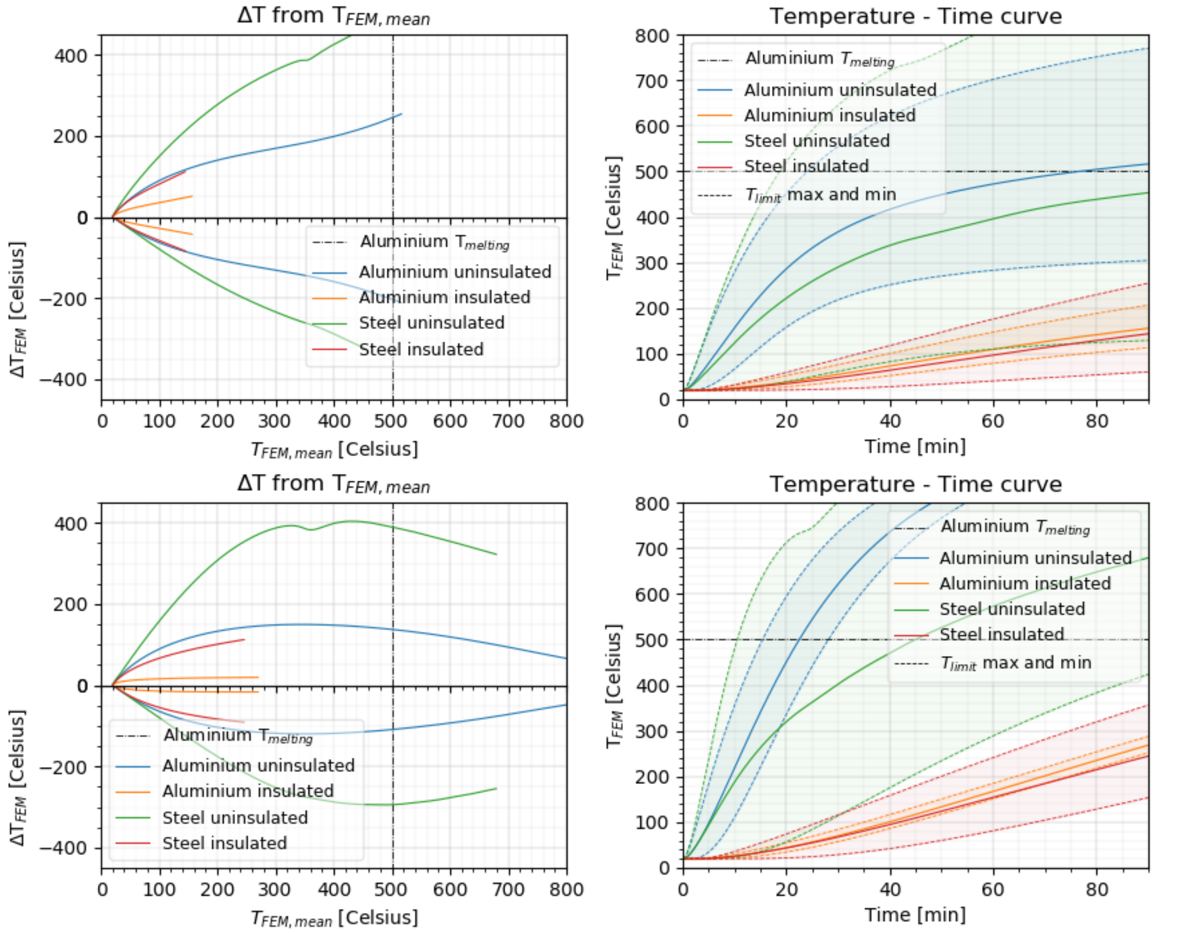

Figure 26 – Transient mean temperature curves for a one sided heated beam and the minimum, maximum deviation from that temperature occurring in the cross-section. Left the absolute deviation from the average, right are the errorbars. From top to bottom: IPE with concrete floor, RHS with concrete floor.

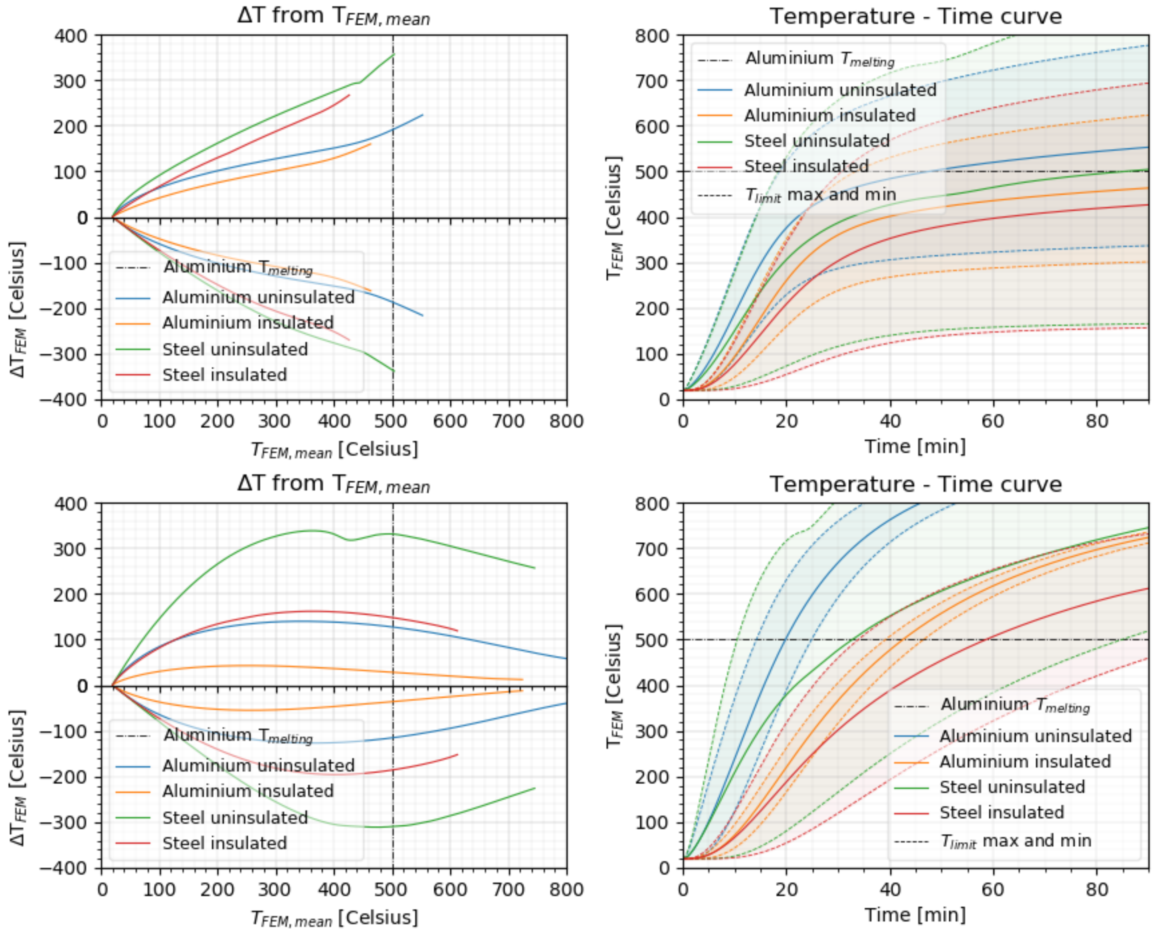

Figure 27 – Transient mean temperature curves for a one-sided heated beam and the maximum and minimum temperature deviation. Left the absolute deviation from the average, right are the errorbars. From top to bottom: IPE with lightweight floor, RHS with lightweight floor.

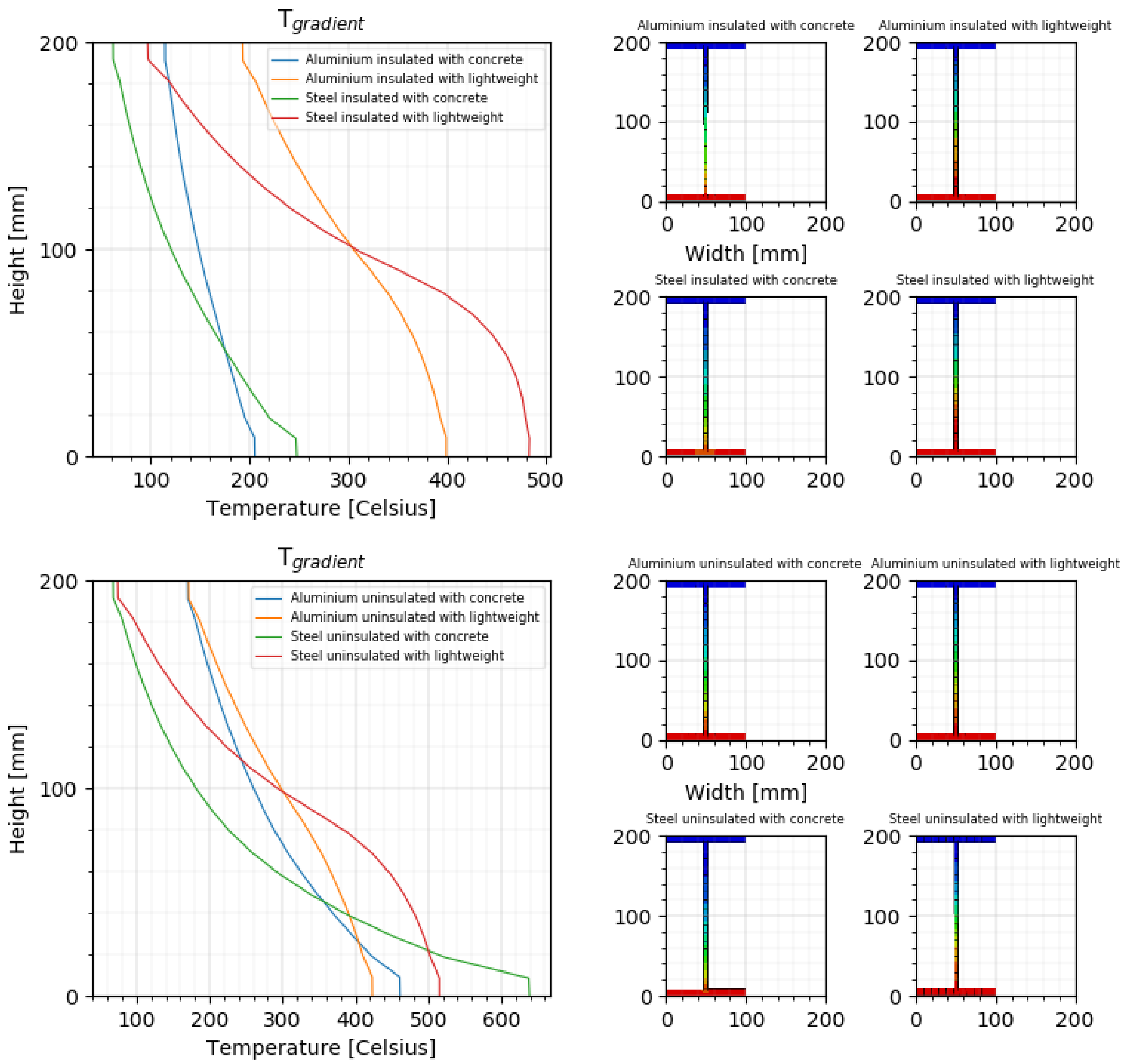

Figure 28 – Temperature gradient over one side heated IPE beam section with flooring for insulated (top) and uninsulated (bottom) case as in Figure 24. $t_{uninsulated,concrete}$ *=20min,* $t_{insulated,concrete}$ *=90min,* $t_{uninsulated,lightweight}$ *=10min,* $t_{insulated,lightweight}$ *=25min.*

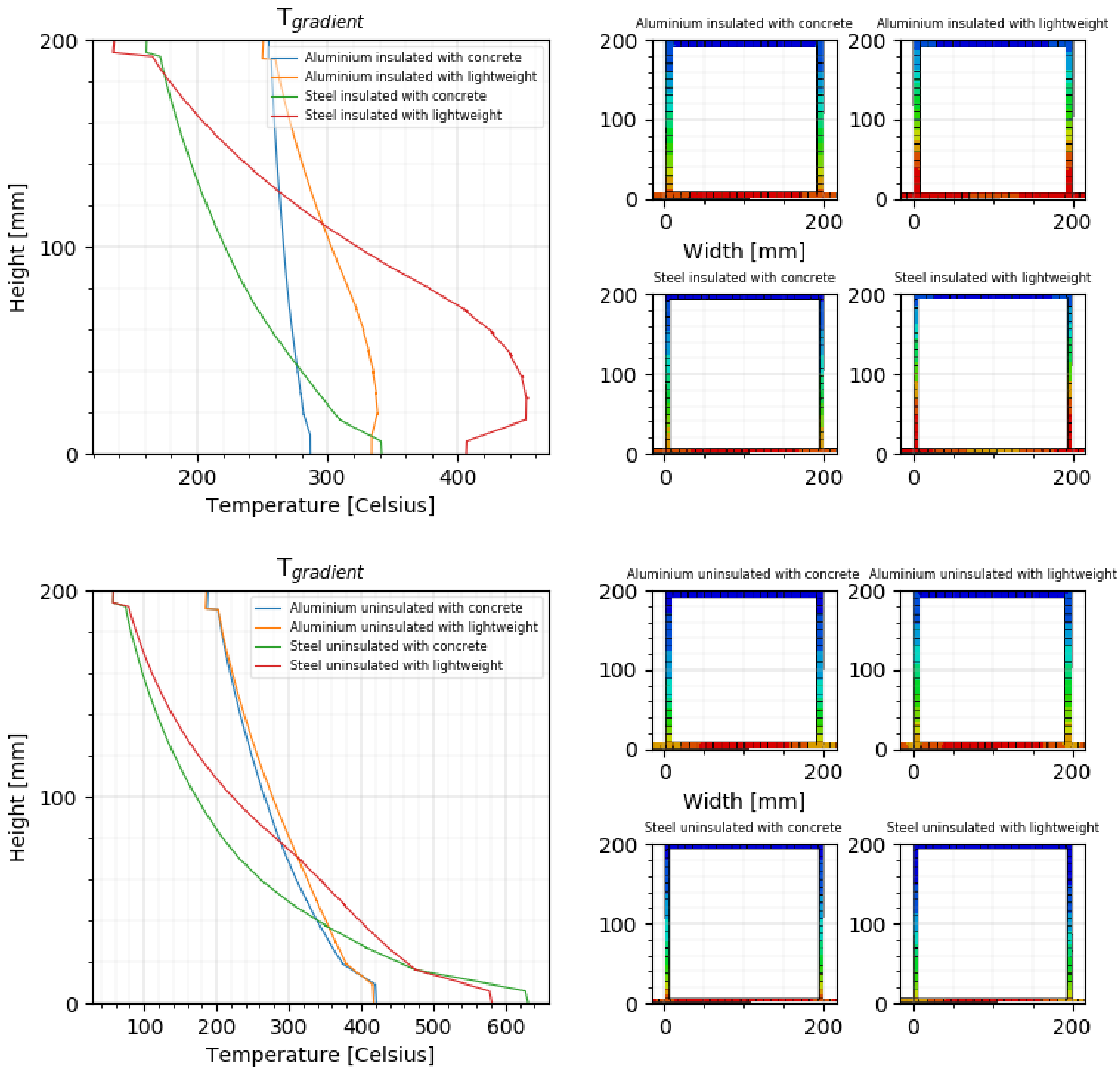

Figure 29 – Temperature gradient over one side heated RHS beam section with flooring for insulated (top) and uninsulated (bottom) case as in Figure 24. $t_{uninsulated,concrete}$ = 10min, $t_{insulated,concrete}$ = 90min, $t_{uninsulated,lightweight}$ = 15min, $t_{insulated,lightweight}$ =25min.

4.2.4 Alternative lightweight floor – sandwich panel

At first glance, the lightweight floor description is indicative when working with less insulated slabs. However sandwich panels are comprised of layers of different stacked materials. To evaluate the effect of such a floor structure, an additional model is made, see Figure 30 with the material properties as described in Table 8. The resulting temperature shows a slightly reduced heating rate as compared to the earlier mentioned lightweight floor, as visible in Figure 31 too Figure 36.

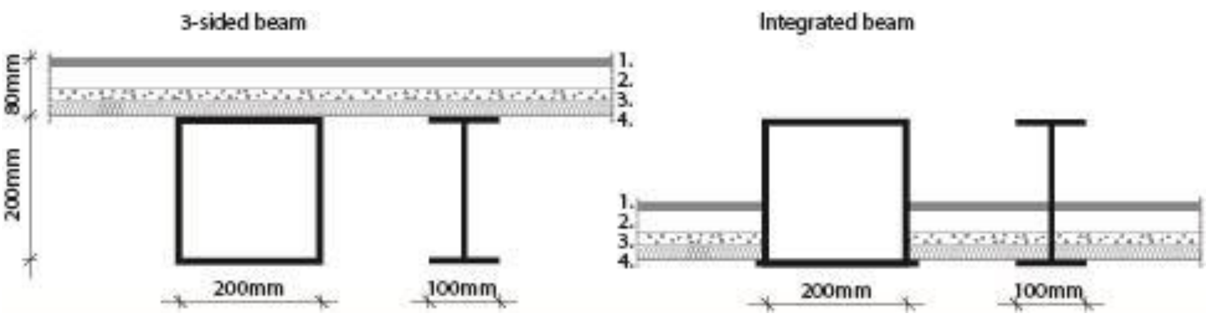

Figure 30 – Cross-sectional view of the three-sided beam and the integrated beam setup with alternative layered flooring.

Table 8 – Alternative lightweight floor setup.

Material	Layer	$\lambda\ [W/mK]$	$\rho\ [kg/m^3]$	$c\ [J/kgK]$
Aluminium	1	207	2700	1005
Air & Aluminium	2	20.7	271	1005
Concrete	3	0.8	2176	1045
Insulation	4	0.12	96	820

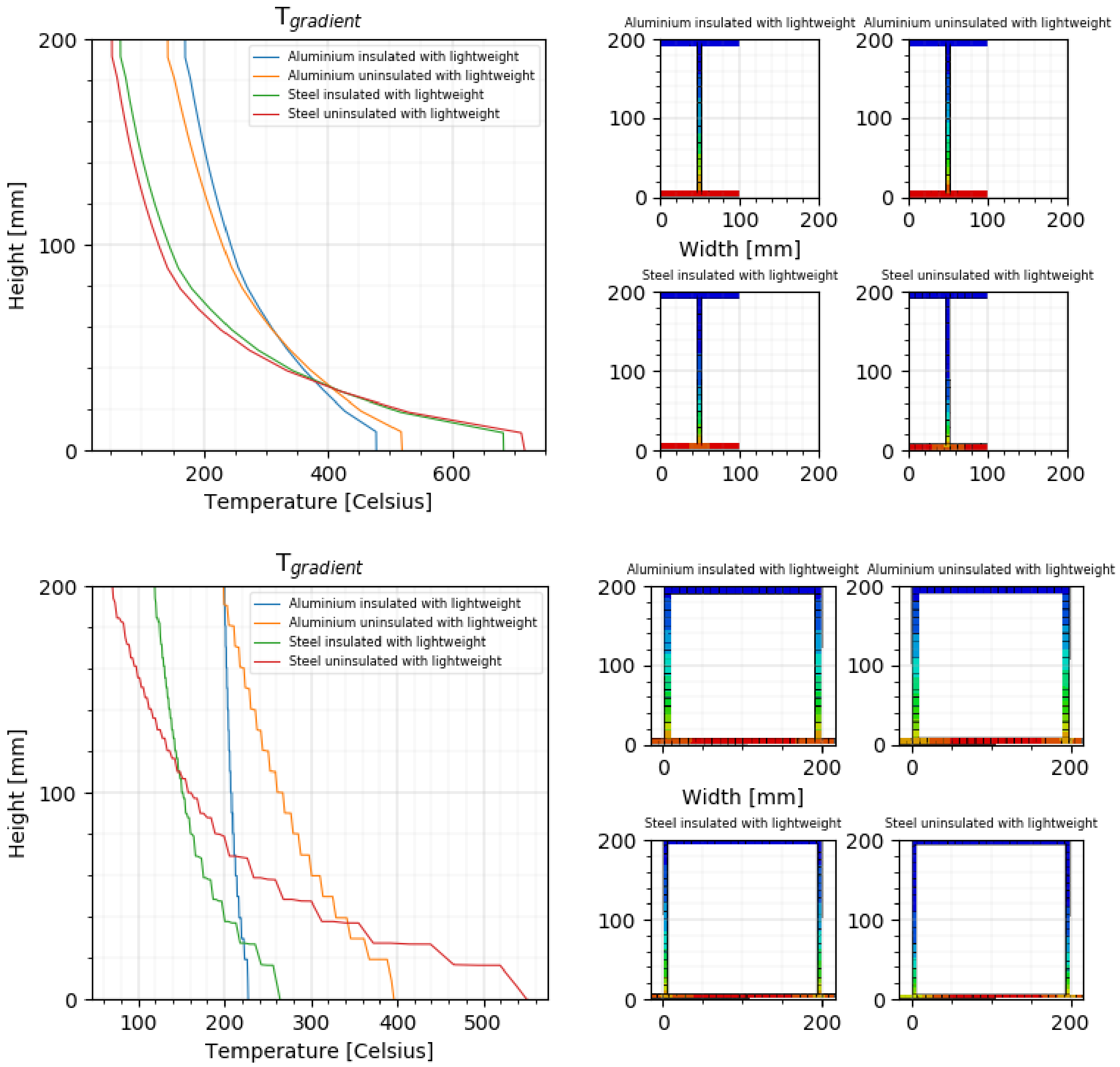

Figure 31 – Thermal gradient of an integrated beam with the alternative lightweight flooring. Time at 40 minutes. From top to bottom an IPE profile and an RHS profile.

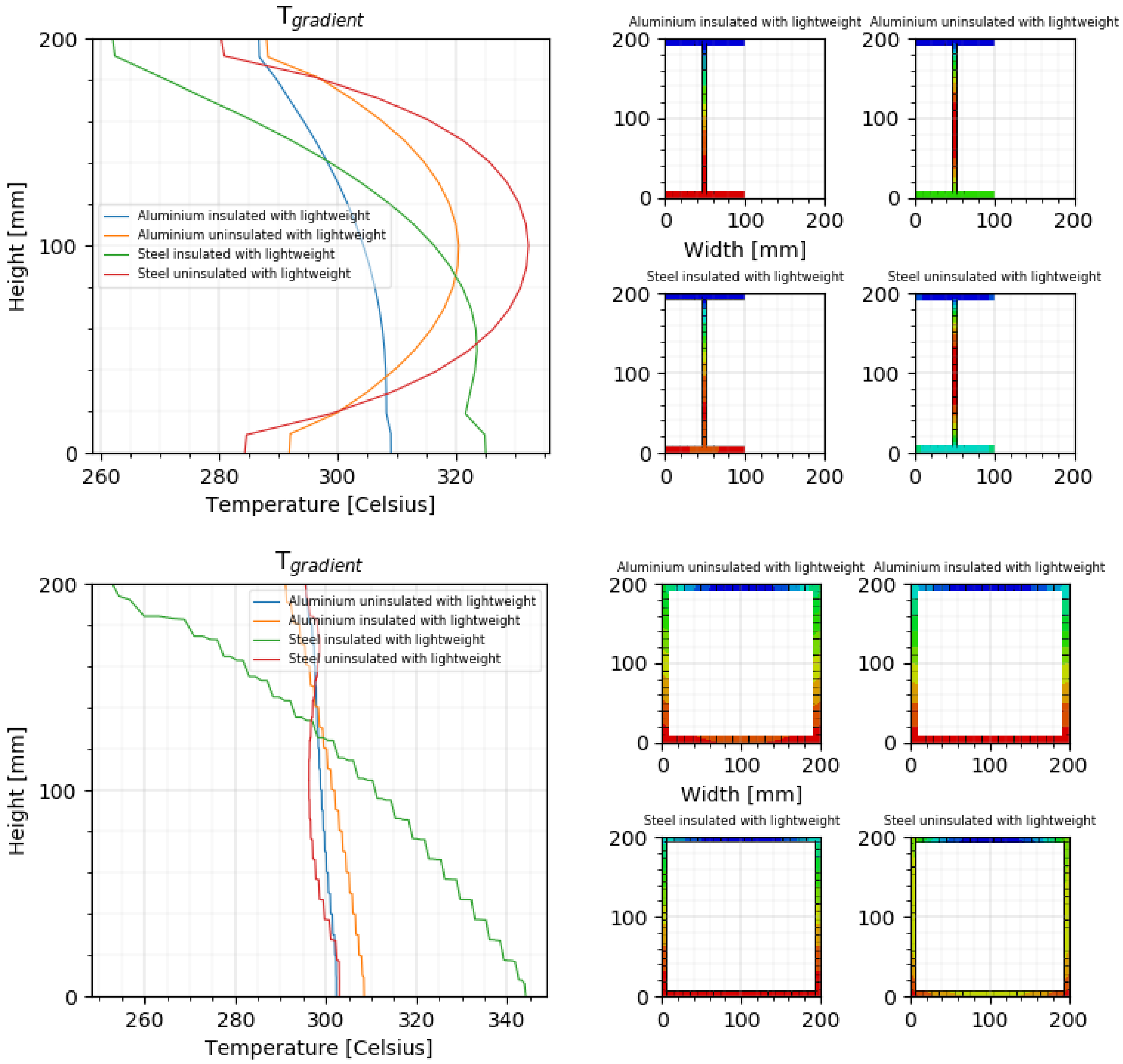

Figure 32 – Thermal gradient of a beam with an alternative lightweight floor for a beam facing three sided fire. Time at 40 minutes. From top to bottom an IPE profile and an RHS profiles.

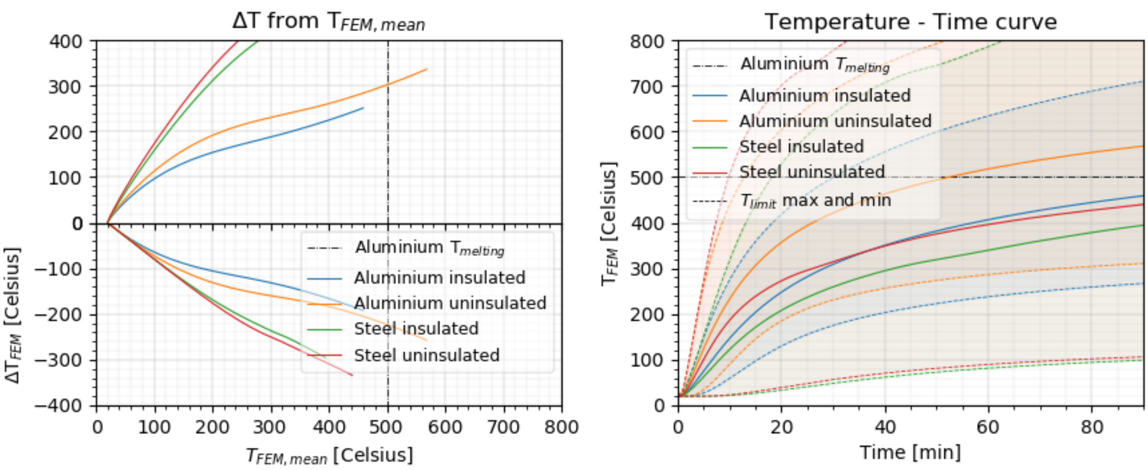

Figure 33 – Temperature time curve for an integrated IPE beam with an alternate lightweight floor.

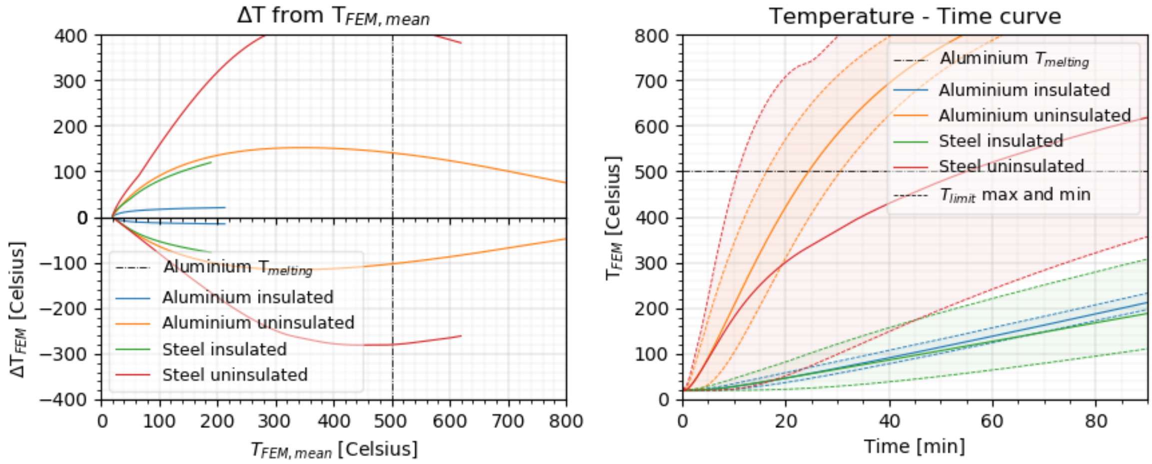

Figure 34 – Temperature-time curve for an integrated RHS beam with an alternative lightweight floor.

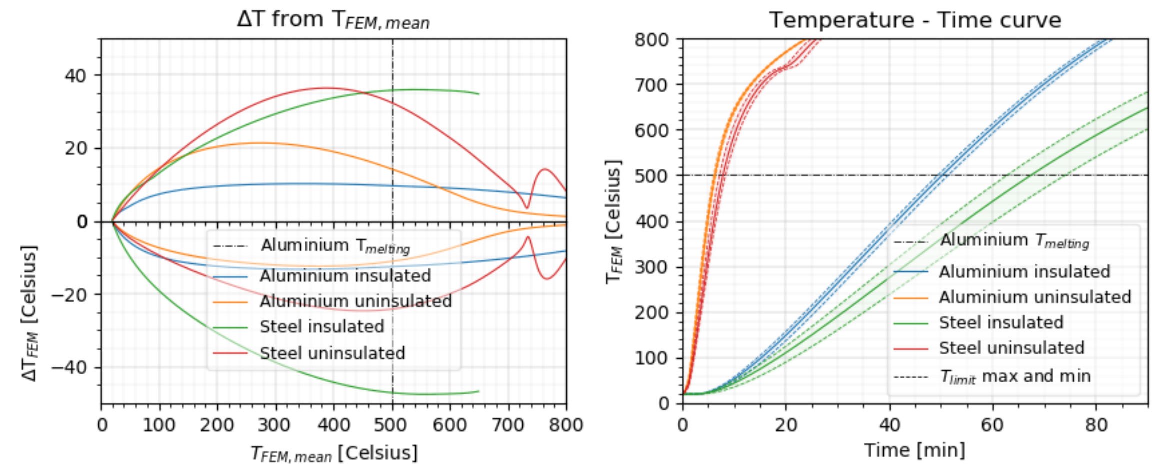

Figure 35 – Temperature time curve for an IPE heated from three sides with an alternate lightweight floor.

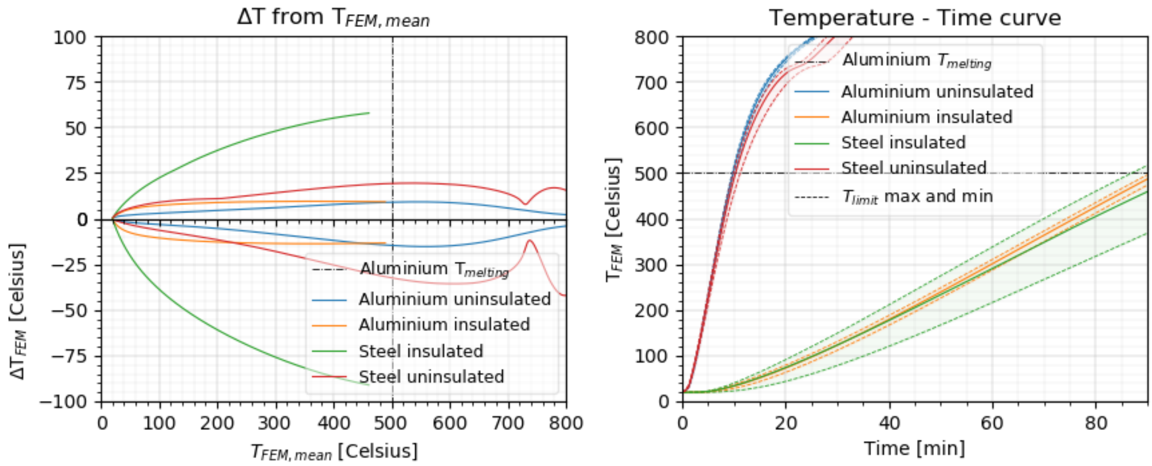

Figure 36 – Temperature time curve for a RHS beam heated from 3 sides with an alternate lightweight floor.

4.2.5 Intumescent paint

Instead of hardboard, blankets or other fibrous materials, metals are increasingly covered with intumescent paints for fire protection. The difficulty in modelling such a material is that the thickness of the coating has a thermal response. The material properties are dependent both on the thickness and temperature of the coating. The most accurate methodology to describe the behaviour would require a coupled transient heat transfer analyses. After the initial thermal calculation, the geometry must be adjusted, hence requiring remeshing and interpolation of the nodal temperatures from the previous calculation step. This process must be repeated every increment until the final time step is reached. This process could be simplified by considering the temperature range at which the foaming occurs and only performing the coupled analysis for this temperature frame. In [58] this range is expressed as 120-240°C of the steel beam, for which before and after the thickness could be assumed constant and stable. Alternatively, which would fit with the previous thermal analysis setup, the thickness of the intumescent coating can be modelled as constant at maximum expansion but adjusting the thermal properties

The numerical setup in [58] produces data which fits with the experimental results as described in the paper. The commercial water-based intumescent coating is described with constant values $c_p = 1200\ J/kgK$; $\rho_p = 200\ kg/m^3$; $\varepsilon_m = 0.95$; with an initial thickness of $t_p = 1500\ \mu m$, and the thermal conductivity calculated following equation *(22)*.

$$\lambda_p = t_p * \frac{V}{A_p} * c_{m,\theta} * \rho_m * \frac{1}{(\theta_{fire,t} - \theta_{m,t}) * \Delta t} * \Delta\theta_{m,t} \qquad (22)$$

Using the alternative approach, the thickness of the coating is set at a constant value, which would be at a maximum expansion of 45mm. Furthermore, the relation between temperature and thermal conductivity is assumed to be linear. The thermal conductivity can be approximated by considering the values found in [58] for the thickness of the coating, corresponding to temperature and thermal conductivity at 5 distinct points – start, coating activation, reaching minimal thermal conductivity, coating reaches maximum expansion and the end point. Considering the dependencies in equation *(22)*, the temperature dependent thermal conductivity for the coating in combination with an aluminium member is approximated following $\lambda_{p,aluminium} = \lambda_{p,steel} * \frac{\left(\frac{V}{Ap}\right)_{steel}}{\left(\frac{V}{Ap}\right)_{aluminium}} * \frac{c_{\theta,steel}}{c_{\theta,aluminium}} * \frac{\rho_{steel}}{\rho_{aluminium}}$.
The last term in *(22)* is neglected, as the difference between steel and aluminium in this regard would be relatively small in comparison to the aforementioned terms. This conclusion can be drawn from the earlier found temperature curves as the variables in the last term are either the same or within the same order of magnitude between the materials. Given all these considerations, the equivalent thermal conductivity in relation to temperature is as described in Figure 37.

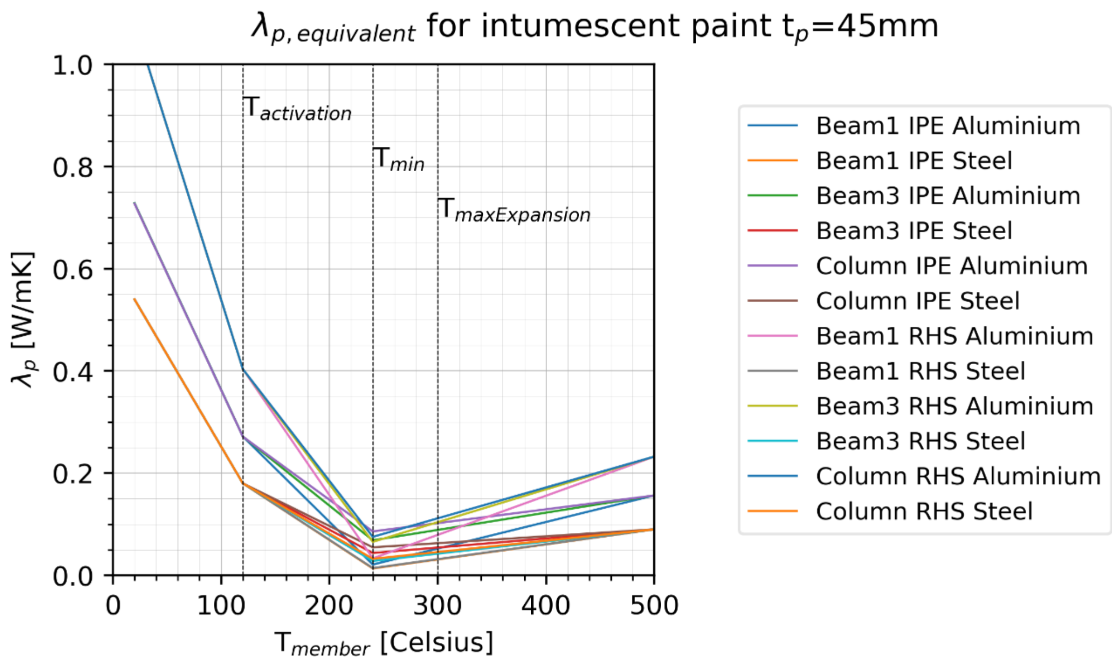

Figure 37 – Equivalent thermal conductivity of intumescent paint with constant thickness for different model descriptions, namely a section heated from all sides (column), three sides (beam3) or one side (beam1) for both aluminium and steel.

Using these values for the insulated cases as discussed before reveals that the thermal response in time slowed significantly after activation has been reached, see Figure 38 too Figure 42. The thermal gradient over the cross section reached after 40 minutes is however, very similar to earlier results, see Figure 43 too Figure 46. In these models the floor is as discussed in 4.2.2. The seemingly inconsistent temperature deviation in the first 150°C in Figure 43 seems to be due to the rapidly decreasing thermal conductivity of the insulation in the first stage of the calculation process. The model needs a small amount of time to reach stable conditions. However the fluctuations are slight (< 5°C) and thus ignored.

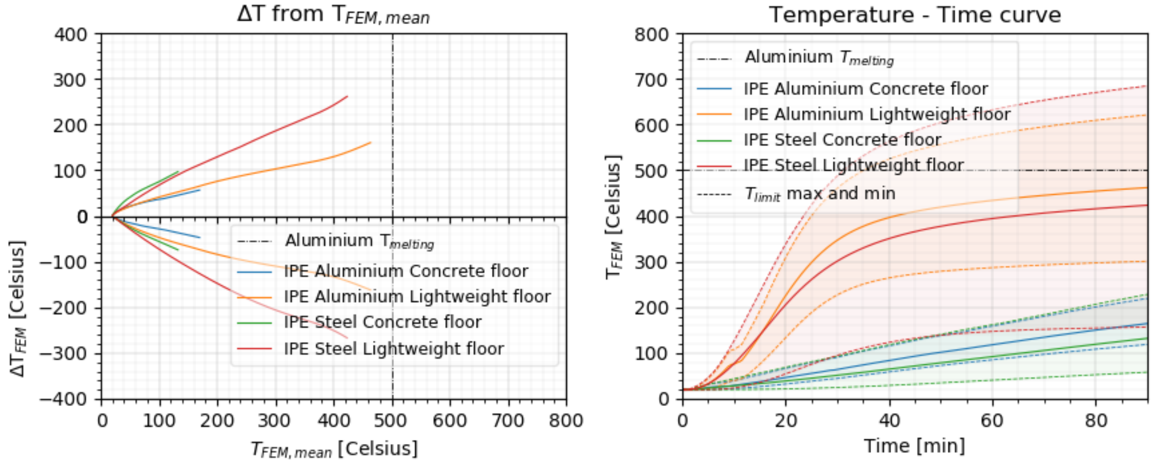

Figure 38 – Temperature time curve for an integrated IPE beam insulated with intumescent paint.

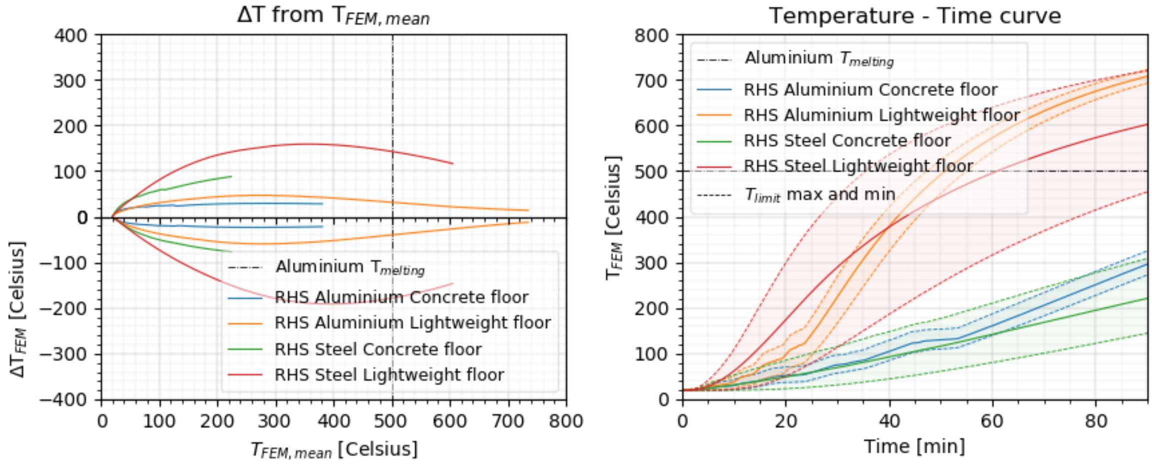

Figure 39 – Temperature time curve for an integrated RHS beam insulated with intumescent paint.

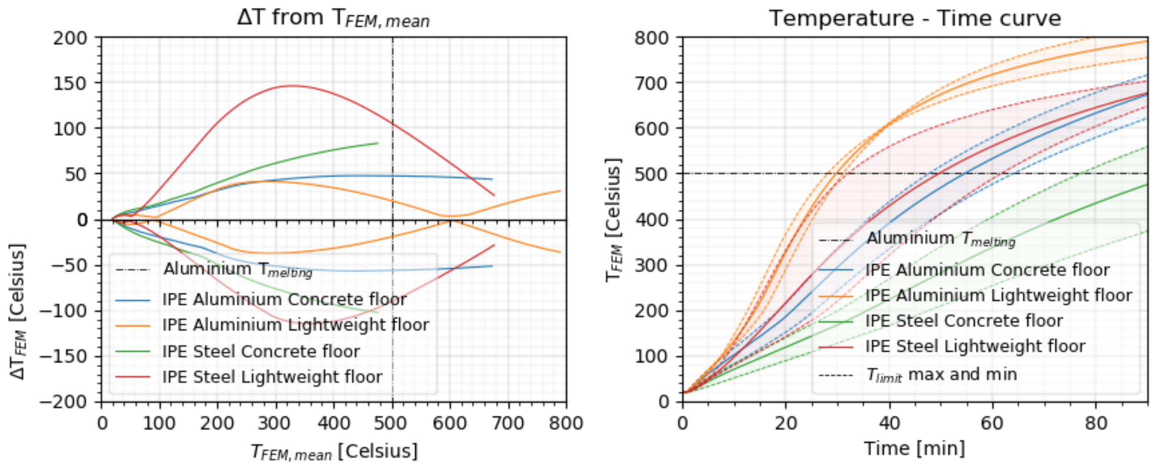

Figure 40 – Temperature time curve for a IPE beam insulated with intumescent paint facing a fire from three sides.

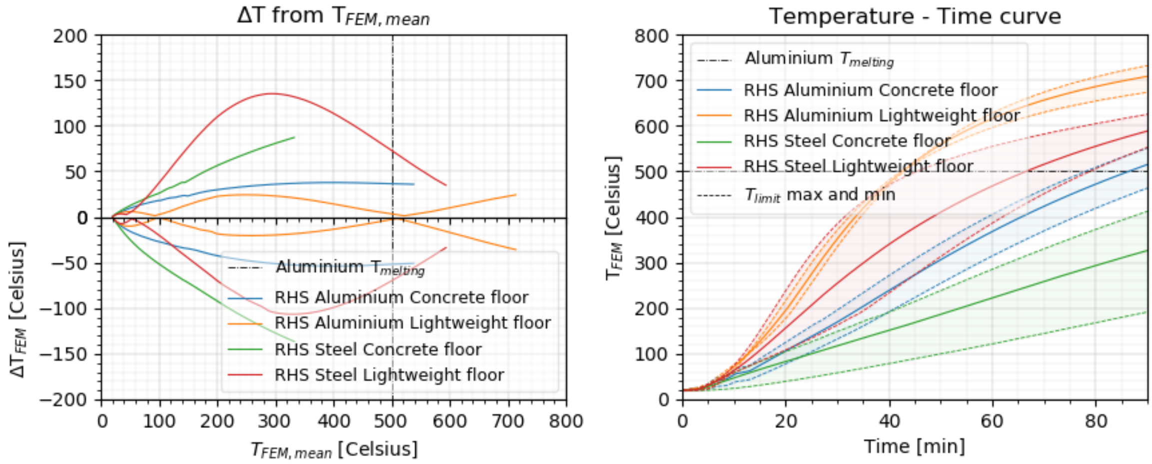

Figure 41 – Temperature time curve for a RHS beam insulated with intumescent paint facing a fire from three sides.

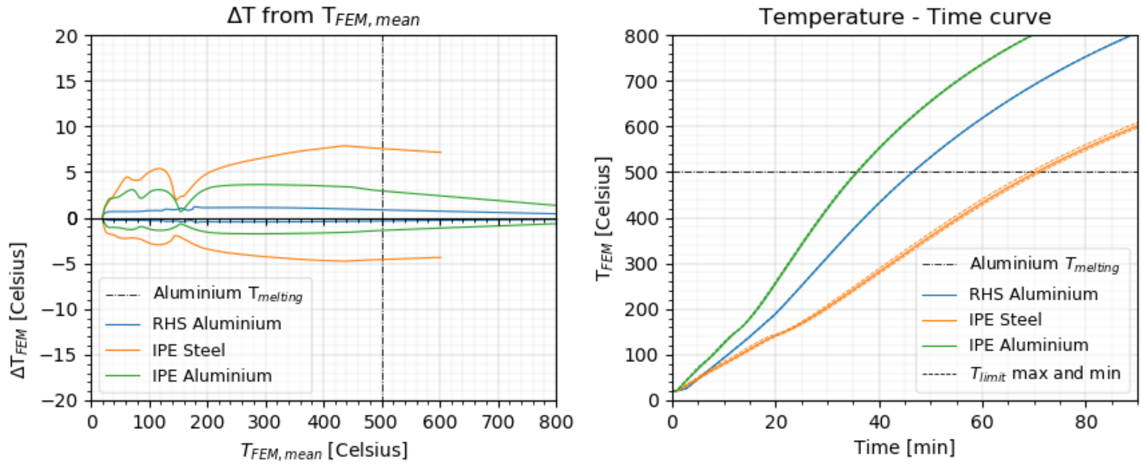

Figure 42 – Temperature time curve for a column insulated with intumescent paint, fire from all sides..

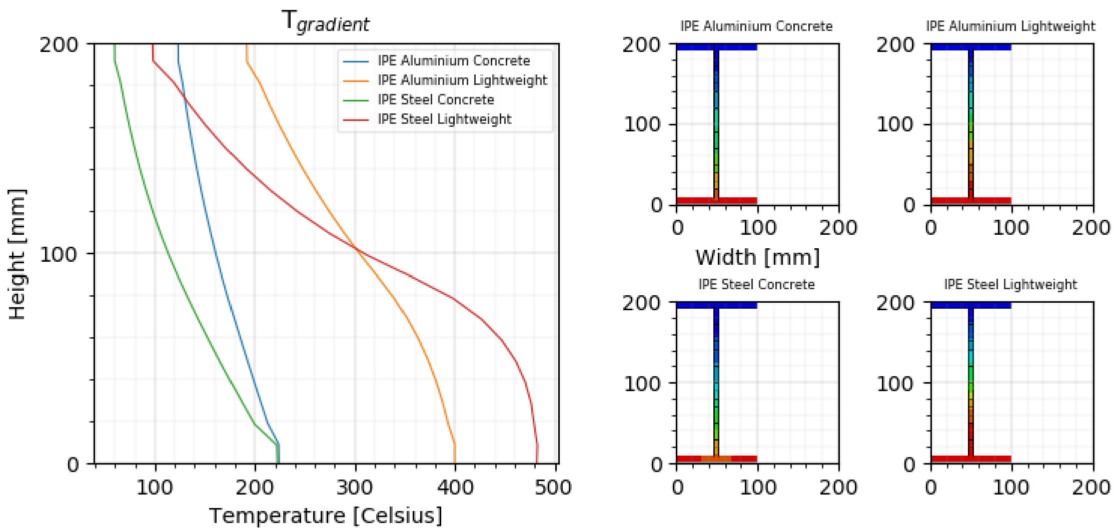

Figure 43 – Thermal gradient for an integrated IPE beam covered with intumescent paint at t=40 minutes

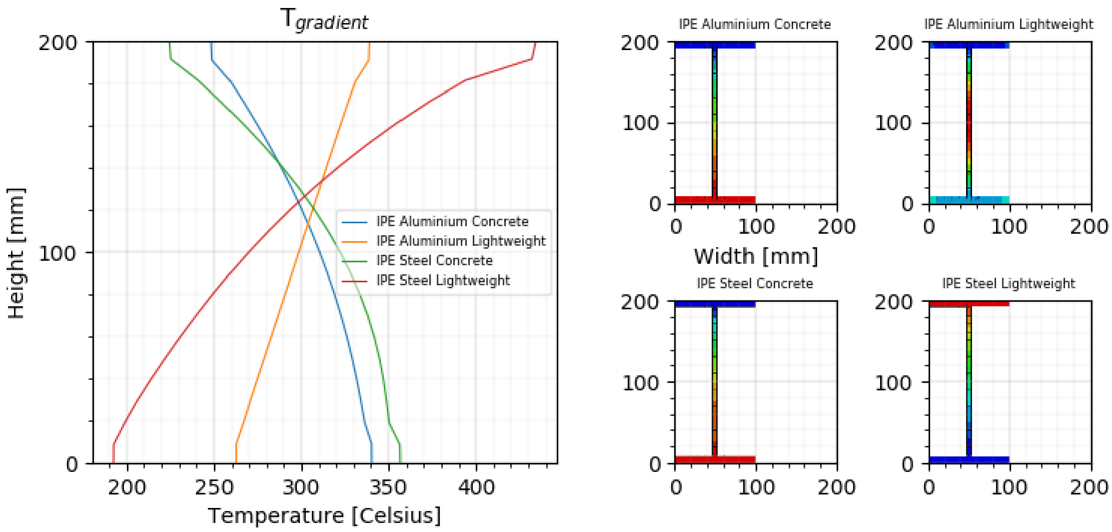

Figure 44 – Thermal gradient of an IPE beam covered with intumescent paint with fire from three sides at t=40min.

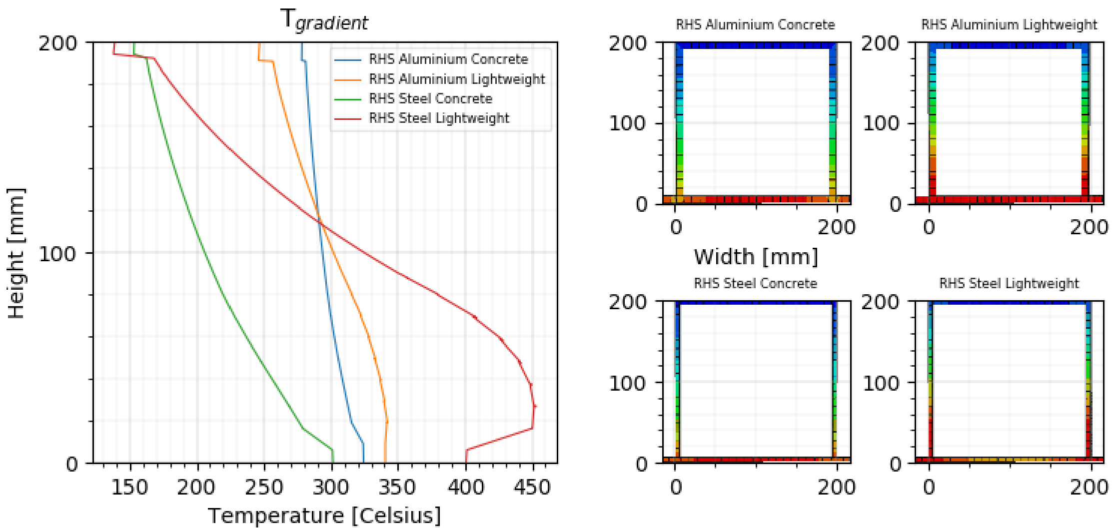

Figure 45 – Thermal gradient of a RHS beam covered with intumescent paint with fire from three sides at t=40min.

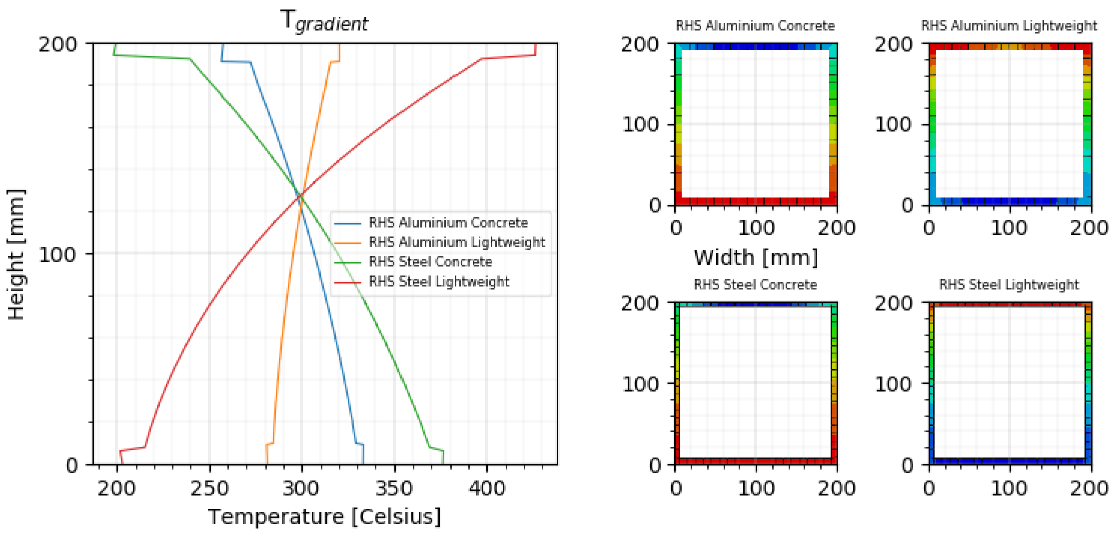

Figure 46 – Thermal gradient of an integrated RHS beam covered with intumescent paint at t=40min.

5. MECHANICAL ANALYSIS

As expressed in the previous chapter, there are three different scenarios to evaluate, that of a 1metre long column, and a 3metre beam facing a three-sided fire (Figure 18) and facing a one-sided fire (integrated beam Figure 24). Given the different loading conditions, thermal gradients and other not yet considered mechanisms, the model consists of a full 3D model. As the thermal gradient in thickness direction of flanges and web is practically nil, it is possible to work with shell elements. In the FEM environment this means that the 3D cross-section can be comprised out of homogenous shell planes, specified at the middle nodes with a set thickness, see Table 3.

In the thermal analysis it was possible to model 2D as the heat input over the length of the section is constant. Due to this fact, the temperature data has to be imposed over the length – z-axis – of the cross-section. To achieve this correctly, Abaqus has two options, namely direct interpolation between nodes when working with similar element types but a different mesh – for example a coarse and fine mesh – or midside node capability. As the elements between the two analysis differ – 2D solids versus 3D shells – only the midside node capability is an option, which requires the element size of the mesh between the two models to be identical. The temperatures of these central nodes in the mechanical shell elements, which is modelled following the central lines of the cross-section, are based on the temperatures from the corner nodes of the heat transfer elements. Using the temperatures of the corner nodes of the elements of the thermal analysis, Abaqus interpolates the midside node temperatures so that all nodes have temperature values assigned, using first order interpolation.

It is assumed that the thermal and mechanical analysis can be performed sequentially, see Figure 9. The mechanical analysis iterates over the temperature frames. Within one temperature frame, a non-linear mechanical analysis can be performed. After convergence, the analysis is restarted for the next temperature frame, building on the strain, stresses and displacements of the previous step. As the thermal analysis is of transient nature, the mechanical analysis comprises a transient non-linear analysis due to the temperature depenedent material properties and possible large deflections. The output of the mechanical analysis consists of (true von Mises) stress and (true logarithmic and plastic) strain values at integration points, and coordinates, rotations, displacements at nodes, all in XYZ-plane.

5.1 Strain relation

Stress-strain relations at elevated temperatures are best presented through transient state experiments instead of steady state test. For steel, the data in EC3 is based on transient tests for which a determining bi-linear relation is observed, therefore the stress-strain relations can be straightforwardly modelled. Aluminium however, has a distinct non-linear stress-strain relation. This is also attributed to the early onset of creep. Creep strain can be accounted for implicitly by altering the stress-strain relation, as is done in [20][5] for alloy 6060-T66. Or explicitly by accounting for primary, secondary (and tertiary) creep as proposed in the Dorn-Harmathy method [16][9].

5.1.1 Implicit stress-strain relation

In [5] the stress-strain relation is modelled taking creep implicitly into account, for which the temperature rate and stress is assumed constant. However, these assumptions are not principally valid. As can be observed in the figures Figure 16, Figure 20, Figure 21, Figure 26 and Figure 27, the temperature does not increase in a linear fashion necessarily. In addition, restrained movement of the specimen – for example boundary conditions restraining thermal expansion – can induce additional stresses to the mechanical loading. In Figure 47, the stress and strain relations at elevated temperatures taking creep implicitly into account following the Ramberg-Osgood equation is plotted.

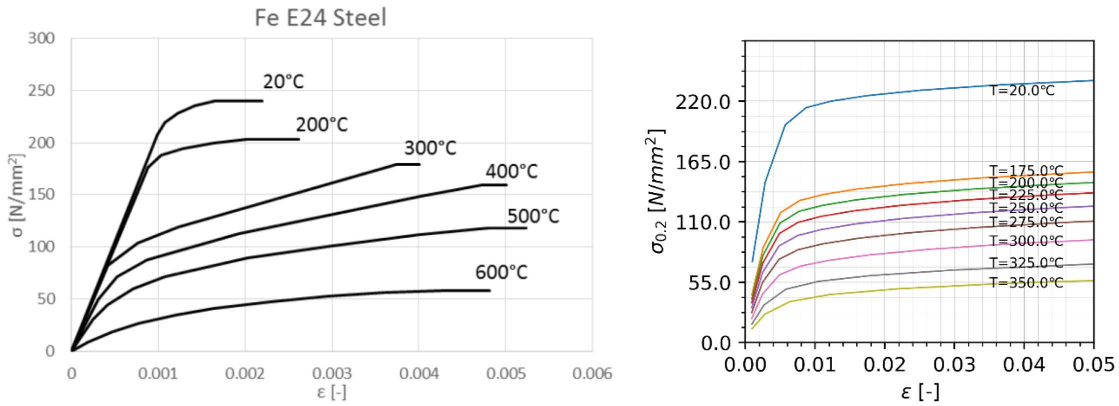

Figure 47 – The altered stress-strain curves for left steel grade Fe E24 [59] *(similar to S235) and right aluminium alloy 6060-T66 with creep implicitly incorporated.*

5.2 Structural model

In EN 13381 the fire test for column specimens is fairly straightforward. Therefore, the structural model for both IPE and RHS section can be specified as unloaded, 1 metre in length and fully restrained at one edge. For beams however, the setup is quite different considering loading, boundary conditions and lateral support.

The lateral support for both the RHS and IPE at bottom and top edge is specified to prevent out-of-plane displacement and focus on pure bending behaviour. Generally the beam is setup as a simply supported beam 3 metres in length, one edge supported with a roller and the other end a hinge. There are three different edge faces at which the support can be specified, that would be the (1) end face (all flanges and web), (2) top flange or the (3) bottom flange.

For loading, there is a difference in weight between the lightweight and concrete floor. In addition, the load face between the 3-sided beam and the integrated beam is different. As the 3-sided beam has the floor on the top edge, this is also where the load is transferred. However, for the integrated beam the load is introduced at both the bottom flange as the top flange. The loads are as described in Table 9. For the 3-sided beam this is situated at the top flange of the beam, while for the integrated beam the load is imposed by ¾ on the bottom flange and ¼ on the top edge, see Figure 48.

Table 9 – Total load on the cross-section in the FEM model, equally distributed on the contact surface at T=0min.

Floor types	Load [kN]	Utilization steel [-]	Utilization aluminium [-]
Concrete	49.5	0.43	0.48
Lightweight	36	0.31	0.35

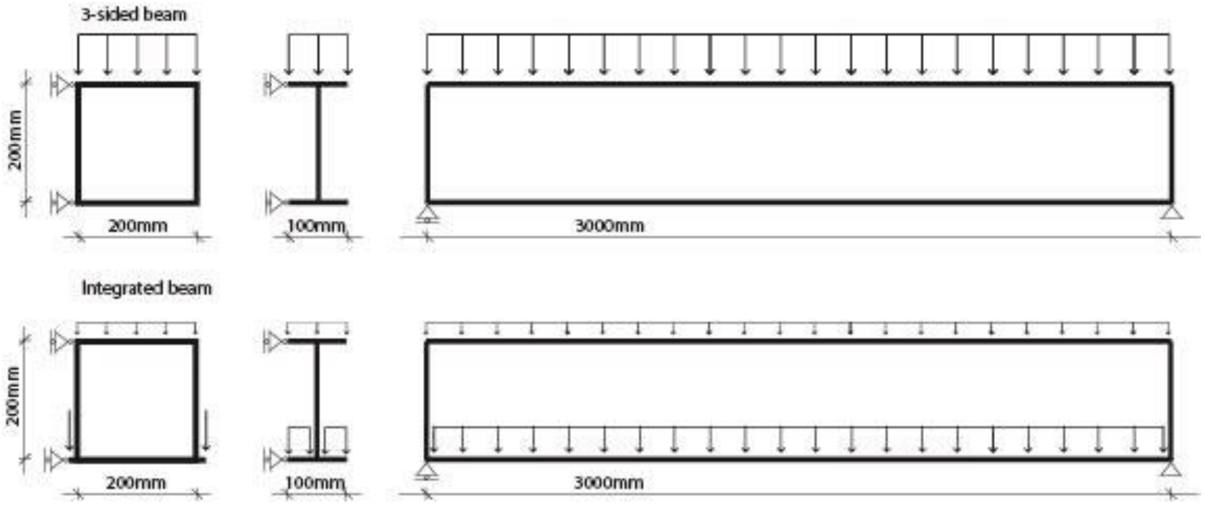

Figure 48 – Structural model for beam models, for both integrated (bottom) and beam facing three-sided heating in case of an evenly distributed load.

In conjunction, for the four point bending test a separate loading scenario is described as in Figure 49. The distribution over top and bottom flanges is as described before, in case of the integrated beam the bottom flange supports ¾ of the load.

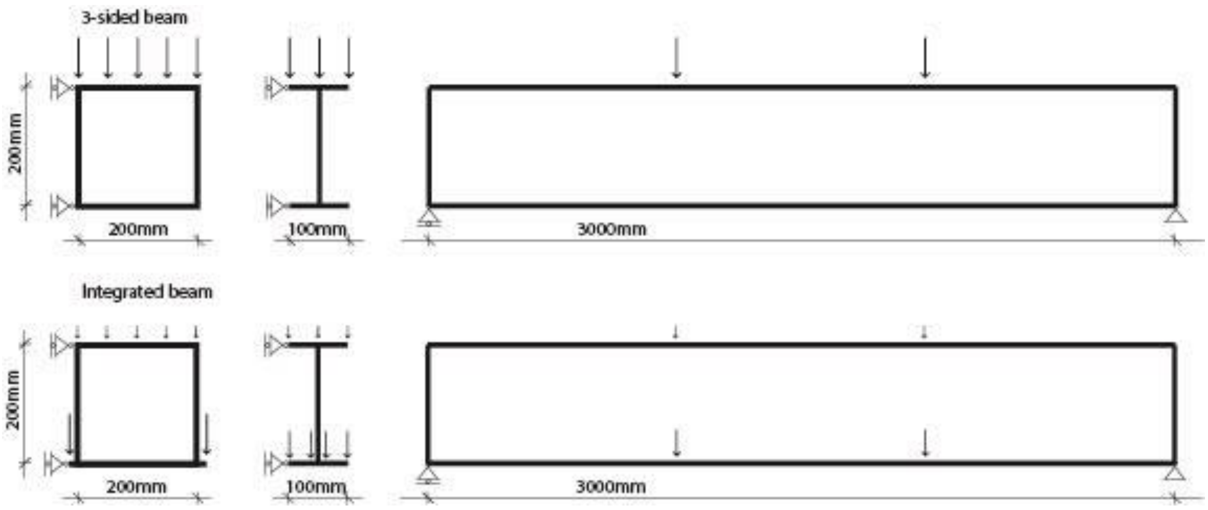

Figure 49 – Structural model for beam facing 3 sided heating (top) and integrated/1-sided heating (bottom) in case of a four point bending test setup.

5.3 Model limits

To get a full overview of the mechanical model to judge and validate, the output includes stress and strain values at integration points of the shell, and rotations and displacements at mesh nodes. This is done at every time step n for in total 90 simulated minutes to ensure sufficiently high enough temperature values (>300°C) are reached despite the insulation. After the initial thermal analysis, the 90 minute mark could be assessed to fit with the failure temperature as expressed in Chapter 3.

Alternatively or in tandem to the critical temperature, failure of the metal can be defined following the deflection limits in Chapter 3 results in a limiting strain of $3.75‰$ and a strain rate below 1.7 $d\varepsilon/dt$. Additionally, to determine whether the insulation is not damaged before the aforementioned limits, a second limiting strain value is proposed. For intumescent paint used on a steel structure, the strain at which the paint layer is damaged has been observed to be 1.3‰ [58] additional strain after the coating has fully expanded, that would be strain at 250 à 300°C.

5.4 Validation mechanical model

5.4.1 Column

Given that the column models are unloaded, the only strain phenomenon to which the model is subjected would be thermal expansion. Contrary to the beam models, the displacement must be viewed in lengthwise direction of the geometry, Z-direction. Considering the relation for the thermal expansion as described in paragraph 3.3.2.2 it is possible to validate the numerically found displacement with that theoretically found using equations (16) and (17). For Abaqus, the thermal expansion coefficient is expressed by dividing the thermal strain with the temperature minus the reference temperature $\alpha_{L,i} = \varepsilon_{th,i}/(\theta_i - \theta_0)$. The reference temperature is used to correct the value because it is assumed that at initial conditions the expansion is zero, that would be at $\theta_0 = 20°C$. The difference between the thermal expansion coefficient in FEM and theory is because the theoretical value was calculated assuming a linear increase in temperature instead of the true value. The resultant coefficient can be observed in Figure 50, in addition to the displacement calculated by hand for comparison and the Abaqus result. The maximum temperature in the cross-section is used for calculation. As is evident, the displacement between the models is practically identical and therefore sufficiently validated. Though note that Abaqus expresses true stresses and strains.

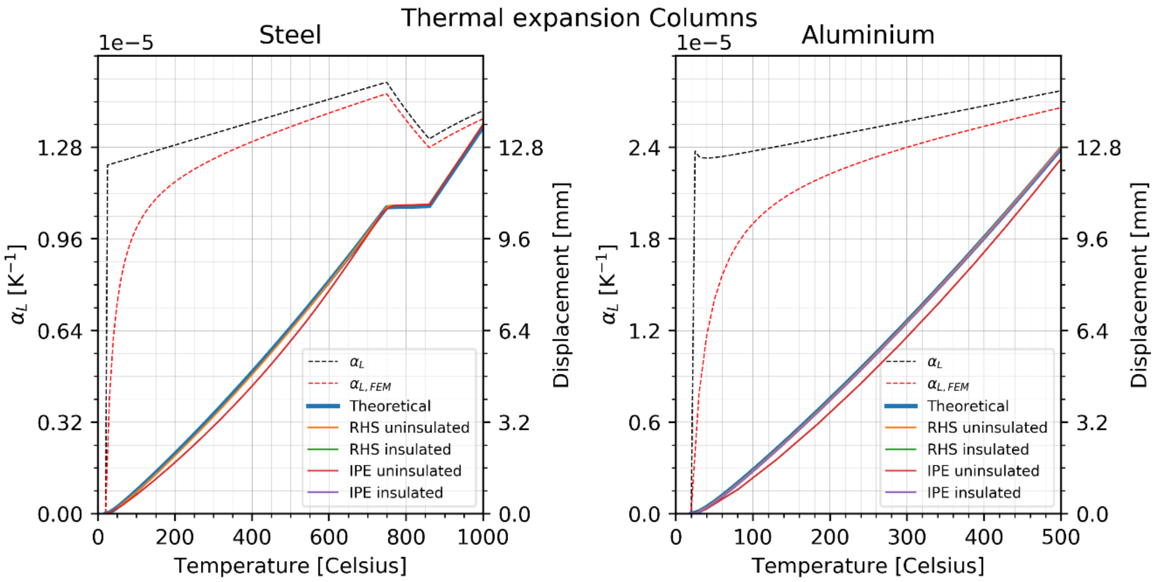

Figure 50 – Thermal expansion coefficient and corresponding theoretical displacement in comparison to the lengthening of the columns found with Abaqus with maximum temperature.

5.4.2 Beam

The beams are loaded in bending – both the integrated beam and the 3-sided beam – and have a thermal gradient over the cross-sectional height due to the non-uniform heating conditions. Therefore the beam exhibits elasto-plastic behaviour in addition to creep and thermal strain, see equation *(15)*. Thermal strain is a combination of thermal elongation and thermal bowing which act in orthogonal directions. Creep strains have been taken implicitly into account by adjusting the stress-strain curves. Presumably, the determining strain and displacement occurs at midspan in the hottest flange. To ascertain this the values at the centroid and at the centres of both flanges is inspected.

The total strain in length direction – identified as Z-axis or S11 in Abaqus for this model – following from equation *(15)*, the total strain can be approximated with equation (23) and the displacement at midspan as equation (24). Note that the temperature difference in this case is taken over the height of the cross-section. The strain hardening factor n is determined by dividing the proof stress by 10 [15]. Given the orientation of the model, the maximum strain occurs in length direction (Z-axis) and the maximum displacement happens orthogonally in the Y-direction at midspan.

$$\varepsilon_{total} = \left(\frac{\sigma}{E_\theta} + 0.002\left(\frac{\sigma}{f_{0.2;\theta}}\right)^n\right)_{mechanical} + \left(\alpha_{L,\theta}\,{}^1\!/_4\,\Delta T + \alpha_{L,\theta}\right)_{thermal} \tag{23}$$

$$D_{midspan} = \left(\frac{5}{384}\frac{qL^4}{E_\theta I}\right)_{elastic} + \left(\frac{\alpha_{L,\theta}\Delta T L^2}{8h}\right)_{thermal} + D_{plastic} \tag{24}$$

The variables in (23)&(24) are as in Figure 47 and Figure 50, which are dependent on the constant cross-sectional temperature as calculated in (6). The thermal gradient ΔT follows from the previous thermal analysis, Figure 20. The load for validation is 20N/mm for steel and for aluminium. Given the different geometry, the loading factor with this load is approximately $\frac{\sigma}{f_{0.2,\theta}} = \frac{1}{2}$ at $T = 20$°C for both.

To determine whether both mechanical and thermal effects for strain and displacement are implemented correctly three simulations were run, (1) that with only thermal expansion and no loading, (2) only loaded with no thermal expansion and with (3) both active. The results of all three is evaluated at midspan at 3 locations, middle of the web, at the centre of the top flange which is facing the ambient side, and the centre of the bottom flange which is heated. Note that all deflection downward, as in towards the fire side, is taken positive while toward the ambient side is negative.

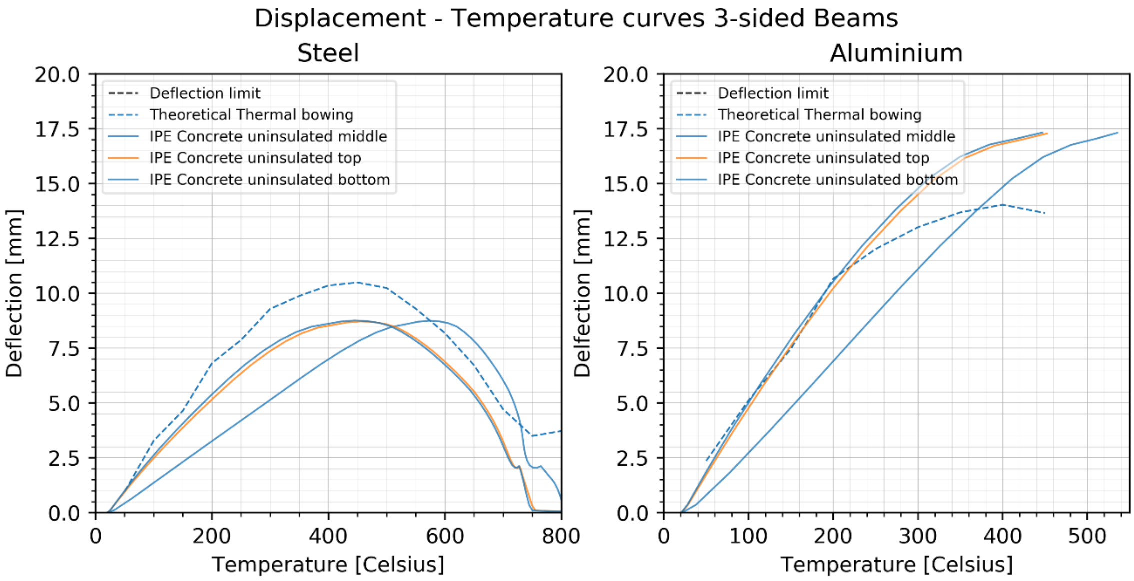

Figure 51 – Thermal expansion at midspan for an uninsulated IPE cross-section for both steel and aluminium, considering different cross-sectional locations: centre bottom flange, middle of web, centre of top (ambient) flange. Note that the result for middle and top coincide.

With the thermal gradient as in Figure 22 the result of the model with only thermal expansion results in Figure 51. The irregularity for steel at circa 700°C is due to the fact that the thermal expansion is constant for a range. The fact that the relation is representative of a concave parabola follows form the fact that the thermal gradient over the cross-section reduces with higher temperatures. Thermal expansion can continue until the melting temperature has been reached. Even though the thermal gradient in aluminium is lower – as expected given its higher thermal conductivity – the thermal expansion is higher, given that the thermal expansion is roughly twice as large. It is evident that the trend in the data is similar between FEM and theory, even if the percentile difference between the values can amount to 25%. The difference is attributed to the effect of the thermal gradient, which in the theoretical model is straightforwardly taken as the minimum and maximum temperature occurring in the cross-section. However in the FEM analysis, it is clear that the thermal gradient is not linear over the cross-section and the results plotted are the actual deflection at midspan with the corresponding local temperature. With this explanation, the implementation of the thermal expansion is assumed to be correct.

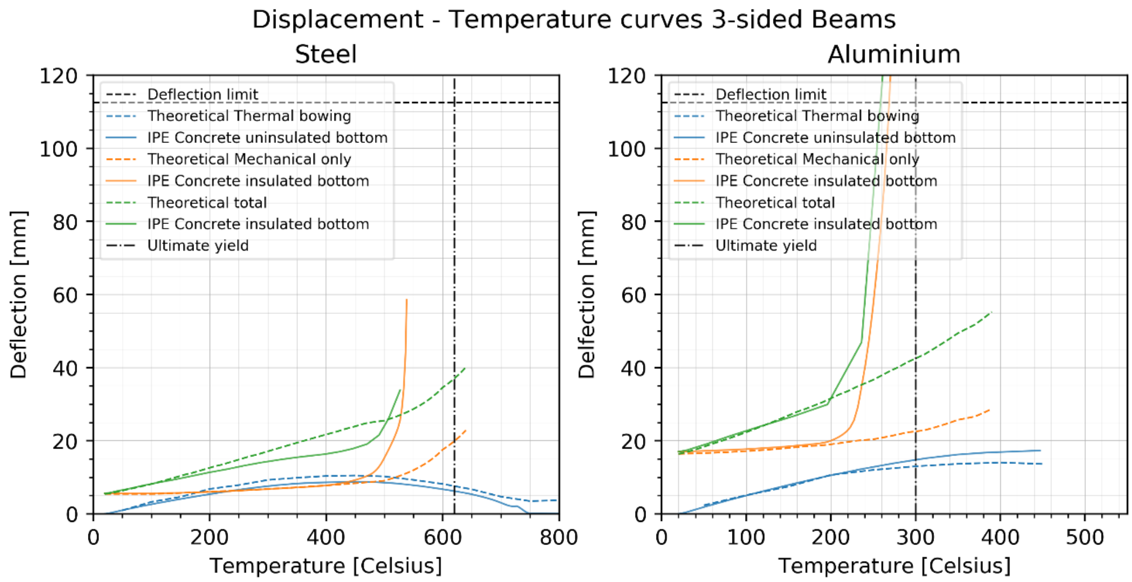

Figure 52 – Deflection for uninsulated IPE section with thermal expansion (1), for insulated IPE with no thermal expansion only loading (2), and deflection for an insulated IPE section with both thermal expansion and loading (3).

In Figure 52 the deflection for all three scenario's is plotted. The green line represents the full analysis, yellow that with only loading and blue with only expansion as in Figure 51. As the theoretical expression of the deflection due to elasto-plastic behaviour is only expressed for elastic behaviour, the vertical dash-dotted lines represent the asymptotes at which the stress values in the FEM model exceed the proofstress and ultimate stress. The fact that elastic deflection does not start at zero is due to the initial deflection at load introduction. In the initial elastic range, the FEM and theoretical results overlap. For steel it is evident that beyond the 205°C the model starts experiencing plastic deformation which results in approaching the asymptote as expected. Note that these values are evaluated with the temperature at the centre of the bottom node, which exhibits the most extreme results. Except for the thermal bowing, the theoretical results are calculated following the temperature development as in EN 1993-1-2 and EN 1999-1-2. The main difference therefore is that the temperature over the cross-section is assumed constant while this is not the case in the FEM analysis. The result of an underestimation of the deflection before failure fits with earlier found results in literature [5][8][19].

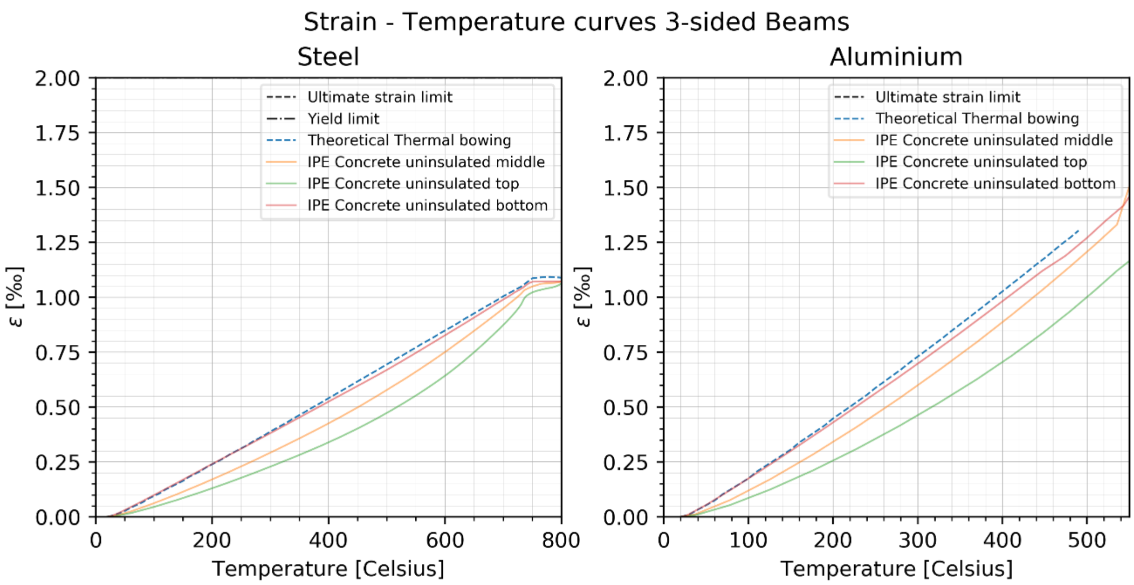

Figure 53 – Thermal strain for an uninsulated IPE section exposed to fire at three sides for both steel and aluminium, considering different cross-sectional locations: centre bottom flange, middle of web, centre of top (ambient) flange at midspan.

The thermal strain and its variation over the height of the beam causes thermal elongation and thermal bowing. For thermal strain, the same thermal gradient issue exists as with deflection. The temperature is not linearly distributed over the cross-sectional. Hence the difference between the different evaluation points at top, bottom and middle of the cross-section in Figure 53. The difference in theoretical value and that at top and middle nodes is due to that of the thermal gradient, which in the theory is taken as the minimum and maximum temperature occurring in the cross-section. While the temperature against which the values for top and middle are plotted are their actual local temperatures. The relation between temperature and thermal strain seems relatively linear, fitting with the thermal expansion coefficient.

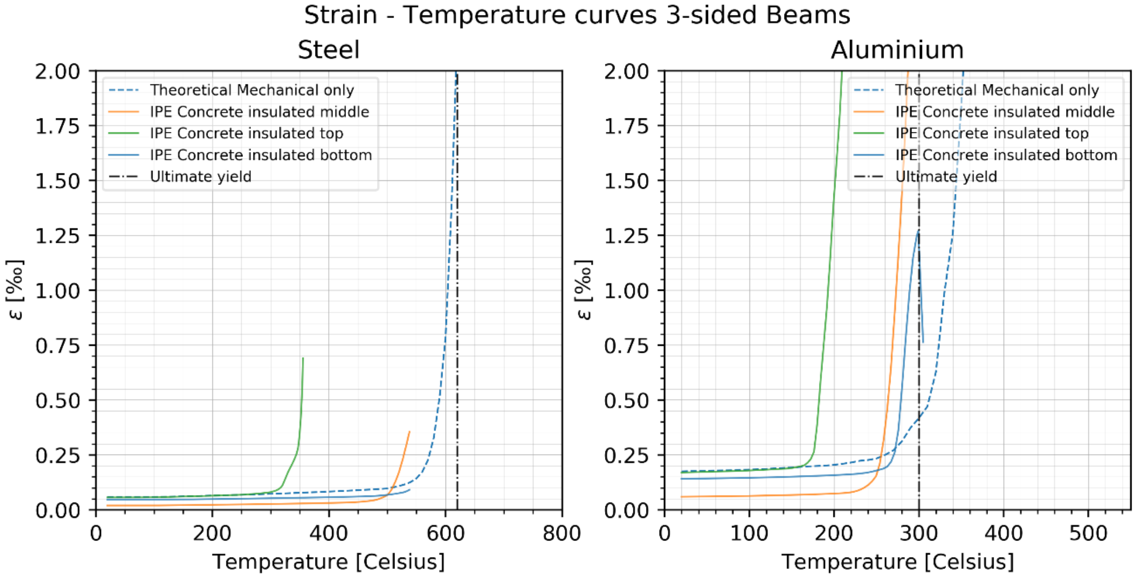

Figure 54 - Mechanical strain of an insulated IPE cross-section exposed to fire at three sides for both steel and aluminium, considering different cross-sectional locations: centre bottom flange, middle of web, centre of top (ambient) flange.

For the mechanical strain the Ramberg-Osgood equation is used. Given that the load is constant, only the thermal dependent variables of Young's modulus and proofstress are determining factors. The result of the theoretical value follows the asymptote of the ultimate stress. Given the cut-off of the red-line in Figure 54, it is evident that yield happens sooner in the centre of bottom flange. The apparent asymptotes that reached in Figure 54 agree with the ultimate yield criteria as in Figure 53. The difference between the theory and the FEM is due to the temperature, as the theoretical results are calculated following the temperature development in EN 1993-1-2 and EN 1999-1-2, see 3.2.3. In addition to this correction, the theoretical strain for aluminium showcases an earlier more gradual curve because of the smooth approximation of the proofstress at elevated temperatures as in Figure 47.

In the FEM analysis Abaqus does not explicitly consider the effect of melting as these limits are not provided and approximates any necessary material properties through linear extrapolation when beyond the given scope. Note that simply taking the maximum values for strain and deflection which occur in the beam does not work, as locally the yield criteria can be met due to local plasticity elsewhere to midspan. This is especially a concern for steel as the Young's modulus degrades faster than the yield stress does [3]. Area's which are susceptible to this include At elevated temperatures it is therefore more susceptible to local yielding. The opposite is true for aluminium.

Given the aforementioned observations, the result of the thermal expansion and mechanical part separately and combined show relatively accurate results for deflection, coinciding with theory in such a way that it can be initially assumed that the model is accurate. However, in further research, the preference for validation lies with an additional simulation model following an actual fire test and comparing the results, and perhaps, simulating a copy of a benchmarked literature reference.

6. RESULTS

While considering the results in the coming figures, the deformed shapes have been plotted against their original shape (red outline) with a scale factor of 3 at the last converged step. For the columns, thermal elongation is taken as positive while shrinkage is negative value. Beyond this, for beams deflection towards the fire side is deemed positive, and toward the ambient side is negative.

6.1 Column

The uninsulated sections were relatively straightforward to develop. However as can be observed from the amount of steps completed with the insulated section, see Table 10, modelling insulation proved much more troublesome. As of yet, it is unclear why these problems were unable to converge. Evaluation of exaggerated results within the 16 steps reveal no clear cause or effect, as the results are consequently equal to zero and the temperature change does not move past $\sim 5.0 * 10^{-4}$ ℃. Convergence is not achieved either when incurring a minimal pressure load to the column head, or when describing a maximum deflection, or under different support conditions. The model that did succeed however, follows the same trends as where described in the validation 5.4.1 and forms a bases to proceed with to at least form a preliminary judgement for this thesis.

Both steel insulated models were unable to run. However, given that similar experiments and models have been thoroughly tested and previously established, the temperature relation to strain for the insulated case can be extrapolated from the uninsulated case for comparison to the beam models. This approach can only be taken because the result of similar analysis has been well established in the past and the conclusion that beams exhibit significant sagging before failure in comparison to columns is a confirmed phenomenon and the reason for the fire test setup as previously discussed with both column and beam tests.

The FEM model for columns is an unloaded situation with one end fully clamped. Therefore only thermal expansion in the lengthwise direction is subject of discussion in this case. Due to the fact that one end is fully clamped, peak stresses can occur at this support, as seen in Figure 55. However, due to the setup these can be neglected. Table 10 accompanies Figure 55.

The overall temperature development in the cross-section, the second plot in Figure 55, shows a consistent temperature development with a minimal deviation. This is as expected given that the section is heated evenly from all sides. Within the 90 minute timeframe, the whole aluminium section achieves melting temperature, therefore the data is capped at a temperature of 500℃ which is at approximately 45 minutes for the insulated section and less than 10 minutes for the uninsulated sections. The dip in the temperature development of steel at ~ 20 minutes is due to a shift in the thermal parameters as the specific heat reaches an asymptote as it is a rational function (1/x type) at this temperature and the thermal conductivity switches from a linear description to constant.

Given that the column models are unloaded, the strain result follows directly from the approximation of thermal elongation as established in the validation of section 5.4. As a result of the temperature dependency of the Young's modulus and proofstress, the strain and deflection curves strongly resemble the shape of the temperature curve. A direct effect of there being no thermal gradient.

Modelling up until the melting temperature of aluminium reveals that the FEM deflection result directly matches with the deflection and deflection rate limit as prescribed in EN 1363 [6][13].

Table 10 – Legend overview for Figure 55 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Minutes	Legend	Model	Minutes	Legend
Aluminium IPE uninsulated	90	Pink	Steel IPE uninsulated	90	Green
Aluminium IPE insulated	90	Grey	Steel IPE insulated	1.6	Red
Aluminium RHS uninsulated	90	Brown	Steel RHS uninsulated	90	Blue
Aluminium RHS insulated	1.6	Purple	Steel RHS insulated	1.6	Yellow

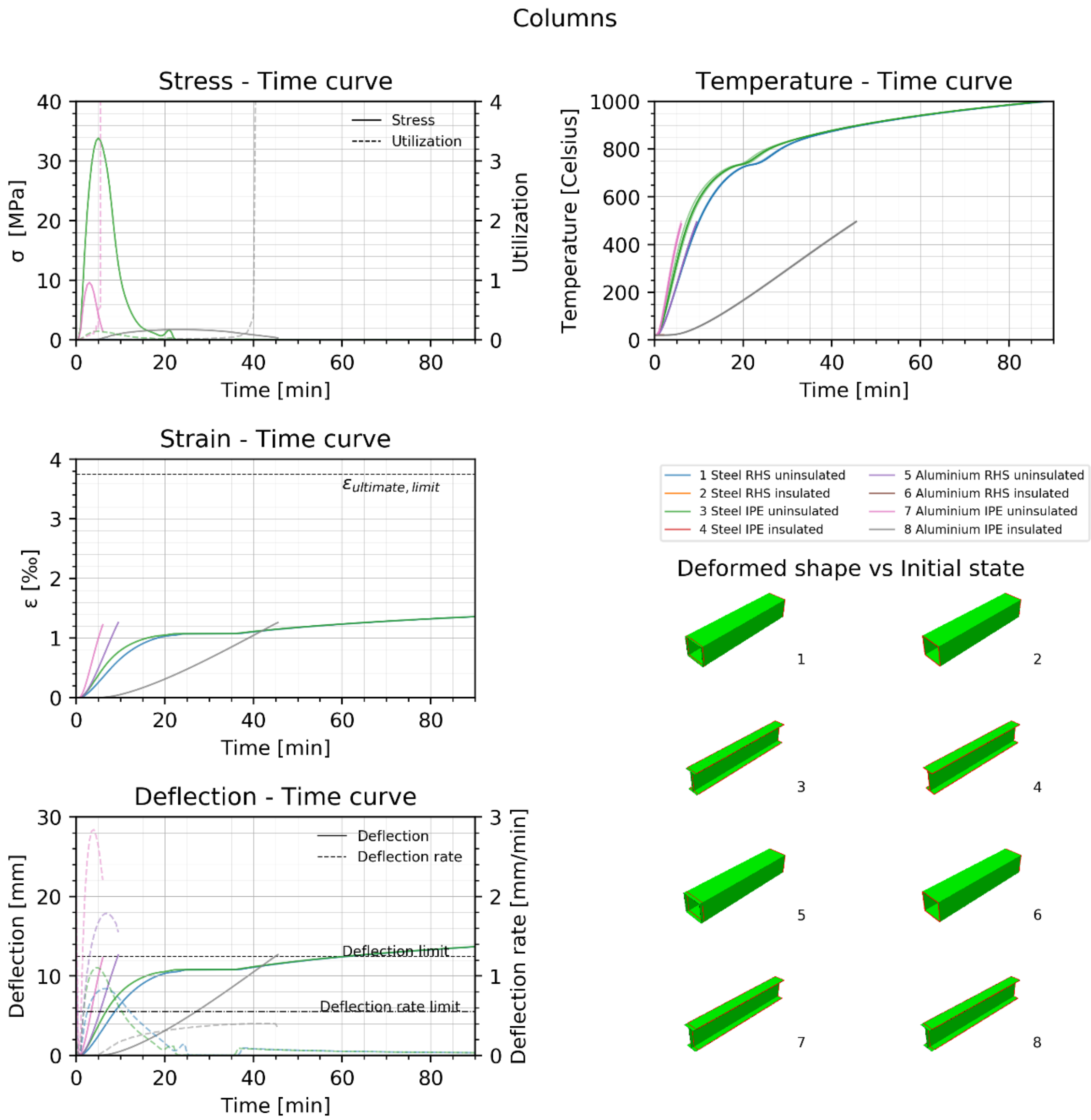

Figure 55 – Results for the full analysis of column sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.1 Columns for larger images of the deformed shapes.

6.2 Beam: three sided fire

6.2.1 Evenly distributed load

Figure 57 shows some unexpected results considering the stress and deflection of the sections with the lightweight floor element. The reduction in the deflection of the RHS with the lightweight floor seems to coincide with that of a significant reduction, or convergence of the temperature values, reducing the thermal gradient. For this case, the combination with the lightweight floor, which has a higher thermal conductivity than concrete, causes the insulated cross-section to heat more quickly through contact with the floor opposed to directly from the fire through the insulation. The thermal gradient is therefore inverted, having the highest temperature at the top instead of the bottom which faces the fire, see. Subsequently thermal bowing causes an upward deflection before elastic deflection becomes dominant. Apparently, thermal bowing at this stage is determining for the deflection of this scenario. There is therefore a shift from 'negative upward bending' to positive bending toward the fire.

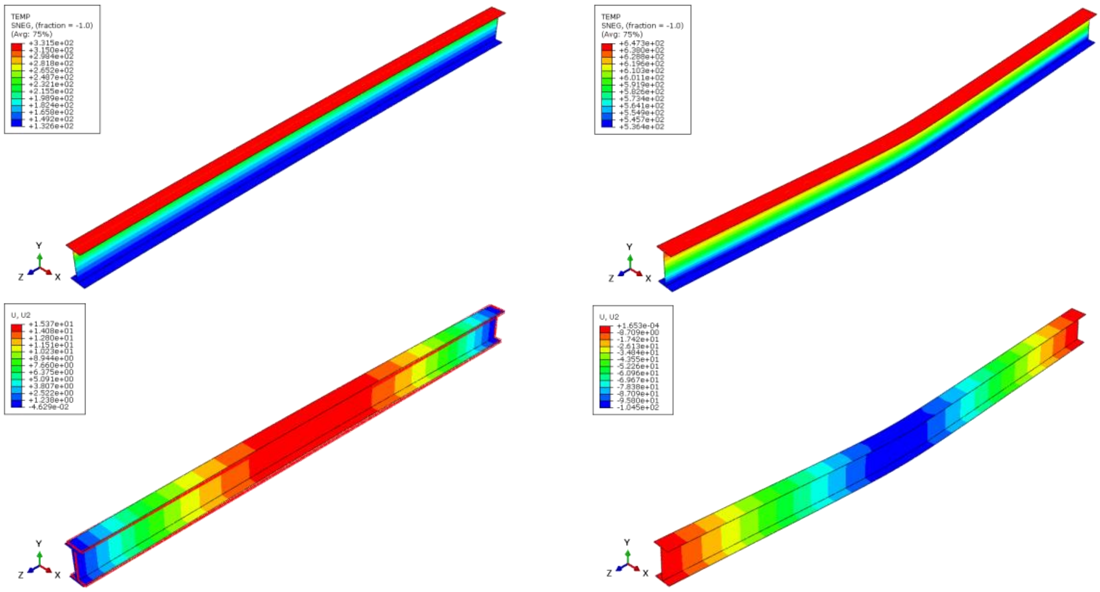

Figure 56 – Beam facing fire from three sides with a lightweight floor, steel IPE section, having an inverted temperature gradient, top images show thermal gradient in which hottest temperature is red and blue is colder. Left is the situation at 30 minutes and right at end 60 minutes. Bottom two images are the magnitude of the deflection in Y direction on the deformed shape, maximum deflection at midspan.

The steel RHS section with the concrete floor seems to not fail within the given time limit and would require a revaluation. However, given the data in the figure, there are no unexpected deviations for this case. The deflection can be observed to steadily increase as would be expected. For the green line, the steel IPE section with a concrete floor, the stress, strain and deflection values all fit with within the expected range. Deflection steadily increases until failure is achieved and a rapid increase is observed.

In all cases it appears that at temperatures exceeding 400℃, the behaviour of steel seems to change most, which fits with the fact that the yield stress starts to decrease at this point. The proportional stress at this stage would be at approximately a fourth of its original value, which would be about equal to the imposed stress on the sections. Therefore, plastic behaviour occurs from this point on.

Table 11 – Legend overview for Figure 57 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	**Minutes**	**Legend**	**Model**	**Minutes**	**Legend**
Steel IPE insulated concrete floor	79	Green	Steel RHS insulated concrete floor	90	Blue
Steel IPE insulated lightweight floor	59	Red	Steel RHS insulated lightweight floor	90	Yellow

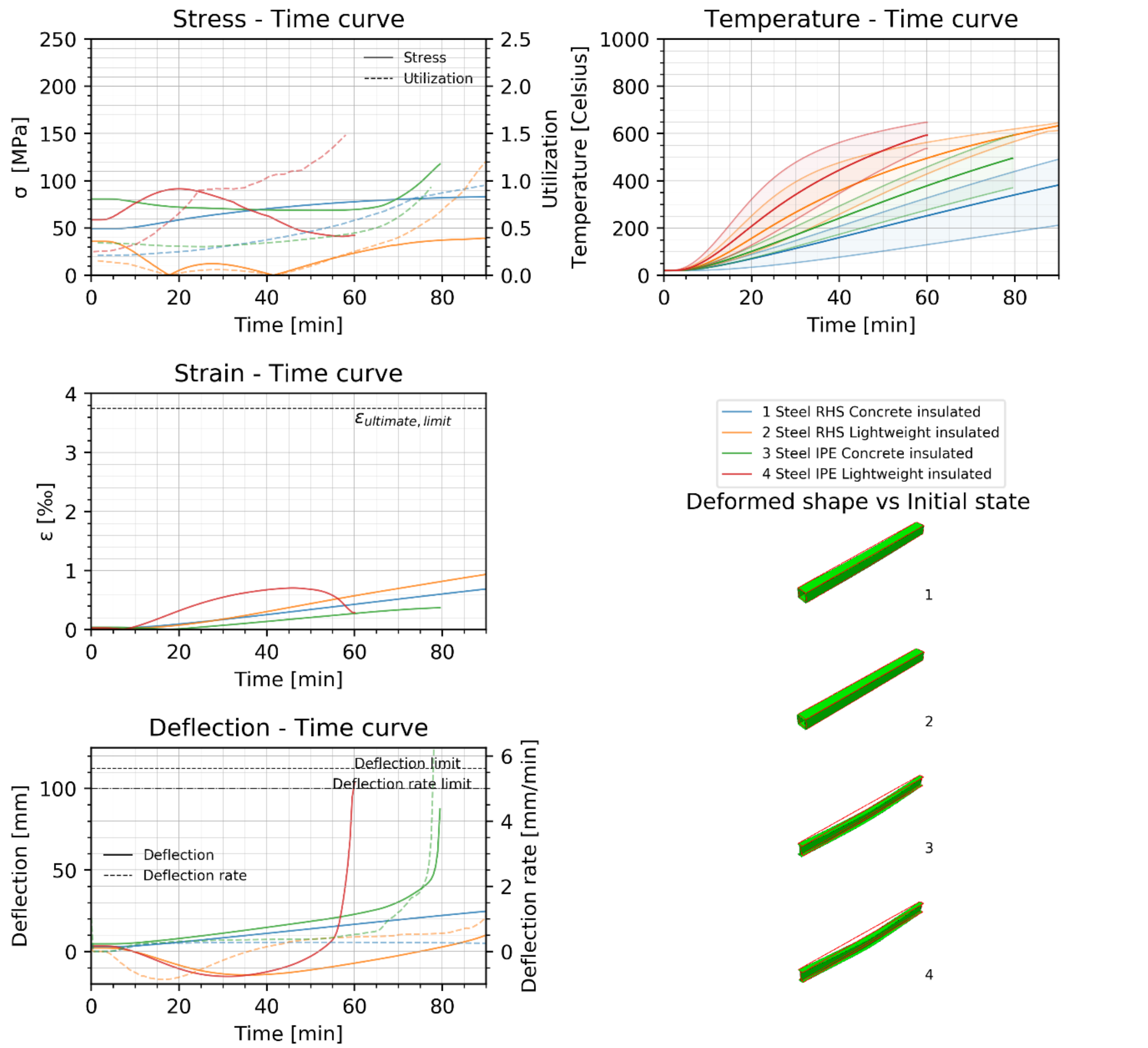

Figure 57 – Results for the full analysis of 3-sided beam with an evenly distributed Q-load, steel insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

As is to be expected with uninsulated section, the thermal gradient is lower and the critical values are achieved within a short time period. Within 10 to 15 minutes the deflection of the beams already reaches limit values. This corresponds with the peaks found in the stress curve which shows a quick cutoff or drop after reaching yield. The dib before this point seems to be due to a redistribution of the stress within the cross-section when the beam roller support appears to yield, a by-product of local peak stresses at the support.

Table 12 – Legend overview for Figure 58 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE concrete floor	41	Pink	Steel IPE concrete floor	96	Green
Aluminium IPE lightweight floor	66	Grey	Steel IPE lightweight	111	Red
Aluminium RHS lightweight floor	111	Brown	Steel RHS concrete	151	Blue
Aluminium RHS concrete floor	111	Purple	Steel RHS lightweight	176	Yellow

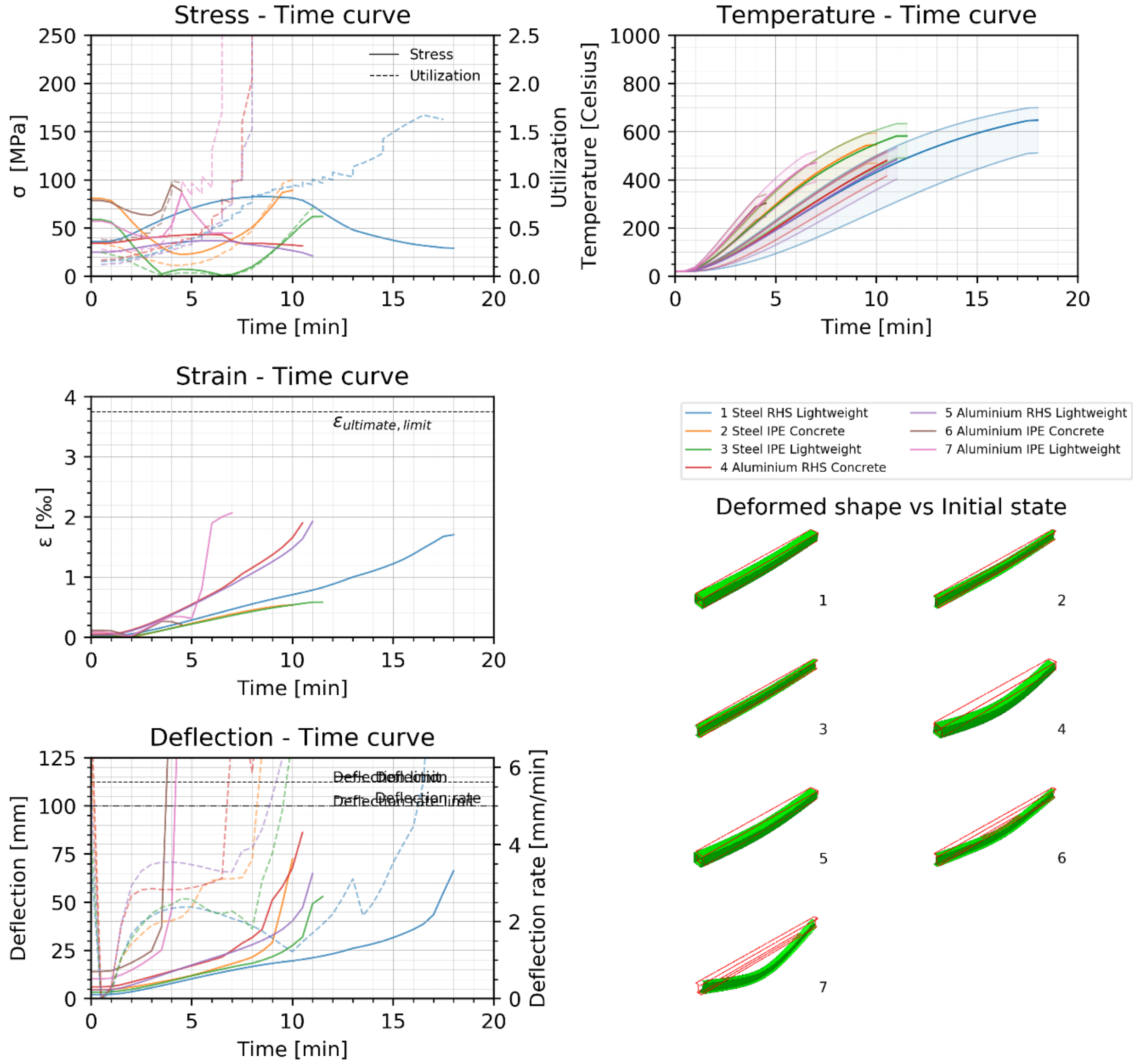

Figure 58 – Results for the full analysis of 3-sided beam with an evenly distributed Q-load, uninsulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

The cut-off temperature for aluminium is 500℃ after which the material properties are no longer of any mechanical magnitude. Apparently the case of an RHS with a concrete floor does not reach critical state. The other scenario's however do. For both the IPE sections, the point at which failure occurs is quite evident, clear as the sharp point/dip in the stress value where the proofstress is exceeded. The sharp turn of the strain for the IPE with the lightweight floor at t=20 minutes seems to be a sharp switch from reaching the proofstress to reaching the ultimate stress of the section. In comparison to the steel results, the thermal gradient is much lower. This is as expected given the larger thermal conductivity of aluminium.

As with the previous Steel IPE lightweight cross-section, the thermal gradient is inverted for the aluminium RHS cross-section with the lightweight floor. Having the highest temperature at the top instead of the bottom which faces the fire. Subsequently thermal bowing causes an upward deflection before elastic deflection becomes dominant. Apparently, thermal bowing at this stage 17-35minutes is determining for the deflection of this scenario.

Table 13 – Legend overview for Figure 59 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE concrete insulated	416	Green	Aluminium RHS concrete insulated	901	Blue
Aluminium IPE lightweight insulated	336	Red	Aluminium RHS lightweight insulated	546	Yellow

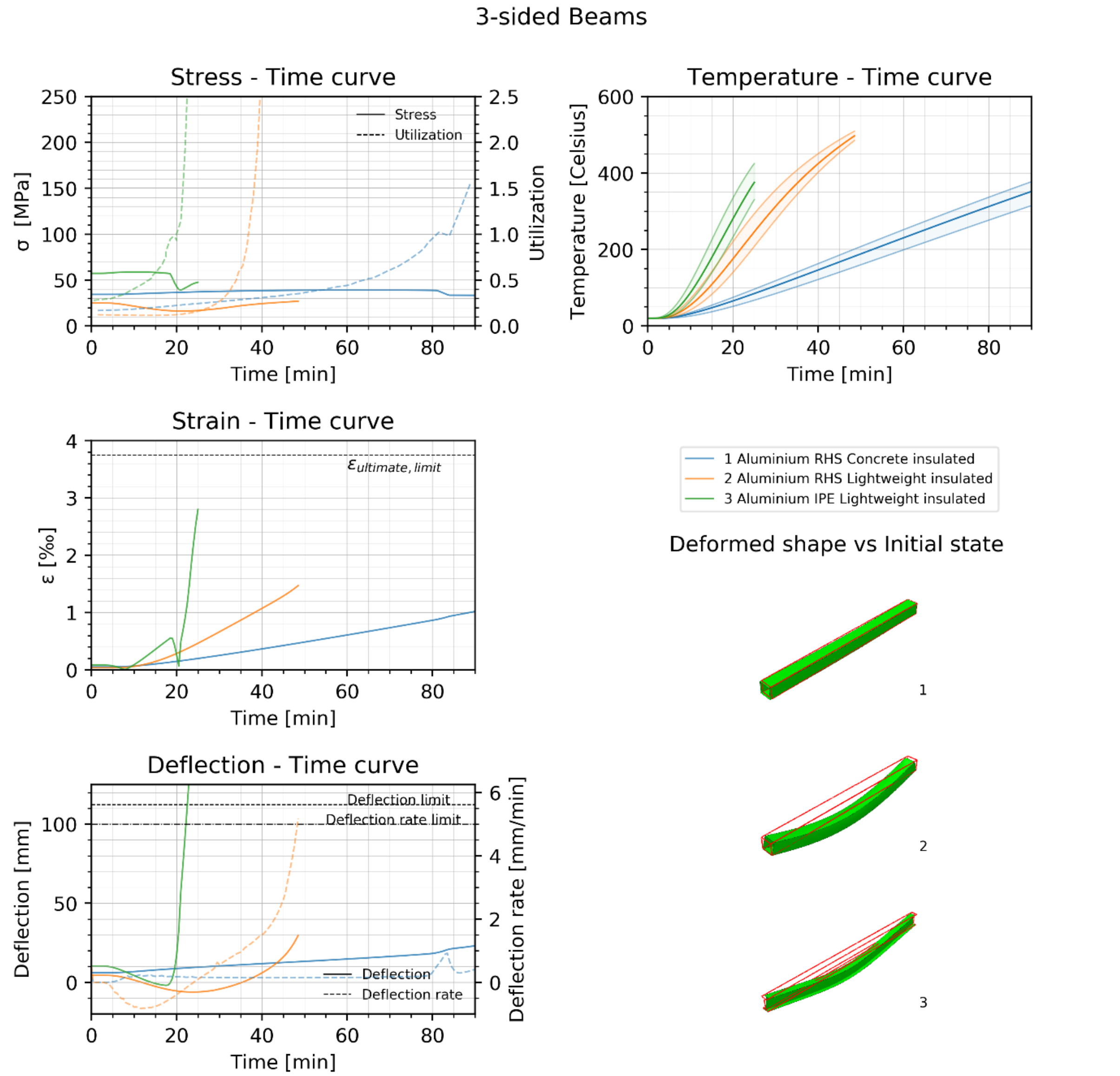

Figure 59 – Results for the full analysis of 3-sided beam with an evenly distributed Q-load, aluminium insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shape.

6.2.2 Four point bending test

A recurring issue with the four point bending test is that local plasticity around the introduction of the load and support occurs, which can cause the analysis to 'fail' prematurely. This problem was initially addressed with the introduction of a rigid area at the partition at which the load is applied. Apparently, this was not a severe enough action to achieve the intended result. With the data that was acquired however, it can be observed that heating happens much more quickly as opposed to an insulated section.

The endpoints of the stress also fit with that of the proportionality stress and then ultimate stress, and the proofstress for respectively steel and aluminium. The stress, strain and deflection results also line up, finding their extreme when expected. What is also clear is the difference between steel and aluminium. The thermal gradient for aluminium is smaller, and the aluminium IPE with lightweight floor fails much earlier than the other scenario's.

The most stress inconsistencies seem to occur with a RHS section. Such section do show a higher moment of inertia than the prescribed IPE sections. Therefore it does fit that the stress with these sections is lower in comparison. Failure is thus at a later time.

Table 14 – Legend overview for Figure 58 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE lightweight	36	Pink	Steel IPE concrete	41	Green
Aluminium RHS concrete	11	Brown	Steel IPE lightweight	91	Red
Aluminium RHS lightweight	111	Purple	Steel RHS concrete	111	Blue
			Steel RHS lightweight	136	Yellow

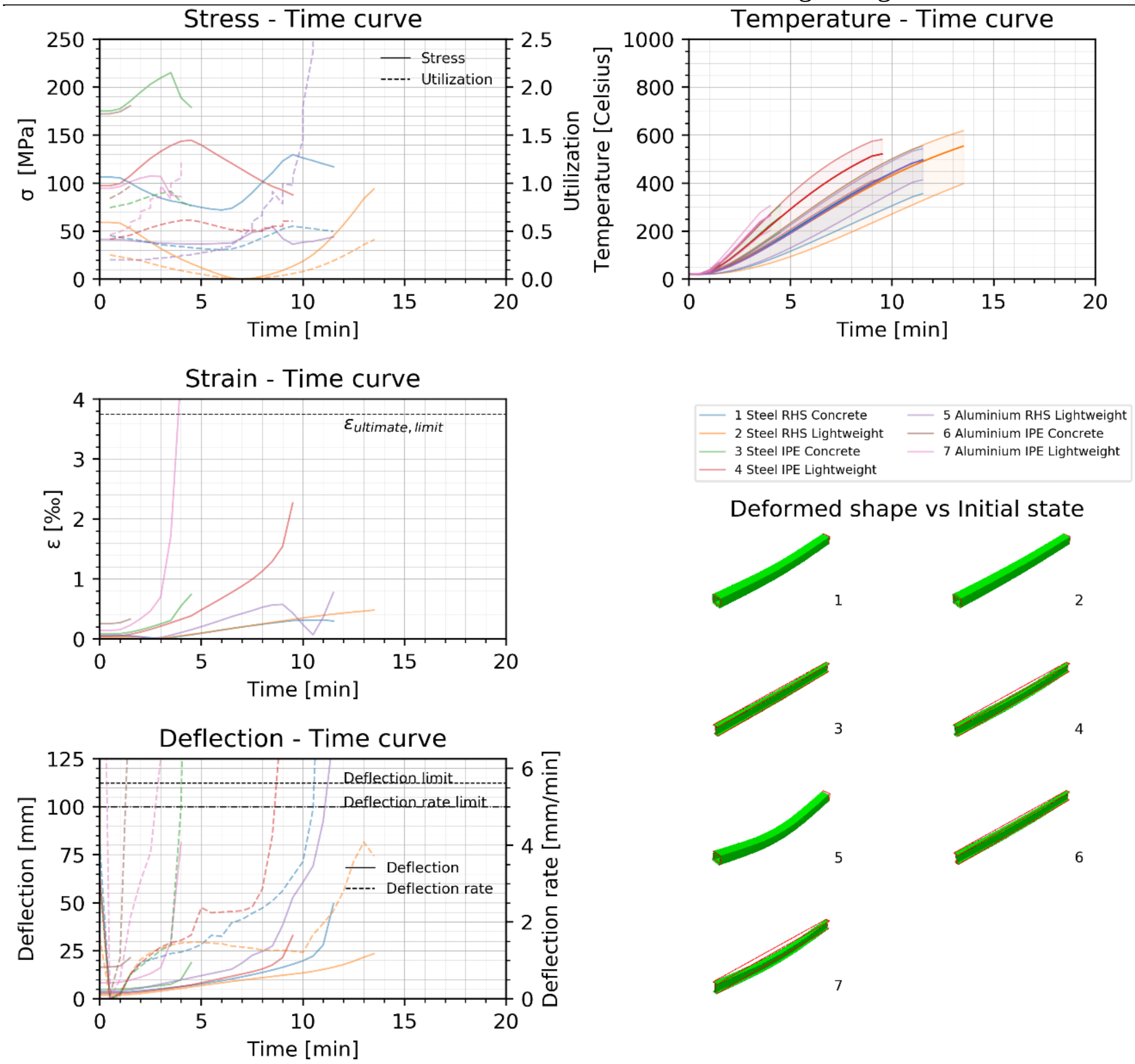

Figure 60 – Results for the full analysis of 3-sided beam in a four point bending test, uninsulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

The effect of the lightweight floor in this case is quite clear, Figure 61. Given the that the lightweight floor was oversimplified and therefore has a higher thermal conductivity. The aluminium cross-section can therefore absorb a lot more heat through this floor system than it would with concrete. Therefore both the RHS and IPE section with the lightweight floor reach the proofstress about twice to thrice as fast as that with a concrete floor. The same behaviour can be observed in Figure 62 for steel in combination with a lightweight floor. Even though the yield stress seems to have been exceeded in these sections, the deformed shape does not seem to support this. Displaying similar thermal expansion reminiscent of the original columns. The strain and deflection values seem to incorporate mechanical and thermal behaviour until proofstress has been reached, and then switch to only thermal expansion. Presumably an effect of the sudden drop of the stress to practically zero while the analysis continues. In this case, the result beyond the forty minute mark is therefore deemed unlikely.

Table 15 – Legend overview for Figure 61 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE concrete insulated	901	Green	Aluminium RHS concrete insulated	776	Blue
Aluminium IPE lightweight insulated	901	Red	Aluminium RHS lightweight insulated	511	Yellow

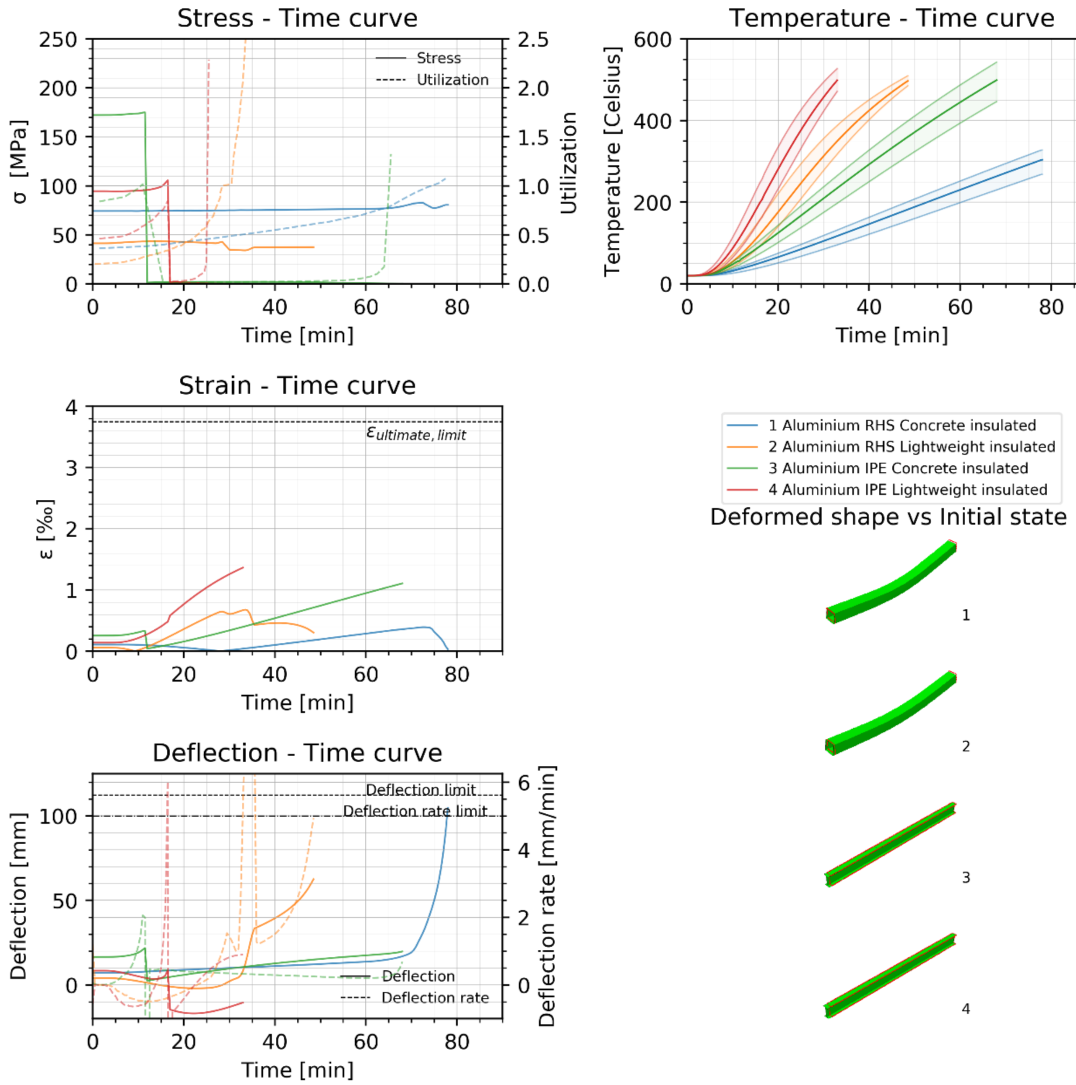

Figure 61 – Results for the full analysis of 3-sided beam in a four point bending test, aluminium insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.3 Four point bending test for larger images of the deformed shape.

Table 16 – Legend overview for Figure 62 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Steel IPE concrete insulated	901	Green	Steel RHS concrete insulated	901	Blue
Steel IPE lightweight insulated	901	Red	Steel RHS lightweight insulated	816	Yellow

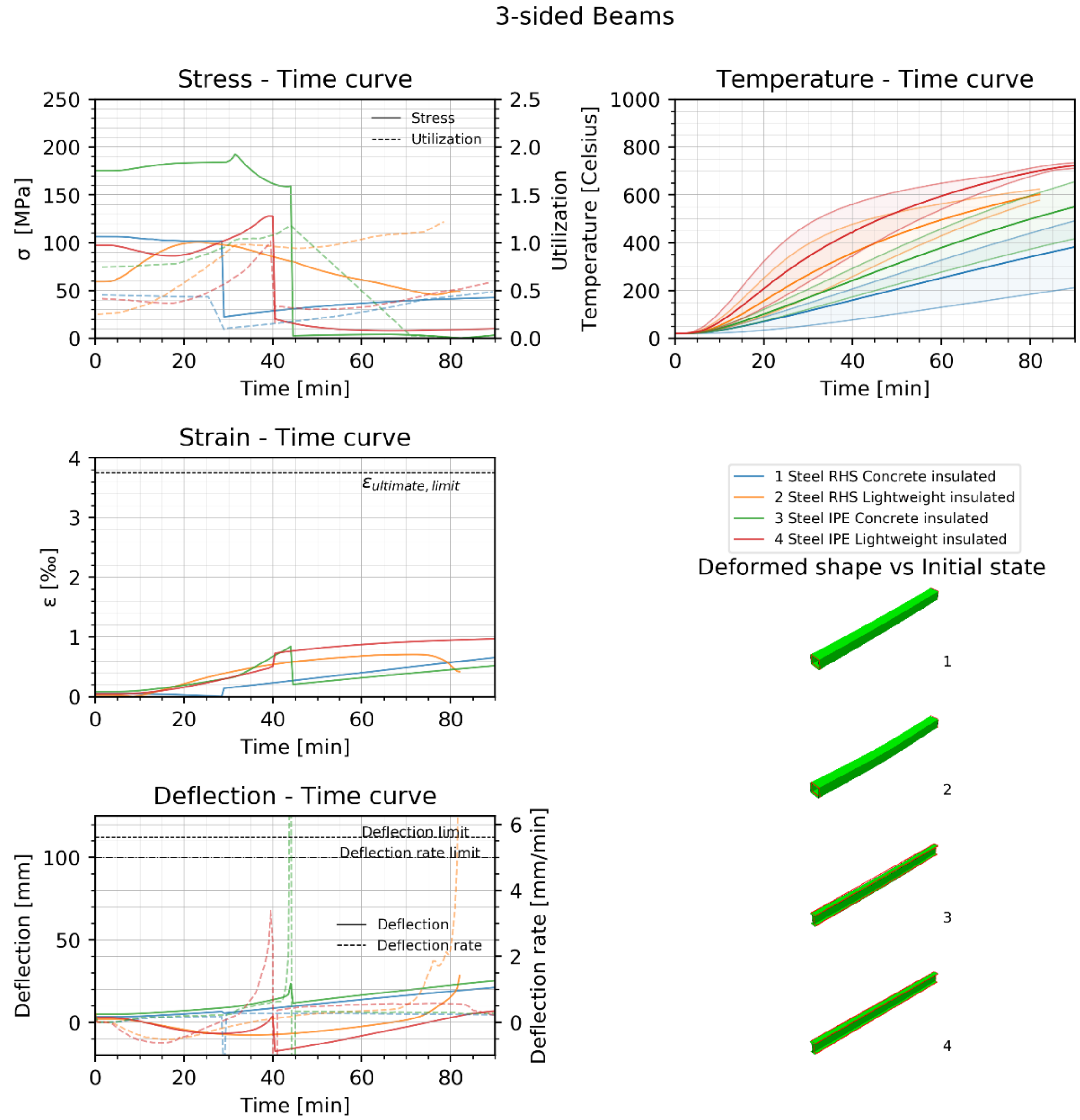

Figure 62 – Results for the full analysis of 3-sided beam with a four point bending test, steel insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.3 Four point bending testD.2 Evenly distributed load. for larger images of the deformed shapes.

6.3 Integrated beam

6.3.1 Evenly distributed load

Due to the fact that with an integrated beam, there is a minimum surface area exposed to the fire load directly. The section can however, gain heat indirectly through the floor which encompasses it. Note that the other side of the cross-section is subject to ambient conditions through which heat can also be lost. This makes it possible to result in larger thermal gradients. This is especially the case for the uninsulated sections in Figure 65 in which the thermal gradient for RHS cross-sections in combination with concrete floors show an unexpectedly large thermal difference. The difference seems exorbitant and unrealistic compared to the gradients found before, also considering the thermal conductivity of the metals themselves.

Given that the insulated cross-sections seem to be even better protected against heat gain, there is a larger number of models which do not reach failure within the time frame, as is with the concrete floor combinations. The same cannot be said for section in combination with the lightweight floors. In Figure 63 the IPE section with the lightweight floor showcases a clear combination of mechanical loading and the effect of thermal bowing. In Figure 64, the same section but with steel does not reach failure, albeit a significant deflection can be observed. This result concurs with the expectation that loaded steel beams showcase larger deformations before failure. In such cases it would therefore be most interesting to proceed with a coupled thermal-mechanical analysis to describe the effect on the heating of the section due to damage to the insulation. The same observation can be made for the uninsulated steel sections in Figure 65.

Table 17 – Legend overview for Figure 63 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE concrete insulated	901	Red	Aluminium RHS concrete insulated	901	Blue
Aluminium IPE lightweight insulated	466	Green	Aluminium RHS lightweight insulated	461	Yellow

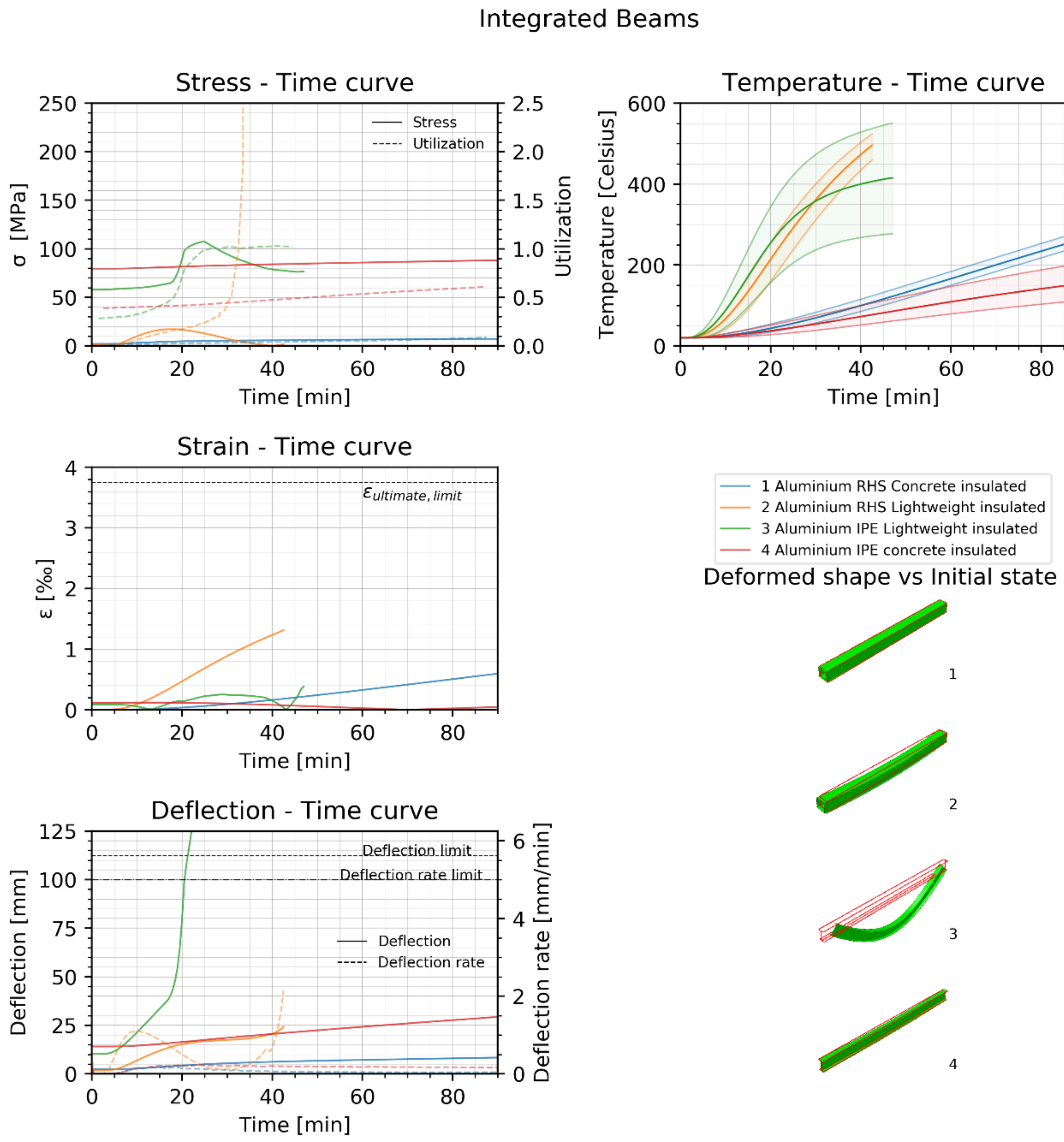

Figure 63 – Results for the full analysis of 3-sided beam with an evenly distributed Q-load, aluminium insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

Table 18 – Legend overview for Figure 64 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Steel IPE lightweight insulated	901	Green	Steel RHS lightweight insulated	246	Blue
Steel IPE concrete insulated	901	Red	Steel RHS concrete insulated	711	Yellow

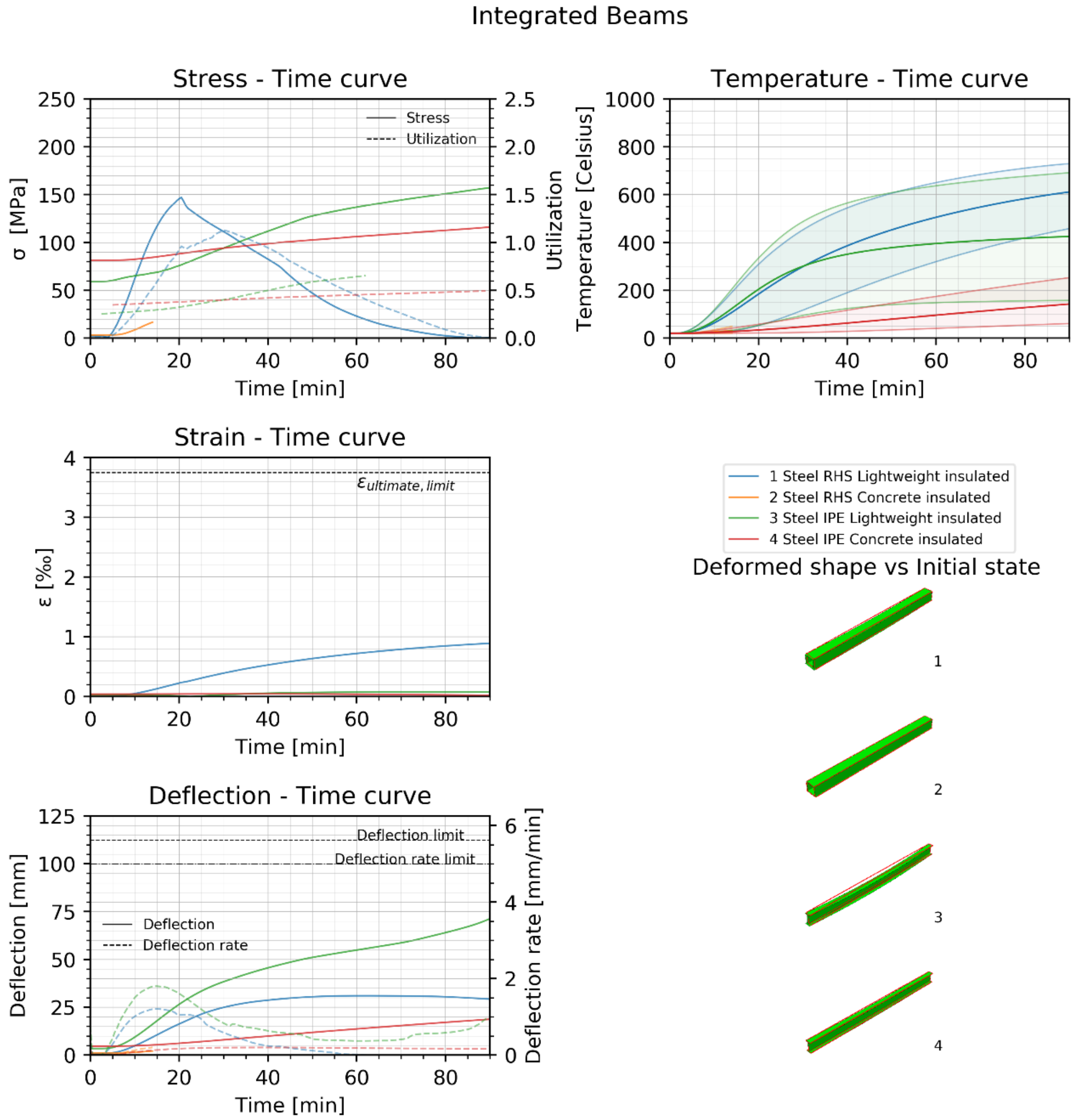

Figure 64 – Results for the full analysis an integrated beam with an evenly distributed Q-load, steel insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

Table 19 – Legend overview for Figure 65 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE lightweight	161	Green	Steel IPE lightweight	296	Pink
Aluminium IPE concrete	166	Red	Steel IPE concrete	351	Grey
Aluminium RHS concrete	206	Blue	Steel RHS lightweight	396	Purple
Aluminium RHS lightweight	186	Yellow	Steel RHS concrete	301	Brown

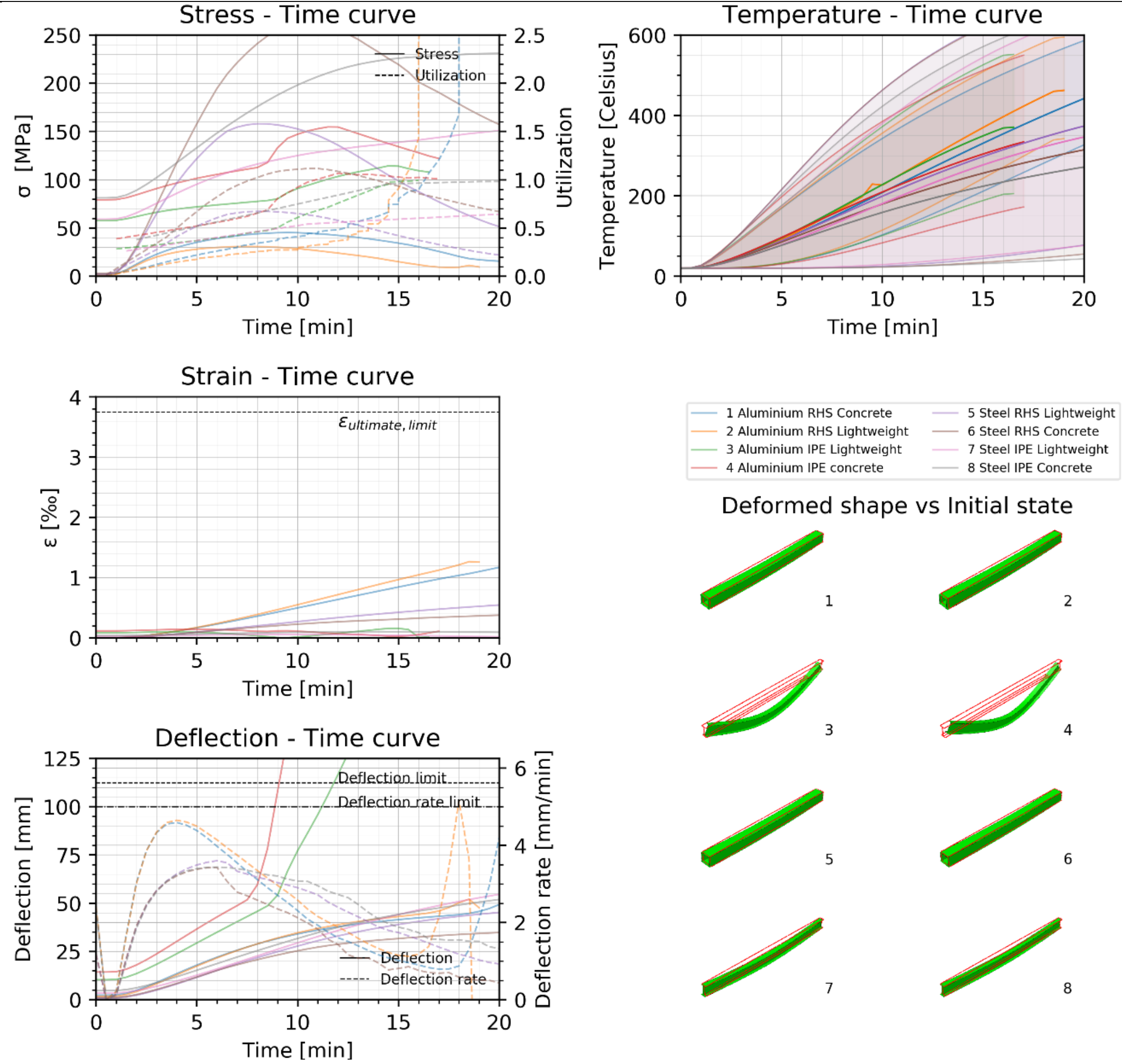

Figure 65 – Results for the full analysis of an integrated beam with an evenly distributed Q-load, uninsulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

6.3.2 Four point bending test

Some of the same observations can be done for the four point bending scenario as with an evenly distributed load. There is a larger number of models which do not reach failure within the time frame. In the case of aluminium, the results in Figure 66 seem have a more gradual effect on the strain development, especially in combination with a concrete floor. In Figure 67, the steel RHS section show very curious stress results. There seems to be an instance of redistribution of the stress through the section. The combination with a concrete floor and steel does not reach failure or any significant deflection. For the lightweight floor though, the statement that loaded steel beams showcase larger deformations before failure. For these cases a coupled thermal-mechanical analysis to describe the effect on the heating of the section due to damage to the insulation would be of interest. As with the evenly distributed load, the same can be said for the steel sections in Figure 68.

Table 20 – Legend overview for Figure 66 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	**Steps**	**Legend**	**Model**	**Steps**	**Legend**
Aluminium RHS lightweight insulated	271	Blue	Aluminium IPE concrete insulated	751	Yellow

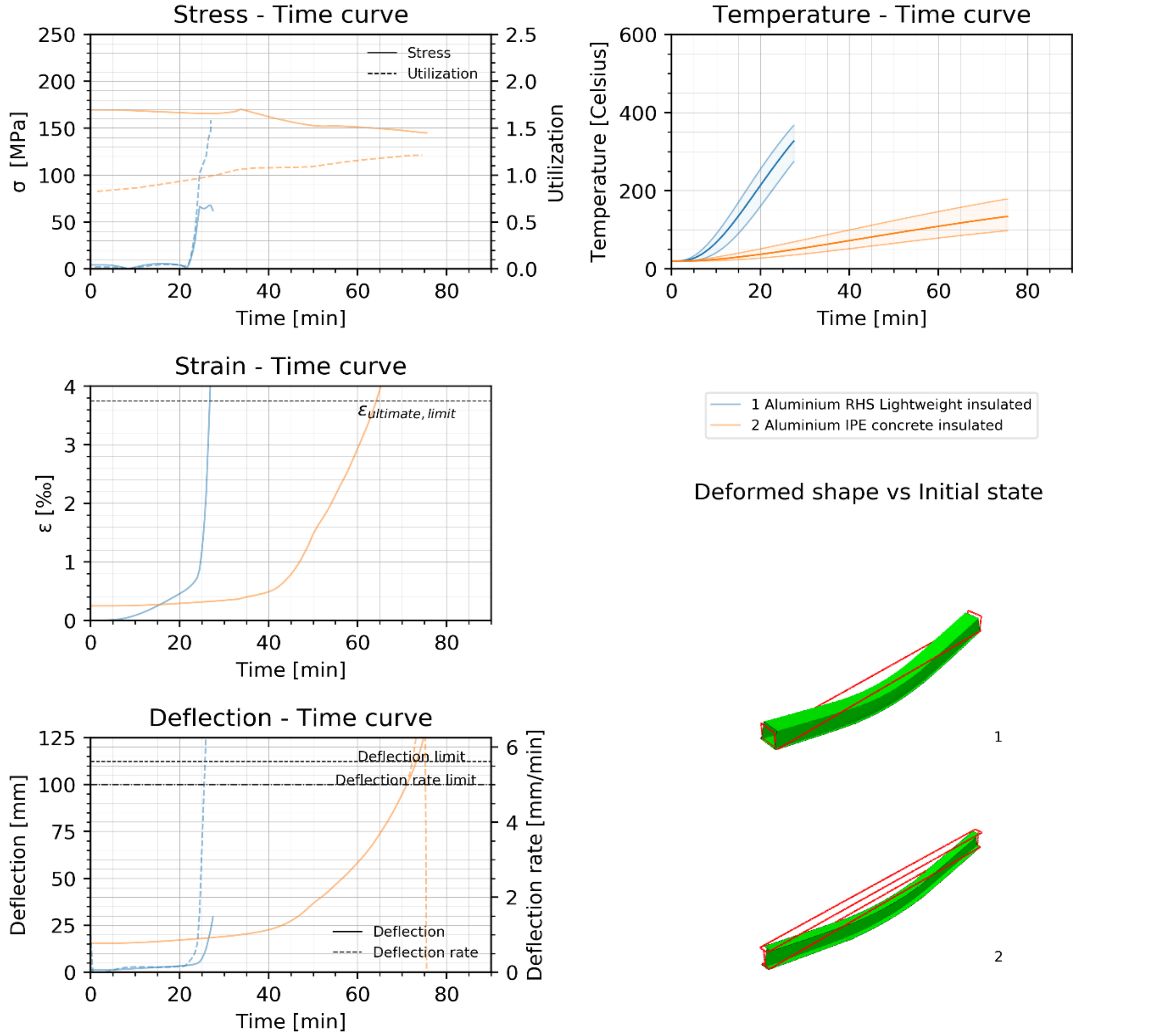

Figure 66 – Results for the full analysis of an integrated beam in a four point bending test, aluminium insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.3 Four point bending test for larger images of the deformed shape.

Table 21 – Legend overview for Figure 67 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	**Steps**	**Legend**	**Model**	**Steps**	**Legend**
Steel IPE lightweight insulated	531	Green	Steel RHS lightweight insulated	516	Blue
Steel IPE concrete insulated	901	Red	Steel RHS concrete insulated	901	Yellow

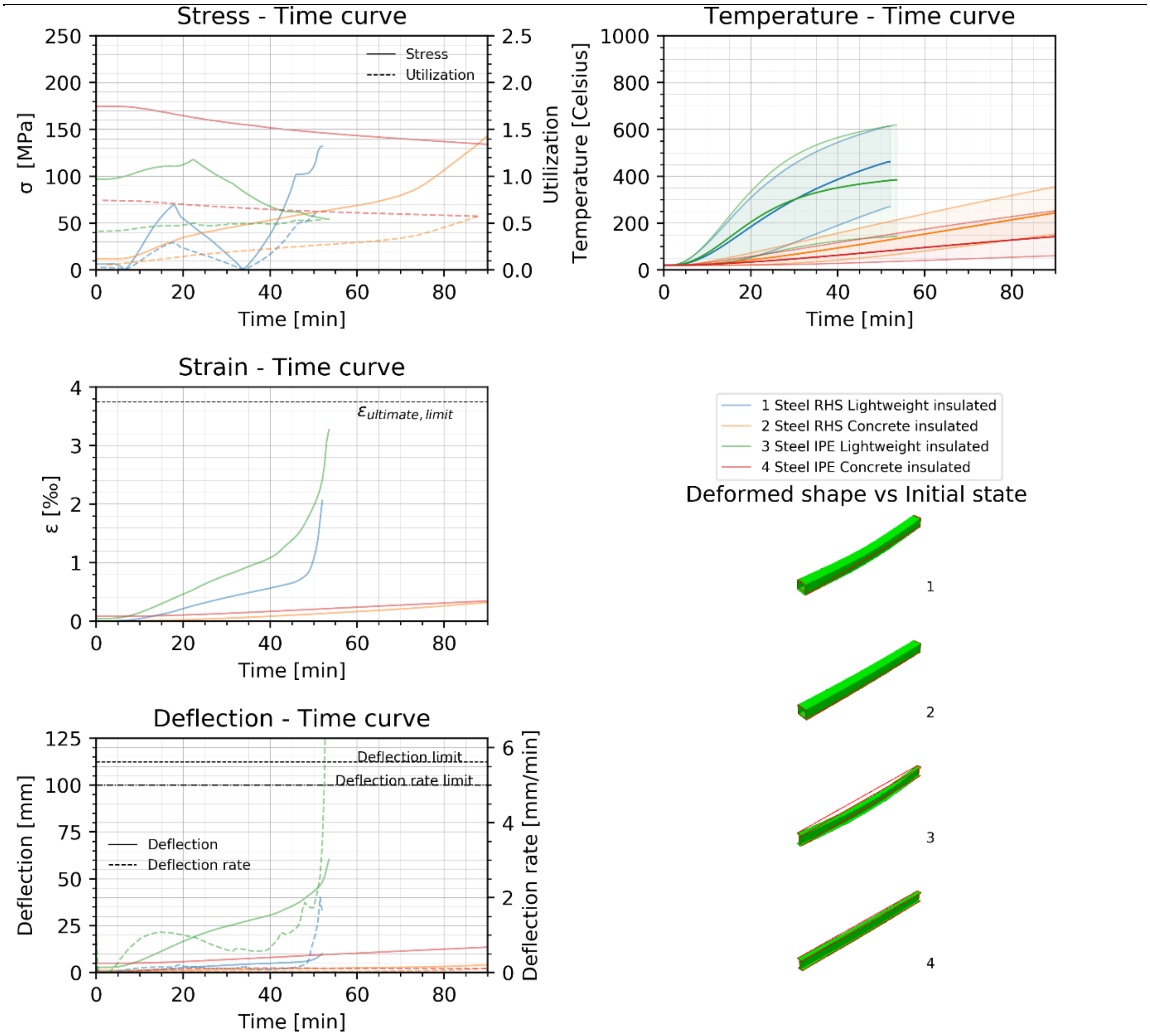

Figure 67 – Results for the full analysis of an integrated beam with a four point bending configuration, steel insulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.3 Four point bending test for larger images of the deformed shapes.

Table 22 – Legend overview for Figure 68 with the number of steps completed in the FEM model. 900 steps confers with 90 minutes which is the full time period over which the separate thermal analysis is run.

Model	Steps	Legend	Model	Steps	Legend
Aluminium IPE concrete	56	Red	Steel IPE lightweight	216	Pink
Aluminium IPE lightweight	101	Green	Steel IPE concrete	191	Grey
Aluminium RHS concrete	96	Blue	Steel RHS lightweight	221	Purple
Aluminium RHS lightweight	151	Yellow	Steel RHS concrete	186	Brown

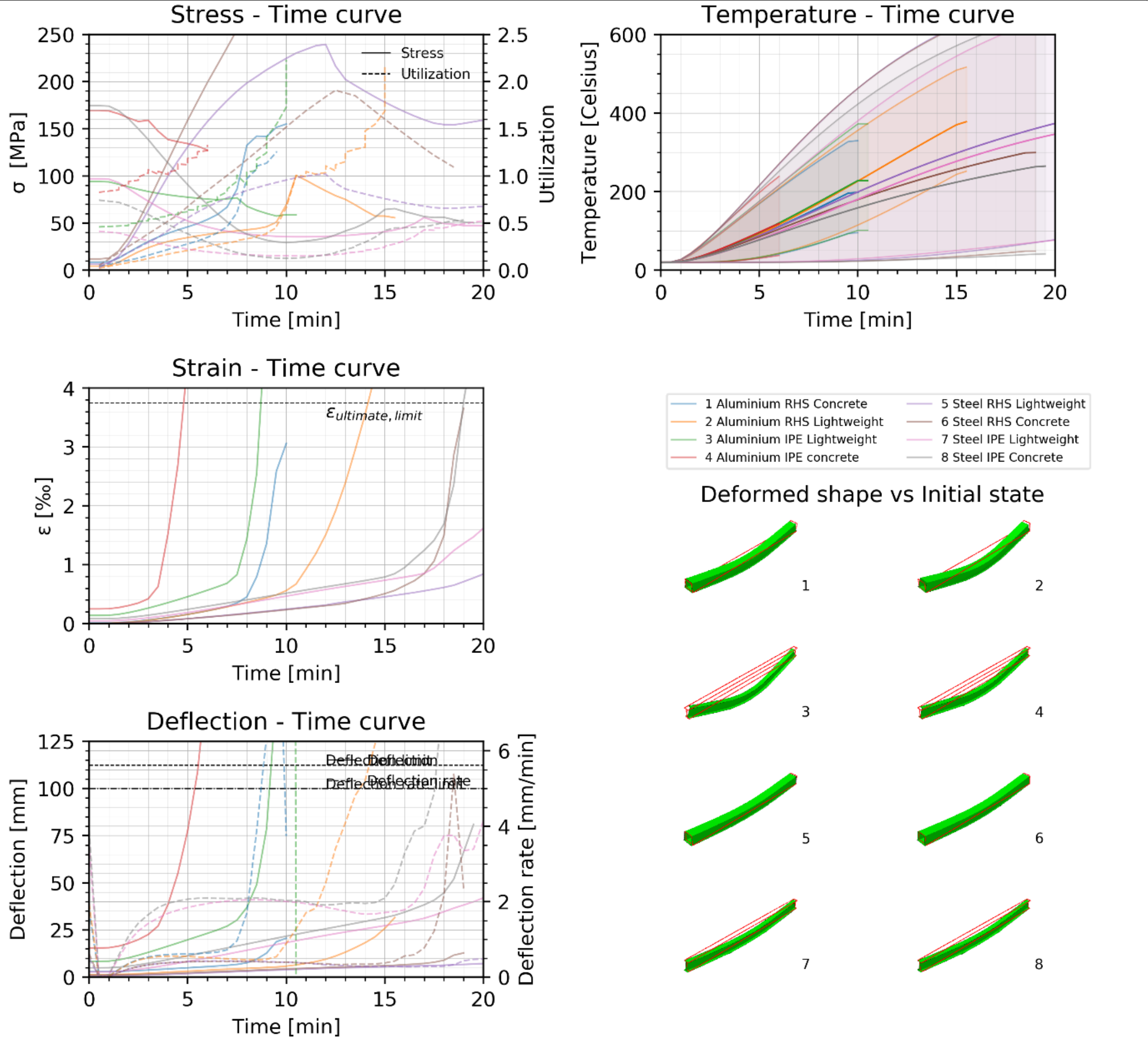

Figure 68 – Results for the full analysis of an integrated beam in a four point bending test, uninsulated sections, showcasing stress, strain, deflection, temperature and the deformed shape, see D.2 Evenly distributed load. for larger images of the deformed shapes.

7. DISCUSSION OF RESULTS

Overall it can be concluded that insulation has a tremendous effect on the temperature increase over time and the implementation of insulation and floor on a beam is determining for the temperature distribution in the cross-section. For aluminium the effect appears to cause the thermal gradient over the cross-section to become more linear, while steel has an inherently larger gradient than aluminium given the fact that it has a lower thermal conductivity.

In particularly concerning the floors, if the floor has a high thermal conductivity and is not insulated while the section is, the heating of the section could be accelerated and the thermal gradient might be inverse to generally expected.

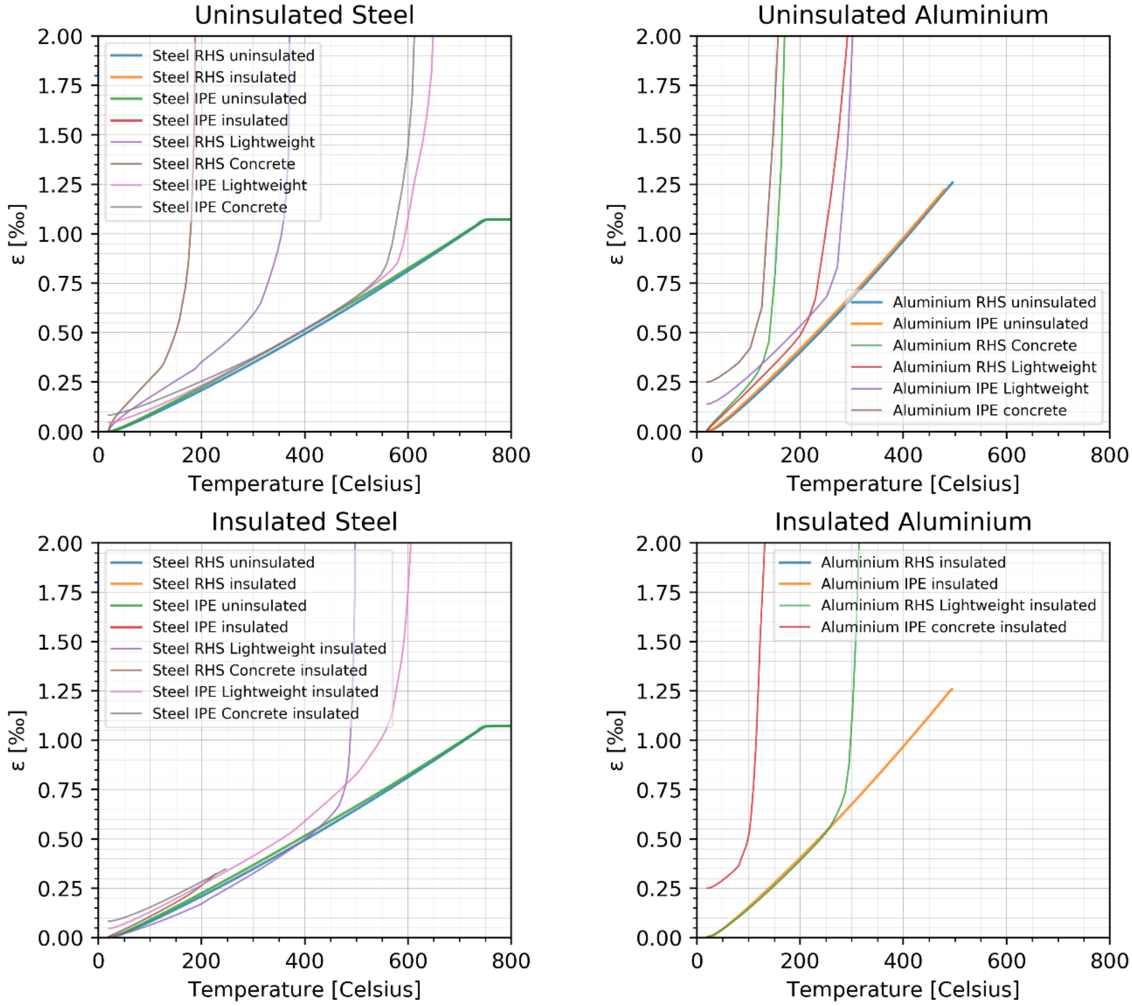

Figure 69 – Comparison of strain-temperature curves for corresponding column types to beam scenario's, in this case for integrated beams subject to a four point bending test.

Plotting the strain results versus temperature of the analysis of the column next to that of the beams should reveal whether there is a distinction to be made between the two. Doing so leads to Figure 69, Figure 70, and Figure 72. In Figure 69 on the right hand side, the results of the steel cross-sections clearly support the fact that steel loaded beam sections show significant sagging before failure. The strain in the case of an insulated IPE section in combination with a lightweight floor shows clear deviation from 400℃ onwards before failure at circa 600 degrees. In contrast, for aluminium, even though the IPE section with a concrete floor (the red line) has a higher starting value that that of the columns, the slope of strain is similar up until rapid failure, further supported by that of an insulated RHS section with a lightweight floor and the uninsulated sections.

A difference in strain magnitude between beams and columns is only of significance when considering the situation in which the insulation is applied. When the insulation is applied in situ, on location when the load is already applied to the section than the magnitude is of little significance. This is due to the fact that the strain at t=0 minutes for a loaded beam may be 0.25‰, the insulation is applied at this point and thus has a strain of zero. However, if the insulation is applied before loading, the strain at start for a loaded beam and insulation is the same and non-zero.

In Figure 72 the established fact that steel shows significant sagging before failure seems not to be supported for an insulated section. However, this is an effect of the data range which has been taken too small to support the theorem in this case. For aluminium though, the strain-temperature curves further support the assessment that the strain difference between column and beam before failure is of much smaller magnitude.

Note in the figures below that for six cases the thermal gradient between temperatures of 50℃ to 350℃ is inverted. Therefore in Figure 70 aluminium RHS with lightweight floor shows a shift in the strain value before failure at 400℃ when the negative thermal bowing deflection is dominated by mechanical failure, as is for aluminium IPE lightweight in Figure 71 uninsulated and in Figure 72 insulated.

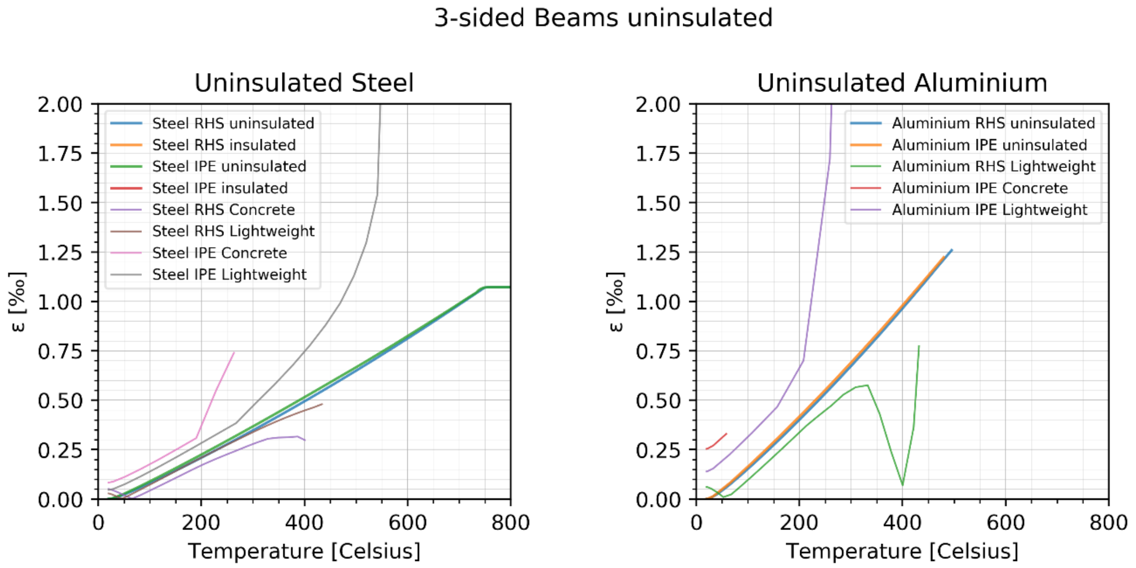

Figure 70 – Comparison of strain-temperature curves for corresponding column types to beam scenario's, in this case for a 3-sided beam subject to a four point bending loading model.

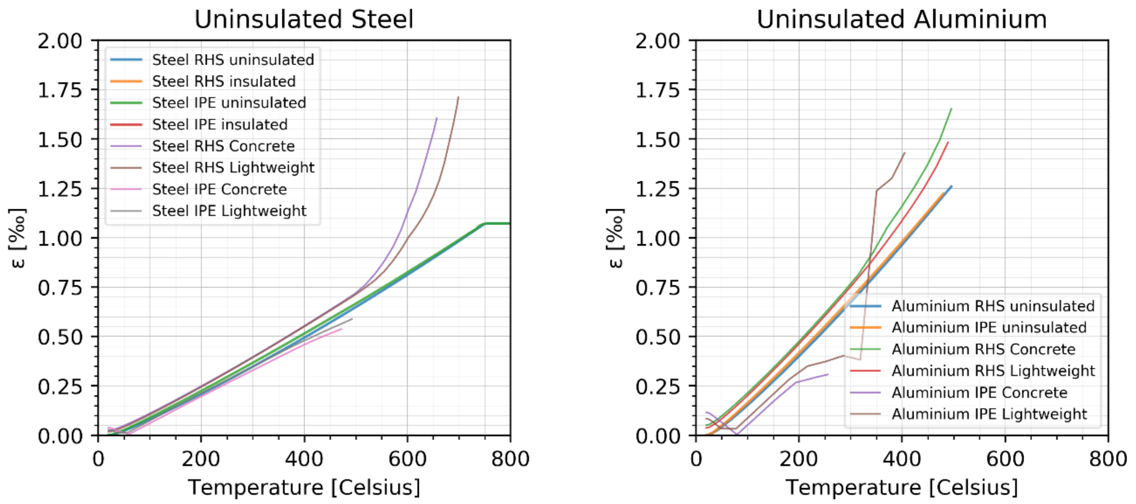

Figure 71 – Comparison of strain-temperature curves for corresponding column types to beam scenario's, in this case for a 3-sided beam subject to an evenly distributed load Q for uninsulated sections.

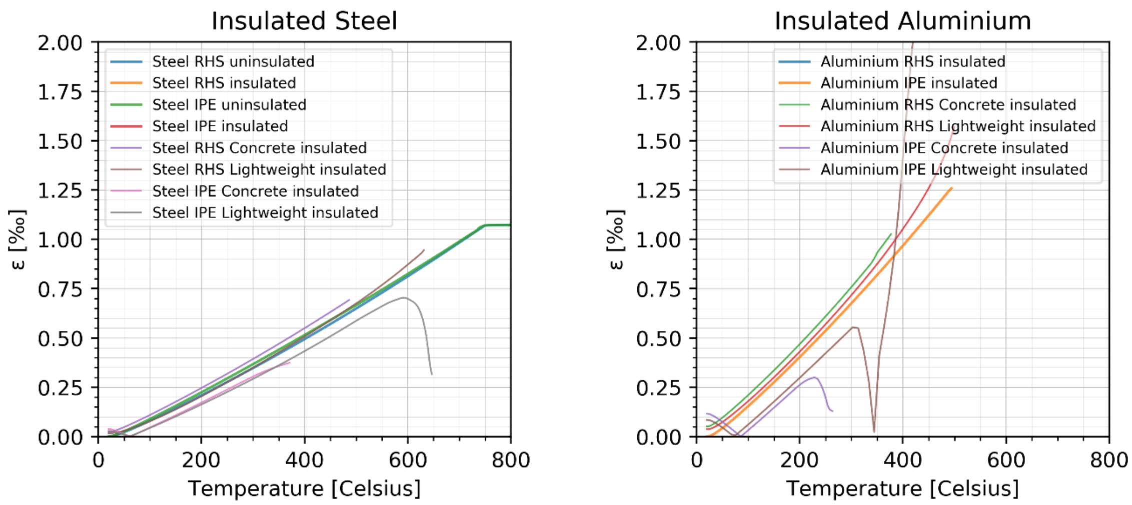

Figure 72 – Comparison of strain-temperature curves for corresponding column types to beam scenario's, in this case for a 3-sided beam subject to an evenly distributed load Q of insulated sections.

This leads to the conclusion that the deformation of a protected aluminium beam exposed to a fire load does **not** differ to any great extent from that of a similar column in such a manner that the protective insulation layer may be damaged prior to failure, and the heating of the beam would be affected. It appears that failure in the case of aluminium happens swiftly when the limiting criteria have been met within a 25℃ range, therefor not implicating the insulation before the critical situation has already been met or is otherwise imminently present.

There are several aspects still subject of debate. This includes the execution of the models itself within the available hardware and software. There was a repetitive occurring error which seemed overly trivial as it had nothing to do with the analysis itself and the results. Apparently the large load on the computer processor caused some Abaqus lock-files to stay active even after finishing an iteration. These lock-files are temporary files to let Abaqus know that a certain analysis is running and while it is, no additional editing can be done. These restrictive files should automatically be deleted after completion of a step and continuing with the restart. However, this seems not to always be the case. This caused an error where the restart for the next iteration could not be achieved. There however, was no indication when this error might occur and a regular purge of cache and outdated model files did not seem to circumvent this issue as a whole and the problem remained present at random intervals. This might have caused some models to be prematurely quit, even though failure or time limits were not exceeded.

During this thesis, more than a hundred varieties were attempted to achieve a full scope of the behaviour. This includes a combination of thermal and mechanical analysis. In some cases the focus might have started to deviate to quantity instead of ensuring quality for each model. The result is a database of more than 1TB of files, which would benefit from a fine tuning to the scenario and specific criteria. In general the time period for the analysis was set constant at 90 minutes while some insulated cases might not have reached failure within this time range.

In addition to these considerations, the validation of the mechanical analysis still leaves questions regarding the exact accuracy of the model specification and why some analysis are unable to run properly. As of yet, this question remains unanswered.

Another undiscussed topic is that of local plasticity. Especially the four point bending models are subject to this effect because the introduction of the load is on a slight area, causing high stresses locally. As can be seen when examining the deformed shapes closely, the places where the loads are introduced are often most heavily distorted. This problem was partially tackled by modelling the area around the introduction point of the load as rigid. However, this did not completely absolve the issue and local failure still occurred in some of the model scenario's. The same behaviour can sometimes be observed in the evenly loaded models, when looking at the supports. Especially in the range of the roller support, the top flange of the IPE section can sometimes be observed to have deflected.

8. CONCLUSION

Returning to questions asked in chapter 2, it is now possible to broker an answer to the question whether the deformation of a protected aluminium beam under fire load differs to that of a similar column, in such a manner that the protective layer is affected and a change in the gradual heating of the beam can be expected. Following the results in chapter 7, there is a positive argument for the omission of full scale loaded beam tests for fire testing with new insulation materials in combination with aluminium. Considering the limit values in EN 13381 and the temperature from which the strain of the beam deviates from the column, to omit the beam test an additional safety margin of 25℃ on the critical temperature for insulated, loaded structures is a recommended. To absolve the need for the loaded aluminium beam test completely however, additional testing is advised to determine if the model fits with an actual fire test, as has been proposed earlier and in chapter 9.

9. FUTURE WORK

There are several angles still left unexplored which would benefit this study further. First and foremost would be the execution of a fire test with aluminium following the recommendations from this report. Given the limited available data, having a more in depth understanding of the material properties from transient state tests could improve the accuracy of the FEM model. In conjunction, the stress-strain relation of aluminium can be improved by considering creep explicitly. In this study, creep has only been implicitly incorporated with adjusted stress-strain curves. However, especially when working with more creep sensitive alloys as the 5000 series would require such an adjustment for primary, secondary and tertiary stage creep as proposed by Dorn-Harmathy [6][14].

Beyond the properties of aluminium, the input values of the insulation in this case have been approximated as true values were unavailable. In addition, it would be of interest to observe (early onset) damage and its effect on the thermal response of the metal specimen. Strain limits have been used to determine when the insulation may incur critical damage. However, due to sensitive corners, damage or other imperfections, the effectiveness of the layer may be compromised. This possibility has been ignored. This is a concern for both paints and other insulation types.

The FEM model itself can be elaborated by considering different loading scenario's, support conditions, geometries such as decking and the definition of contact between surfaces (beam – insulation – flooring). In addition, it is of interest whether a coupled thermal-mechanical analysis may improve the accuracy, especially in case of early onset damage to the insulation. Lastly, the fire conditions can be adjusted to represent a real fire instead of the standard fire curve as to observe a more realistic situation.

10. REFERENCES

[1] Vebon NOVB, “Feiten en Cijfers Branden - VEBON-NOVB,” 2018. [Online]. Available: http://vebon-novb.nl/nl/handige-informatie/feiten-en-cijfers/feiten-en-cijfers-branden. [Accessed: 23-Apr-2018].

[2] CEN, *NEN-EN 1991-1-2:2002+C1:2009+NB:2011 Belasting bij brand*. 2002.

[3] CEN, *NEN-EN 1993-1-2:2005+C2:2009+NB:2015 Eurocode 3: Ontwerp en berekening van staalconstructies bij brand*. NEN, 2005.

[4] CEN, *NEN-EN 1999-1-2:2007/NB:2011 Nationale bijlage bij NEN-EN 1999-1-2 Eurocode 9: Ontwerp en berekening van aluminiumconstructies bij brand*. NEN, 2007.

[5] J. Maljaars, L. Twilt, and F. Soetens, “Flexural buckling of fire exposed aluminium columns,” *Fire Saf. J.*, vol. 44, no. 5, pp. 711–717, 2009.

[6] CEN, *NEN-EN 1363-1:2012 en Bepaling van de brandwerendheid - Deel 1: Algemene eisen*. NEN, 2012.

[7] CEN, *NEN-EN 13381-4:2013 Beproevingsmethoden voor de bepaling van de bijdrage aan de brandwerendheid van constructie-onderdelen - Deel 4: Passieve bescherming aangebracht op stalen constructiedelen*. NEN, 2013.

[8] CEN, *NEN-EN 13501-2:2016 Brandclassificatie van bouwproducten en bouwdelen*, 3rd ed. NEN, 2016.

[9] J. Maljaars, F. Soetens, and L. Katgerman, “Constitutive model for aluminum alloys exposed to fire conditions,” *Metall. Mater. Trans. A Phys. Metall. Mater. Sci.*, vol. 39 A, no. 4, pp. 778–789, 2008.

[10] Morgan Thermal Ceramics UK Ltd., “Fire Protection for Process Equipment hydrocarbon & jet fire protection,” 2017.

[11] Z. Pásztory, T. Horváth, S. V. Glass, and S. Zelinka, “Experimental investigation of the influence of temperature on thermal conductivity of multilayer reflective thermal insulation,” *Energy Build.*, vol. 174, pp. 26–30, Sep. 2018.

[12] Efectis Nederland, “brandwerendheid-testen-in-een-oven-44656 @ www.brandveilig.com,” *Brandveilig.com*, 2016. [Online]. Available: https://www.brandveilig.com/onderwerpen/brandvertraging/brandwerendheid-testen-in-een-oven-44656. [Accessed: 03-Apr-2018].

[13] M. Łukomski, P. Turkowski, P. Roszkowski, and B. Papis, “Fire Resistance of Unprotected Steel Beams-Comparison between Fire Tests and Calculation Models,” *Procedia Eng.*, vol. 172, pp. 665–672, 2017.

[14] Technische Commissie CEN/TC 250, *NEN-EN 1991-1-1:2002+C1:2009+NB:2011 Eurocode 1: Belastingen op constructies*, no. December 2011. 2002.

[15] S. Jiang, Z. Xiong, X. Guo, and Z. He, “Buckling behaviour of aluminium alloy columns under fire conditions,” *Thin-Walled Struct.*, vol. 124, no. 1239, pp. 523–537, 2018.

[16] E. Kandare, S. Feih, A. Kootsookos, Z. Mathys, B. Y. Lattimer, and A. P. Mouritz, “Creep-based life prediction modelling of aluminium in fire,” *Mater. Sci. Eng. A*, vol. 527, no. 4–5, pp. 1185–1193, 2010.

[17] J. R. Cannon, *The One-Dimensional Heat Equation - Encyclopedia of Mathematics and its applications*, 1st ed. Menlo Park, California: Addison-Wesley Publishing company/Camrbidge University Press, 1984.

[18] M. B. Wong and J. I. Ghojel, “Sensitivity analysis of heat transfer formulations for insulated structural steel components,” *Fire Saf. J.*, vol. 38, no. 2, pp. 187–201, Mar. 2003.

[19] C. Rippe, S. Case, and B. Lattimer, “Modeling post-fire behavior of aluminum structural components using a maximum temperature approach,” *Fire Saf. J.*, vol. 91, no. February, pp. 561–567, 2017.

[20] J. Maljaars, F. Soetens, and H. H. Snijder, “Local buckling of aluminium structures exposed to fire. Part 1: Tests,” *Thin-Walled Struct.*, vol. 47, no. 11, pp. 1404–1417, 2009.

[21] J. Maljaars, F. Soetens, and H. H. Snijder, “Local buckling of aluminium structures exposed to fire. Part 2: Finite element models,” *Thin-Walled Struct.*, vol. 47, no. 11, pp. 1418–1428, 2009.

[22] O. Delgado Ojeda, J. Maljaars, and R. Abspoel, “Fire exposed steel columns with a thermal gradient over the cross-section,” *Thin-Walled Struct.*, vol. 98, pp. 103–110, 2016.

[23] I. T. G. Van Der Waart Van Gulik, "BOUWEN MET STAAL TECHNISCHE COMMISSIE 3 FSE SEMINAR-16 APRIL 2015 (TRONET) BRANDWERENDE COATING," 2015.
[24] Morgan Thermal Ceramics UK Ltd., "Denka ® Alcen ® Blankets," 2016.
[25] Morgan Thermal Ceramics UK Ltd., "Data sheet FireMaster ® Marine Plus blanket," 2018.
[26] Morgan Thermal Ceramics UK Ltd., "Kaowool^{TM} Blanket S/Kaowool^{TM} Blanket SZr."
[27] Insulcon, "Technical Datasheet Fyrewrap LT blanket," vol. 49, no. 0, pp. 31–33.
[28] W.-Y. Wang and G.-Q. Li, "Behavior of steel columns in a fire with partial damage to fire protection," *J. Constr. Steel Res.*, vol. 65, no. 6, pp. 1392–1400, Jun. 2009.
[29] W. Chen, J. Ye, and X. Li, "Thermal behavior of gypsum-sheathed cold-formed steel composite assemblies under fire conditions," *J. Constr. Steel Res.*, vol. 149, pp. 165–179, Oct. 2018.
[30] J. Ding and Y. C. Wang, "Realistic modelling of thermal and structural behaviour of unprotected concrete filled tubular columns in fire," *J. Constr. Steel Res.*, vol. 64, no. 10, pp. 1086–1102, 2008.
[31] A. Espinos, M. L. Romero, and A. Hospitaler, "Advanced model for predicting the fire response of concrete filled tubular columns," *J. Constr. Steel Res.*, vol. 66, no. 8–9, pp. 1030–1046, Aug. 2010.
[32] K. Xiang, G.-H. Wang, and Y.-C. Pan, "Thermal Properties and Heat Transfer in Concrete Filled Steel Tube Reinforced Concrete Columns Exposed to Fire," in *2014 7th International Conference on Intelligent Computation Technology and Automation*, 2014, pp. 869–874.
[33] W. E. Luecke, McColskey, C. N. McCowan, and F. W. Gayle, *Mechanical Properties of Structural Steels*. USA: National Institute of Standards and Technology, 2005.
[34] B. Barthelemy, "Heating Calculation of Structural Steel Members," *J. Struct. Div.*, vol. 102, no. 8, pp. 1549–1558, 1976.
[35] J. G. Kaufman, *Parametric Analyses of High Temperature Data for Aluminum Alloys*, 1st ed. Metals Park, OH: ASM International, 2008.
[36] P. Martin, "Thermoelectric Materials and Applications," *Battelle Pacific Northwest Lab. new bullitin*, no. summer, pp. 30–31, 2005.
[37] S. Fan, B. He, X. Xia, H. Gui, and M. Liu, "Fire resistance of stainless steel beams with rectangular hollow section: Experimental investigation," *Fire Saf. J.*, vol. 81, pp. 17–31, 2016.
[38] I. W. Burgess, J. El Rimawi, and R. J. Plank, "Studies of the Behaviour of Steel Beams in Fire," *J. Constr. Res.*, vol. 19, pp. 285–312, 1991.
[39] A. J. P. Moura Correia, J. P. C. Rodrigues, and P. V. Real, "Thermal bowing on steel columns embedded on walls under fire conditions," *Fire Saf. J.*, vol. 67, pp. 53–69, Jul. 2014.
[40] M. M. S. Dwaikat, V. K. R. Kodur, S. E. Quiel, and M. E. M. Garlock, "Experimental behavior of steel beam–columns subjected to fire-induced thermal gradients," *J. Constr. Steel Res.*, vol. 67, no. 1, pp. 30–38, Jan. 2011.
[41] A. Agarwal, L. Choe, and A. H. Varma, "Fire design of steel columns: Effects of thermal gradients," *J. Constr. Steel Res.*, vol. 93, pp. 107–118, Feb. 2014.
[42] J. Cai and J. Feng, "Thermal buckling of rotationally restrained steel columns," *J. Constr. Steel Res.*, vol. 66, no. 6, pp. 835–841, Jun. 2010.
[43] J. Cai, J. Feng, and J. Zhang, "Thermoelastic buckling of steel columns with load-dependent supports," *Int. J. Non. Linear. Mech.*, vol. 47, no. 4, pp. 8–15, May 2012.
[44] J. G. Kaufman, *Properties of Aluminium Alloys—Tensile, Creep, and Fatigue Data at High and Low Temperatures*, 1st ed. Metals park, OH: ASM International, 1999.
[45] P. T. Summers, S. W. Case, and B. Y. Lattimer, "Residual mechanical properties of aluminum alloys AA5083-H116 and AA6061-T651 after fire," *Eng. Struct.*, vol. 76, pp. 49–61, 2014.
[46] S. Selamet and M. E. Garlock, "Predicting the maximum compressive beam axial force during fire considering local buckling," *J. Constr. Steel Res.*, vol. 71, pp. 189–201, 2012.
[47] Z. Chen, J. Lu, H. Liu, and X. Liao, "Experimental investigation on the post-fire mechanical properties of structural aluminum alloys 6061-T6 and 7075-T73," *Thin-Walled Struct.*, vol. 106, pp. 187–200, 2016.
[48] W. Y. Wang and G. Q. Li, "Fire-resistance study of restrained steel columns with partial damage to fire protection," *Fire Saf. J.*, vol. 44, no. 8, pp. 1088–1094, 2009.
[49] M. S. (Mohammed S. . El Naschie, *Stress, stability, and chaos in structural engineering : an energy approach*. McGraw-Hill, 1990.

[50] Cooke G., "The structural response of steel I-section members subjected to elevated temperature gradients across the section," City University London, 1987.
[51] Technische Commissie CEN/TC 250, "NEN-EN 1990 Grondslagen van het constructief ontwerp," 2014.
[52] M. Liu, L. Zhang, P. Wang, and Y. Chang, "Buckling behaviors of section aluminum alloy columns under axial compression," *Eng. Struct.*, vol. 95, pp. 127–137, 2015.
[53] E. C.-Y. To and B. Young, "Performance of cold-formed stainless steel tubular columns at elevated temperatures," *Eng. Struct.*, vol. 30, no. 7, pp. 2012–2021, Jul. 2008.
[54] csengineer, "02-03-AlumaBridge-Deck.jpg (365×212)," 2014. [Online]. Available: https://csengineermag.com/archived_assets/cdn/2014/06/CS_Prod-SoftwareGuide/02-03-AlumaBridge-Deck.jpg. [Accessed: 06-Aug-2019].
[55] CEN, *NEN-EN 1993-1-1:2006+A1:2014+NB:2016 Staal - Algemene regels*. NEN, 2016.
[56] CEN, *NEN-EN 1999-1-1:2007+A2:2014+NB:2011 Eurocode 9: Ontwerp en berekening van aluminiumconstructies*. NEN, 2007.
[57] S. F. A. J. G. Zegers, *Lightweight floor system for vibration comfort*. Eindhoven: Technische Universiteit Eindhoven, 2011.
[58] D. de Silva, A. Bilotta, and E. Nigro, "Experimental investigation on steel elements protected with intumescent coating," *Constr. Build. Mater.*, vol. 205, pp. 232–244, Apr. 2019.
[59] J. Witteveen and L. Twilt, "Behaviour of steel columns under fire action," *IABSE*, vol. 23, 1975.

11. APPENDICES

A: List of figures and tables

B: Mechanical analysis with intumescent paint

C: Mechanical analysis with sandwich floor

D: FEM images of deformed model shapes

E: FEM thermal analysis script

F: FEM mechanical analysis script

G: Postprocessing script

A: LIST OF FIGURES AND TABLES

B: MECHANICAL ANALYSIS WITH INTUMESCENT PAINT

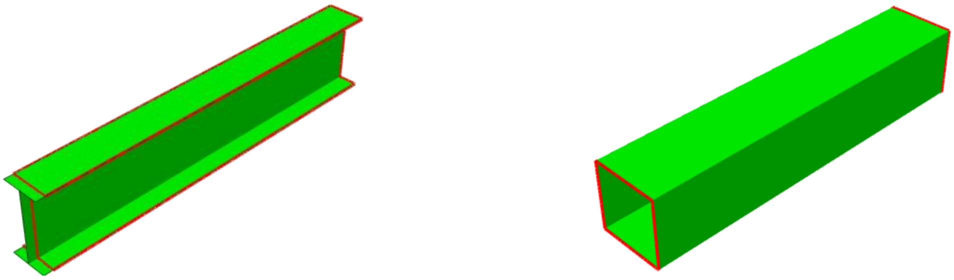

Figure 73 - Aluminium column insulated with intumescent paint.

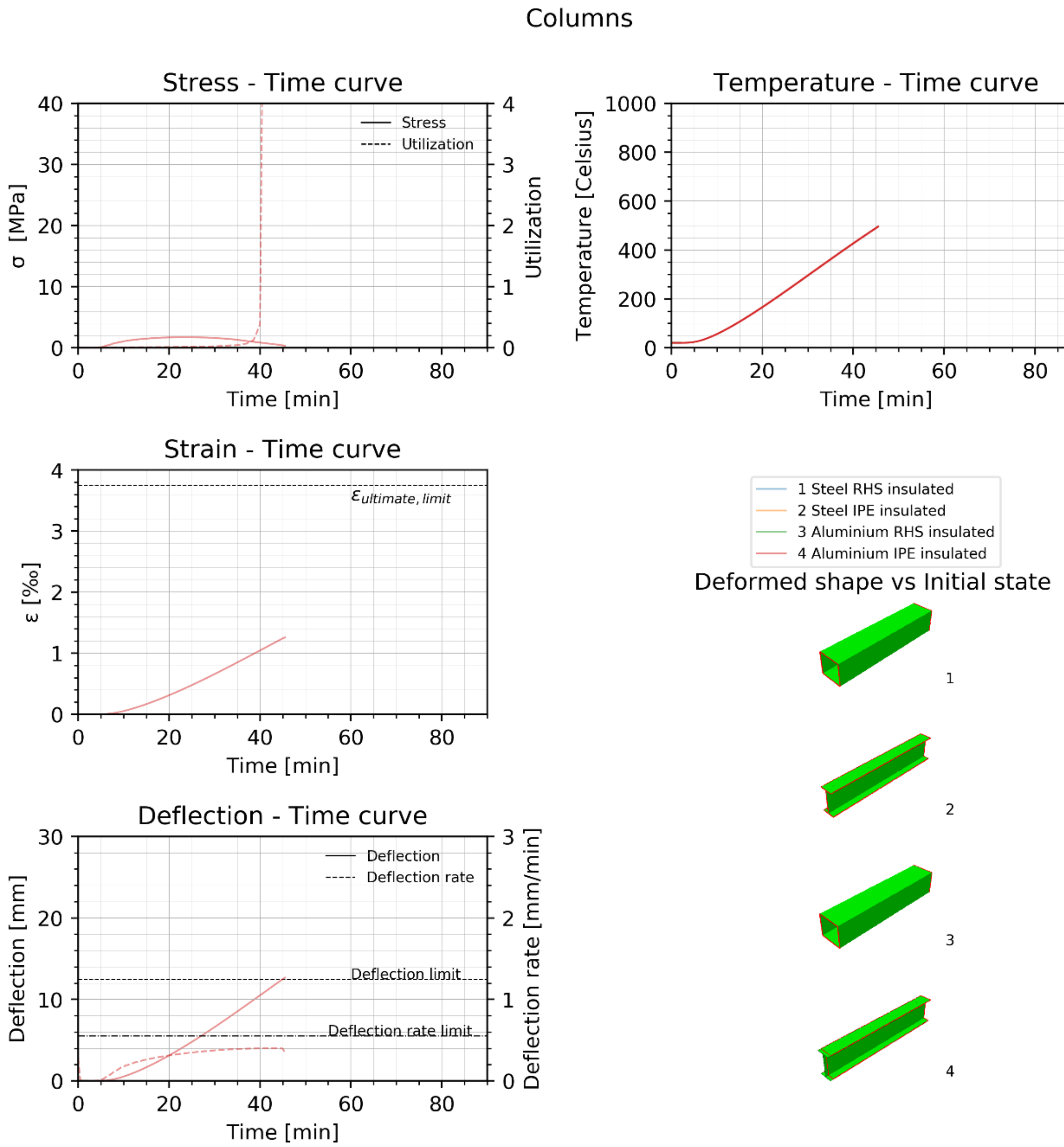

Figure 74 – Column covered with a layer of intumescent paint.

As expressed earlier, this model also faced running issues. However that for an aluminium IPE insulated section was successful. In comparison with the columns as discussed in 6.1 Column, the failure time is slightly increased. The strain and deflection fit with earlier found relations. The same can be said for the results found with 3-sided beams in both load situations. Especially for those in combination with a lightweight floor, for which the thermal gradient may be inverted as is with the insulated IPE section.

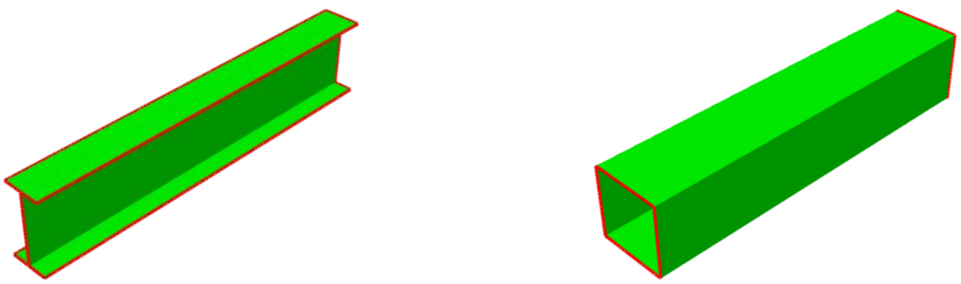

Figure 75 - Steel column with intumescent paint

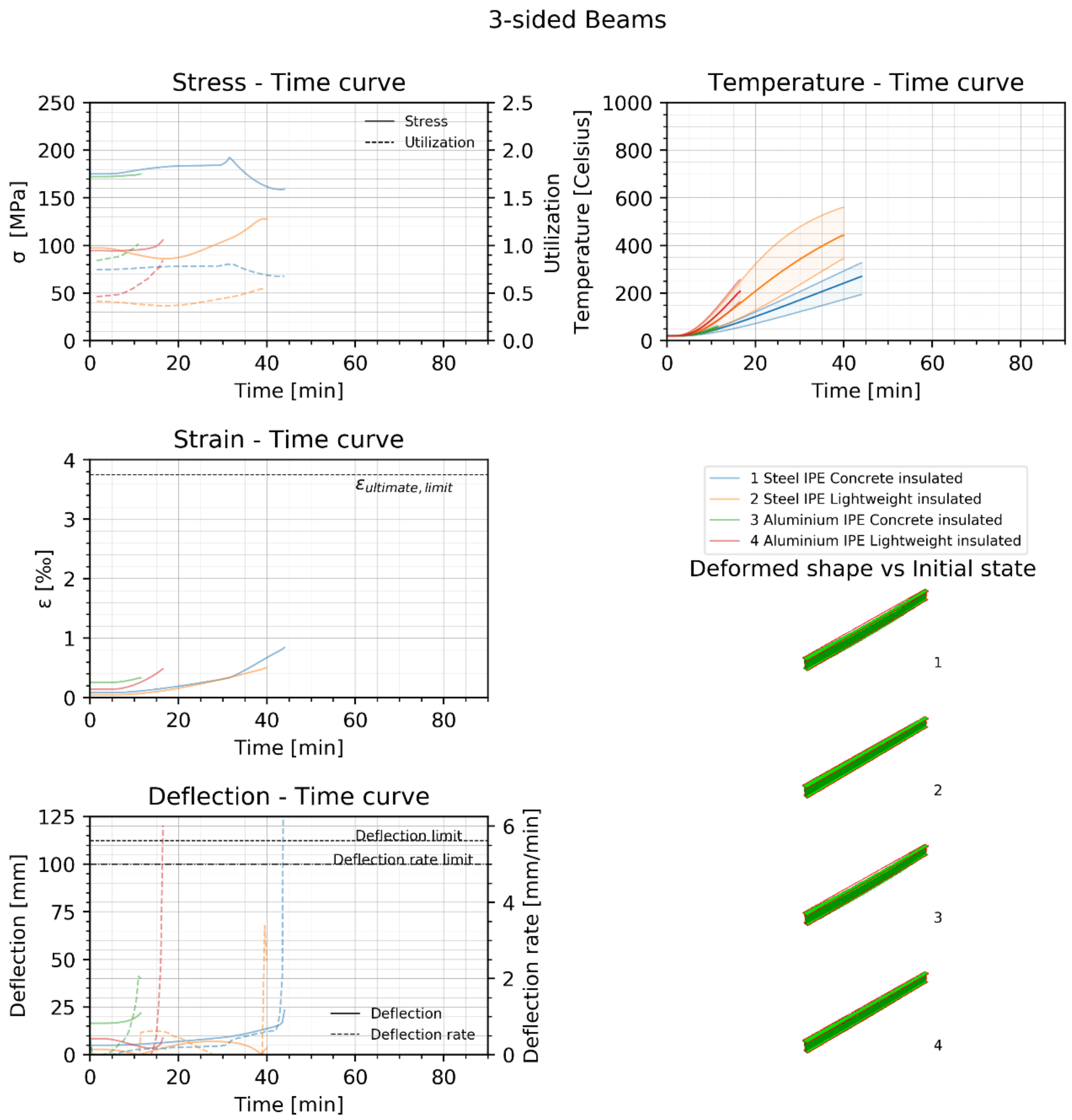

Figure 76 - 3-sided beam with an intumescent paint layer in a four point bending test.

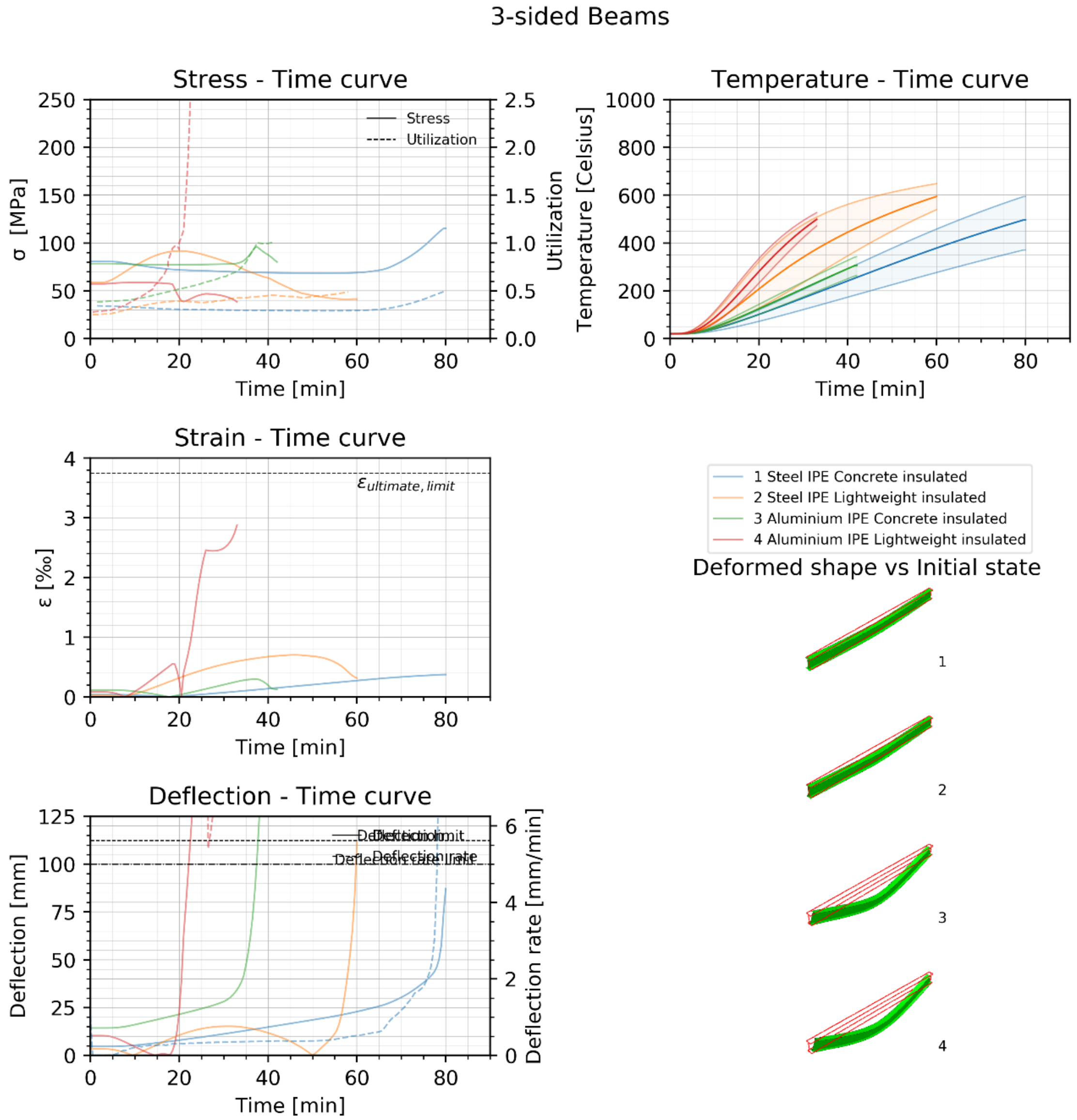

Figure 77 – 3-sided beam with intumescent paint layer with an evenly distributed load.

C: MECHANICAL ANALYSIS WITH SANDWICH FLOOR

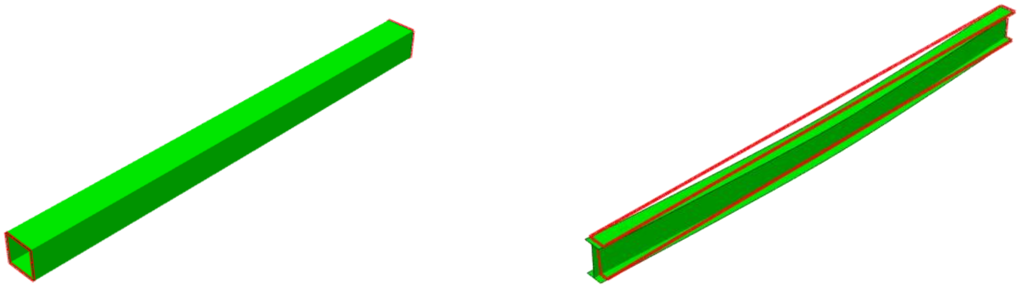

Figure 78 - Steel 3-sided beam with evenly distributed load, insulated.

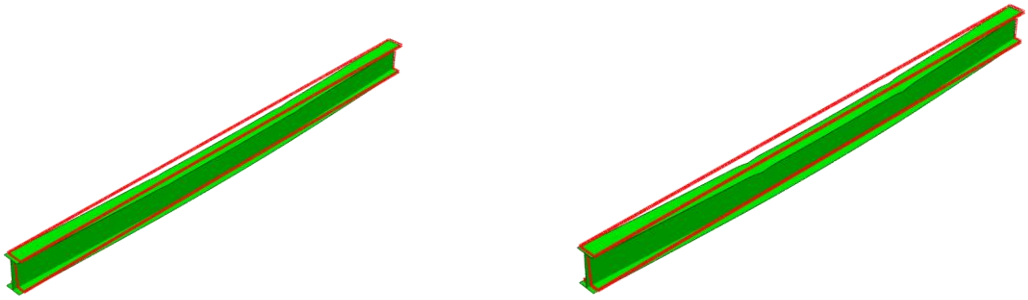

Figure 79 - Steel 3-sided beam with four sided beam insulated, right aluminium 3-sided beam with four sided load.

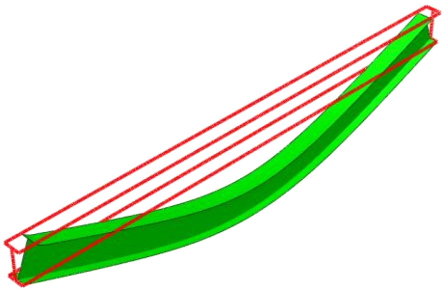

Figure 80 - Aluminium 3-sided beam with lightweight floor with evenly distributed load, insulated.

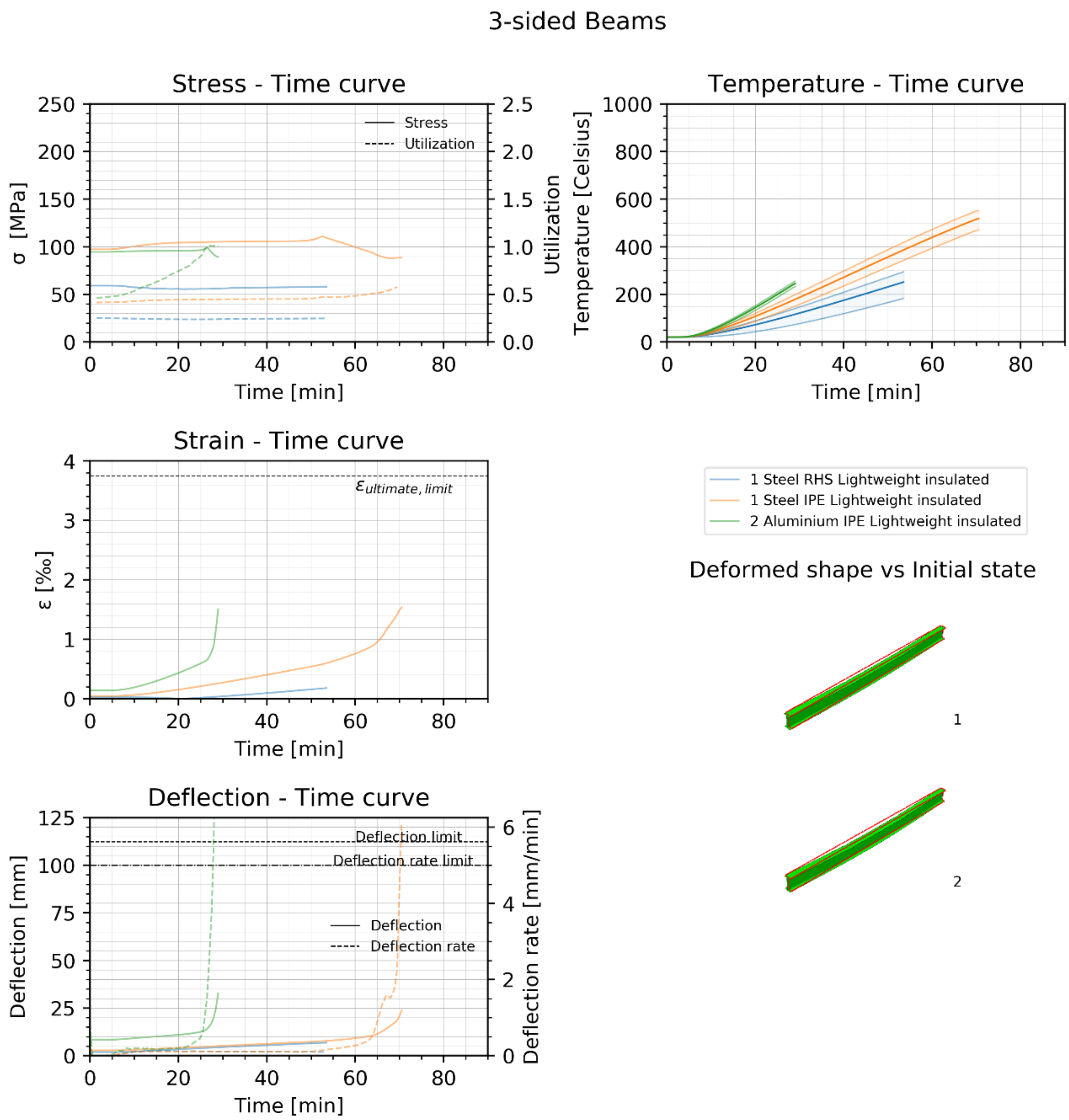

Figure 81 – 3-sided beam with an alternative lightweight floor system, representative of a sandwich system and its effect on the temperature distribution. Load configuration as a four point bending test.

As discussed in an earlier chapter, the lightweight floor has a direct effect on the thermal gradient in the section. Assuming that the sandwich panel is better insulated due too its layered built, the exposed side of the beam is better protected and thus heating of the section is slowed. As a result to less exposure the thermal gradient is also found to be considerable less. The effect on the temperature development is observed in both load cases, that of an evenly distributed load and an four point bending test. The behaviour of the strain and deflection fit with aforementioned patterns.

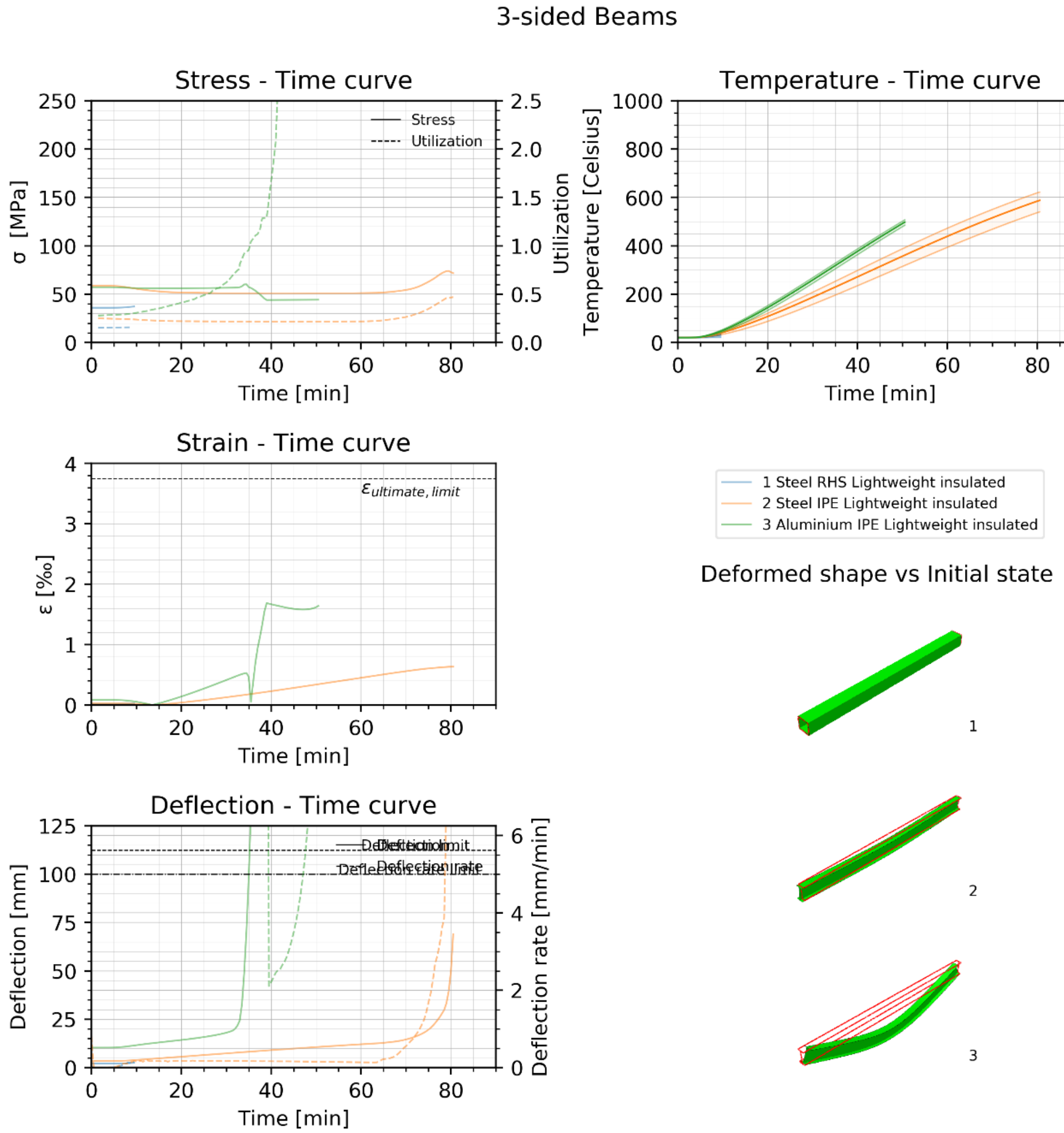

Figure 82 – 3-sided beam with an evenly distributed load with an alternative lightweight floor system, namely that of a sandwich panel.

D: FEM IMAGES OF DEFORMED MODEL SHAPES

Original shape is outlined in red. The green shape is the deformed shape with a scalefactor of 3.

D.1 Columns

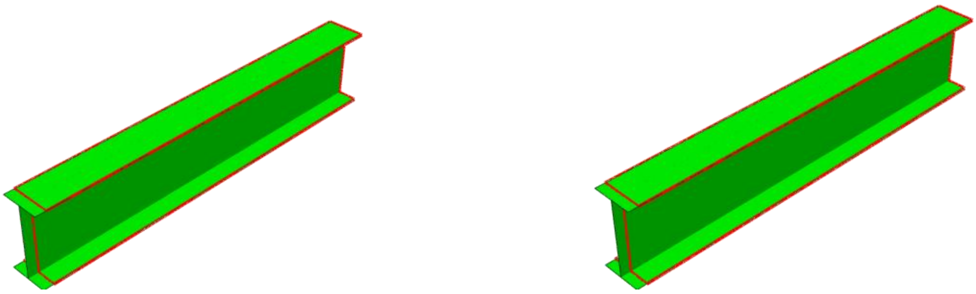

Figure 83 - Aluminium Column IPE uninsulated left, insulated right

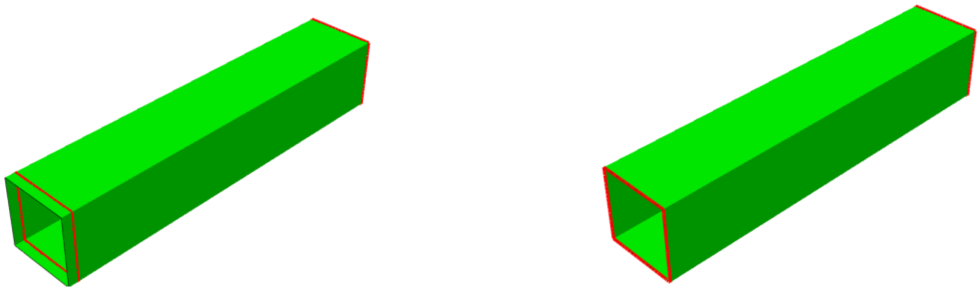

Figure 84 - Aluminium column RHS uninsulated left, insulated right

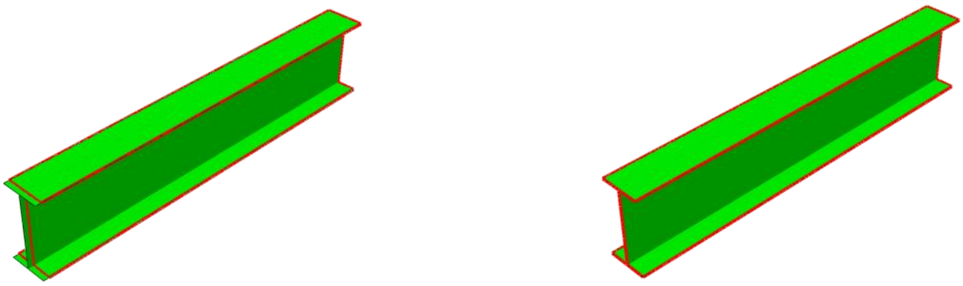

Figure 85 - Steel column IPE section uninsulated left, insulated right

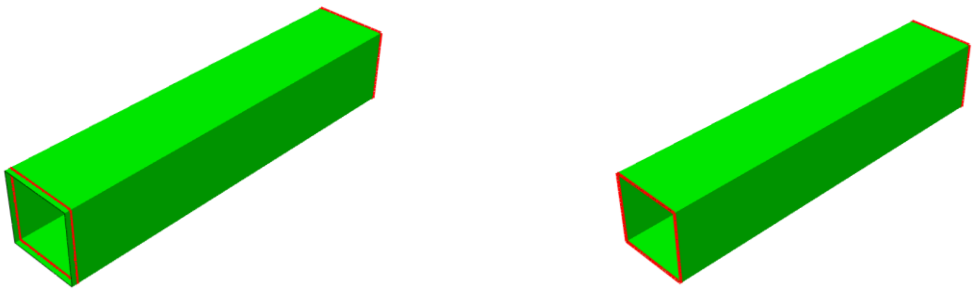

Figure 86 - Steel column RHS section uninsulated left, insulated right

D.2 Evenly distributed load.

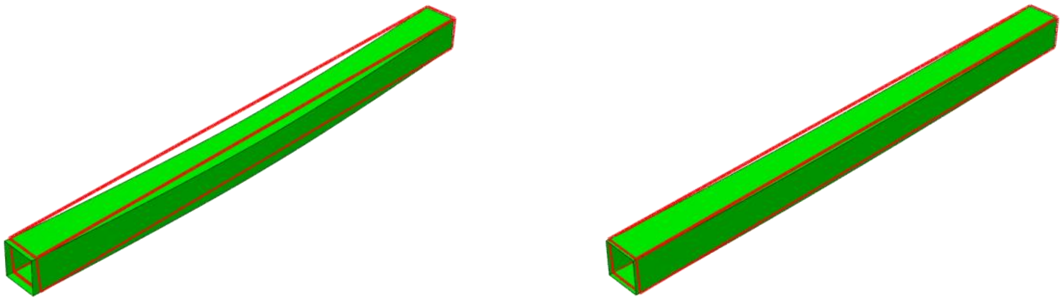

Figure 87 – Steel 3-sided RHS beam with a concrete floor left uninsulated, right insulated with a distributed load Q

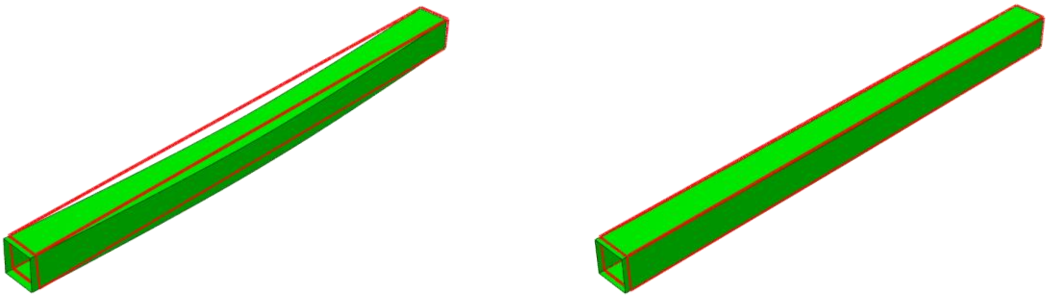

Figure 88 – Steel 3-sided beam RHS with a lightweight floor, left uninsulated, right insulated with a distributed load Q

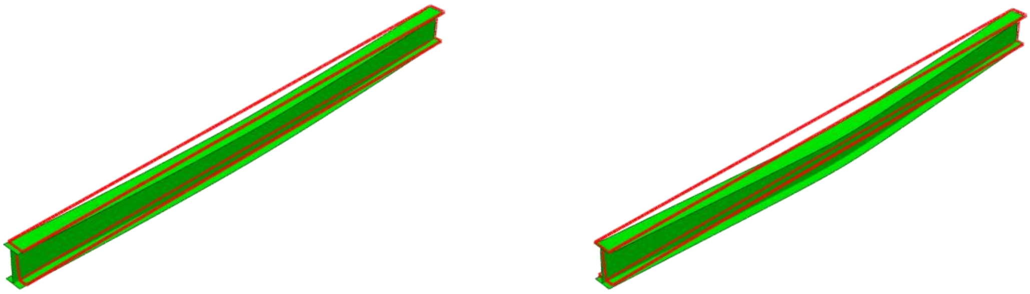

Figure 89 – Steel 3-sided IPE beam with a lightweight floor with an evenly distributed load Q, right uninsulated, left insulated.

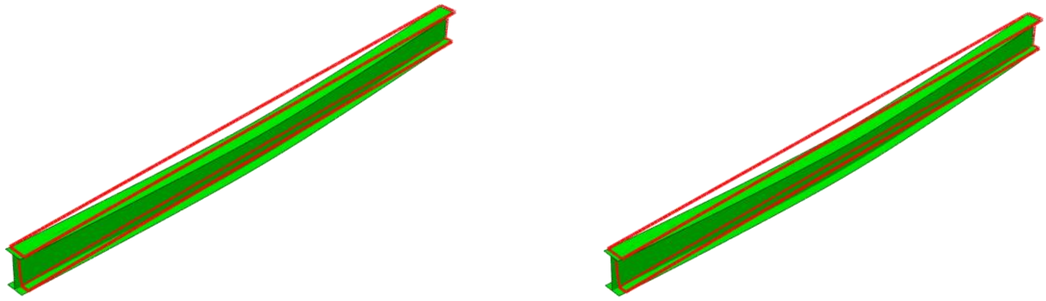

Figure 90 – Steel 3-sided IPE beam with a concrete floor with an evenly distributed load Q, right uninsulated, left insulated.

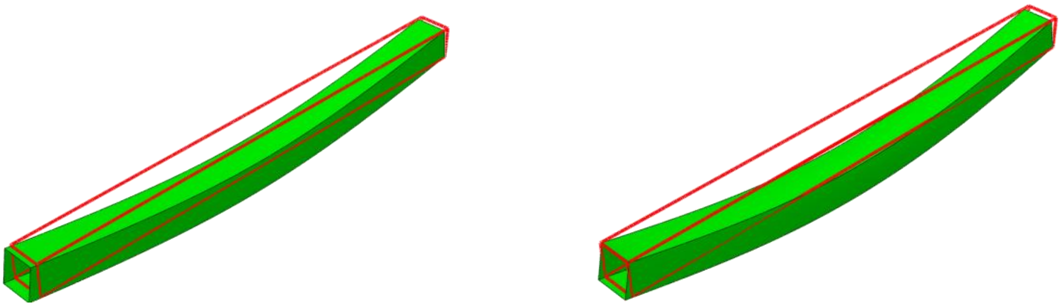

Figure 91 - Aluminium 3-sided beam with lightweight floor, left uninsulated, right insulated with distributed load.

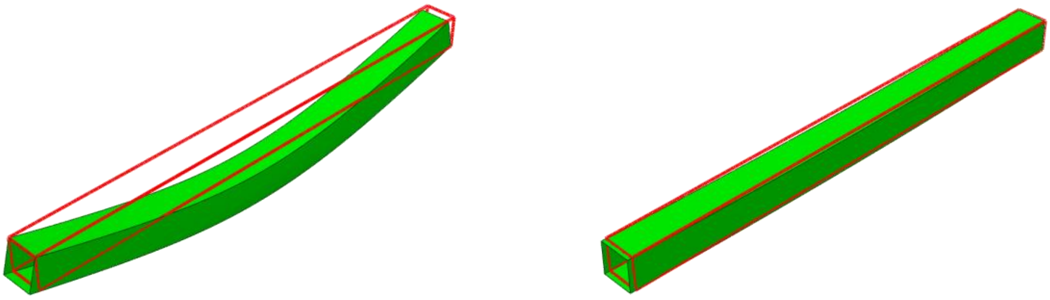

Figure 92 - Aluminium 3-sided beam with concrete floor and evenly distributed load Q, left uninsulated, right insulated.

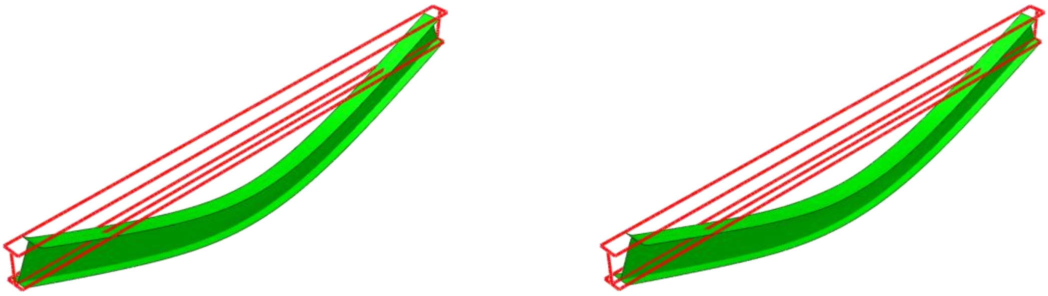

Figure 93 - Aluminium 3-sided beam with lightweight floor, evenly distributed load, left uninsulated, right insulated.

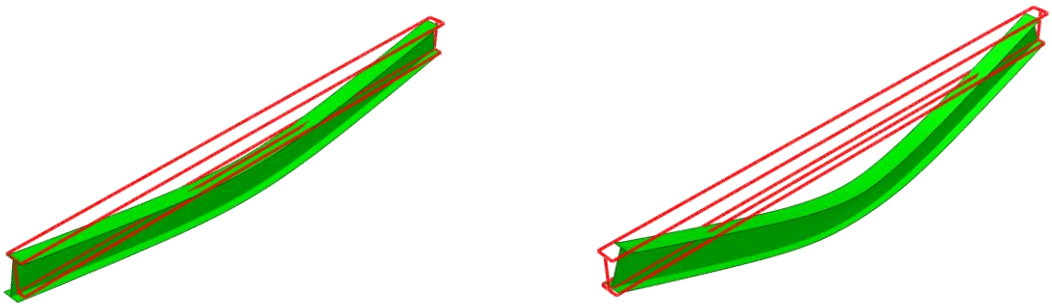

Figure 94 - Aluminium 3-sided beam with concrete floor, evenly distributed load, left uninsulated, right insulated.

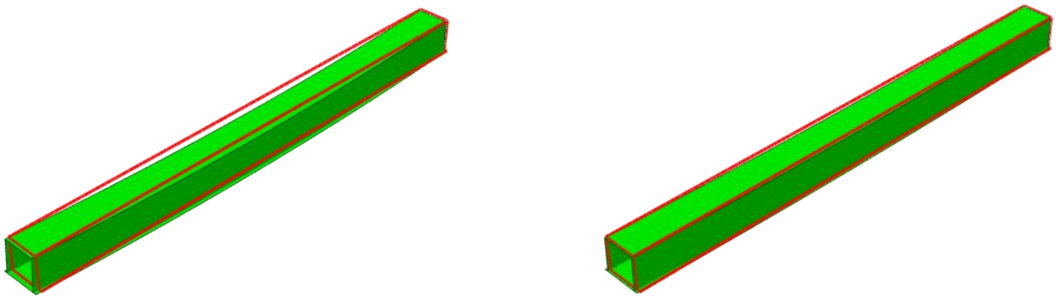

Figure 95 - Steel integrated beam lightweight floor evenly distributed load, left uninsulated, right insulated.

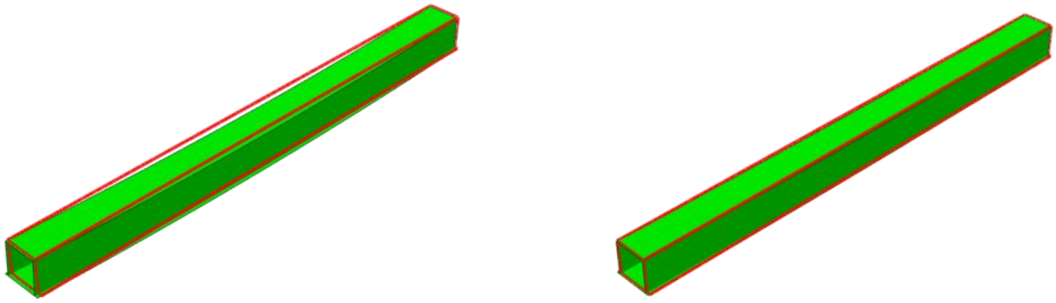

Figure 96 - Steel integrated beam with concrete floor and evenly distributed load, left uninsulated and right insulated.

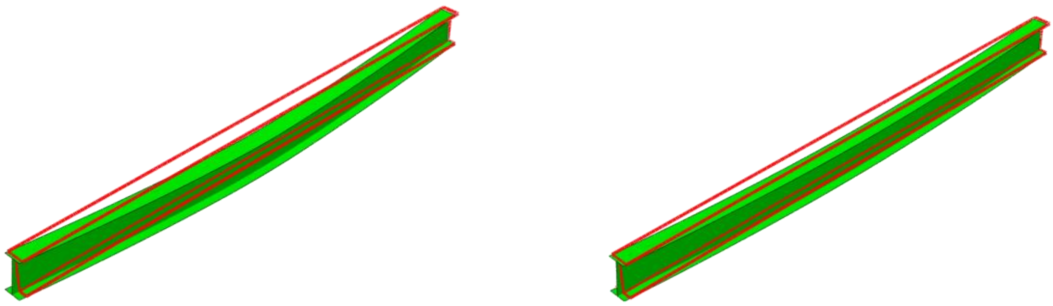

Figure 97 - Steel integrated beam with lightweight floor and evenly distributed load, left uninsulated, right insulated.

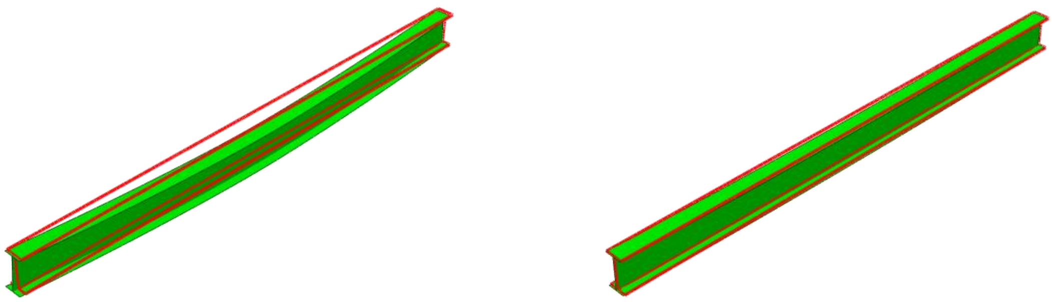

Figure 98 - Steel integrated beam with concrete floor and evenly distributed load, left uninsulated and right insulated.

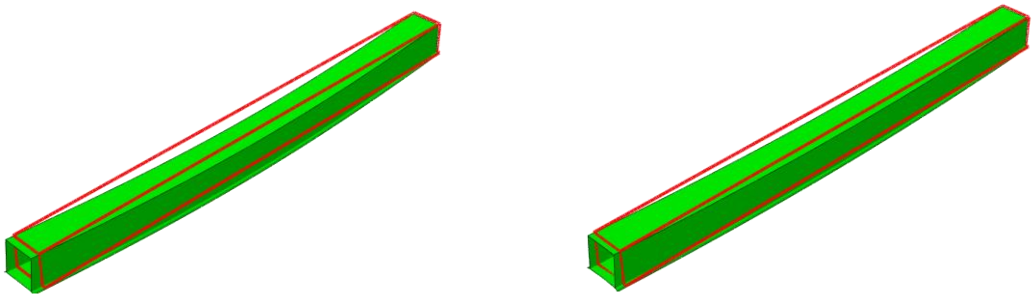

Figure 99 - Aluminium integrated beam with lightweight floor and evenly distributed load, uninsulated left, insulated right.

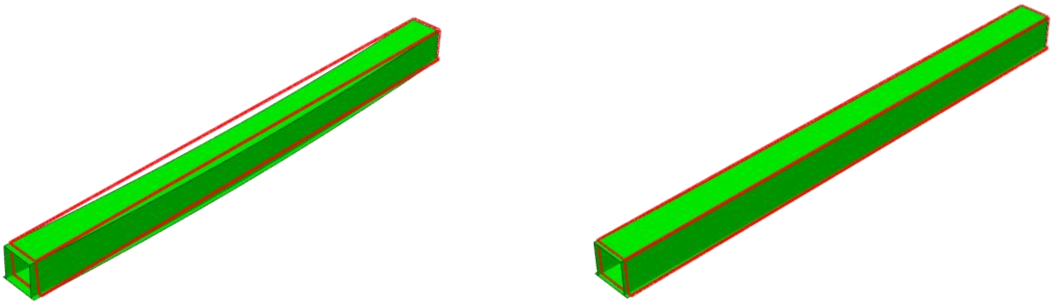

Figure 100 - Aluminium integrated beam with concrete floor and evenly distributed load, left uninsulated, right insulated.

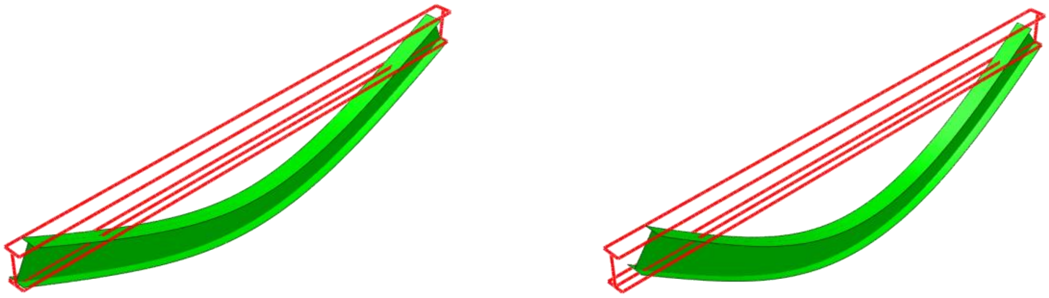

Figure 101 - Aluminium integrated beam with lightweight floor and evenly distributed load, left uninsulated and right insulated.

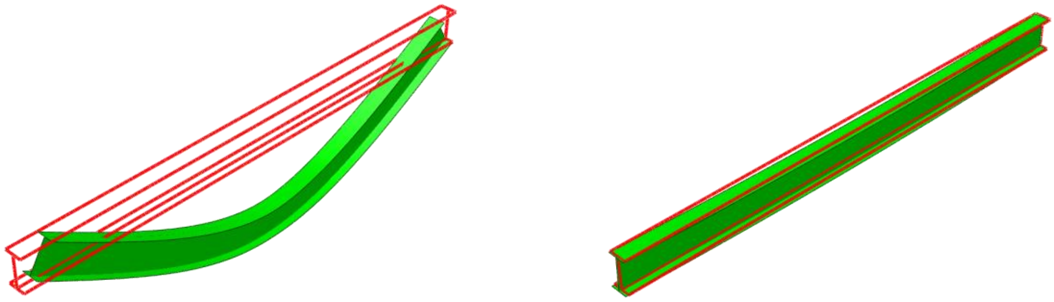

Figure 102 - Aluminium integrated beam with concrete floor and evenly distributed load left uninsulated, right insulated.

D.3 Four point bending test

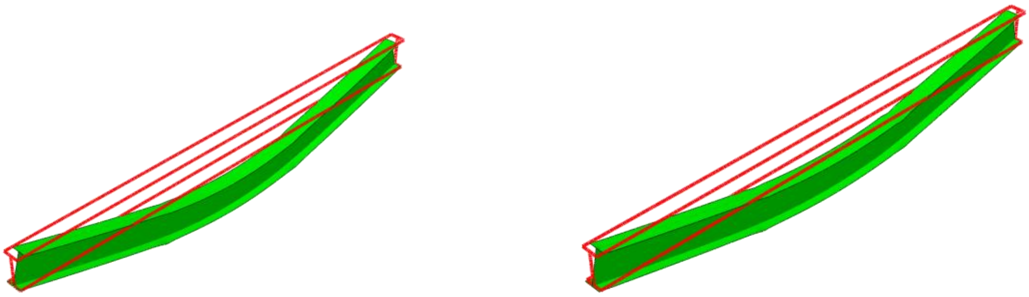

Figure 103 - Aluminium integrated beam concrete floor four point bending test, left uninsulated, right insulated.

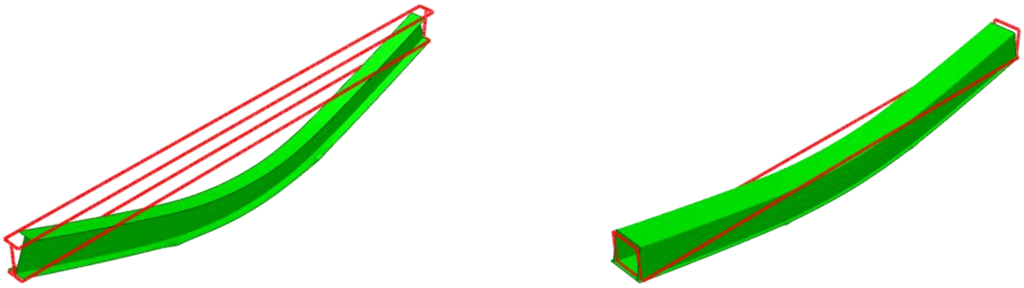

Figure 104 - Aluminium integrated beam uninsulated, lightweight floor on the left, concrete floor on the right.

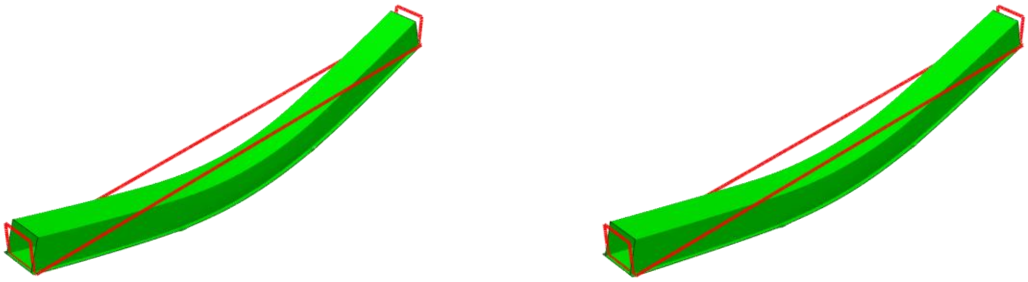

Figure 105 - Aluminium integrated beam with lightweight floor four point bending test, left uninsulated, right insulated.

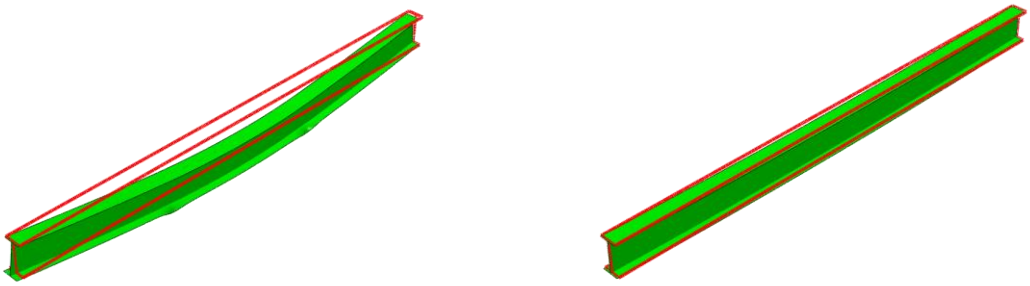

Figure 106 - Steel integrated beam with concrete floor, left uninsulated, right insulated.

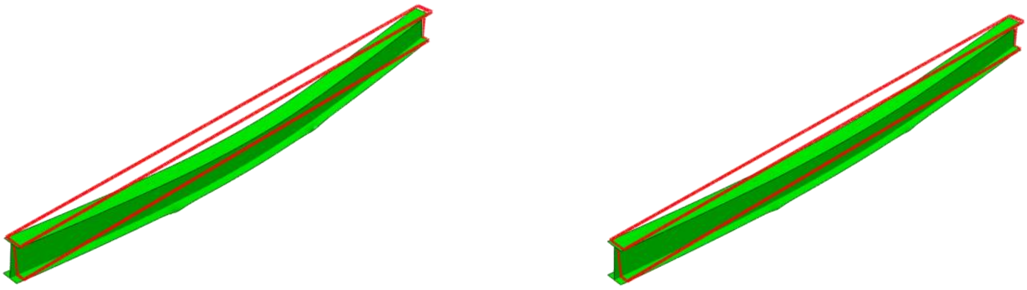

Figure 107 - Steel integrated beam lightweight floor, left uninsulated, right insulated.

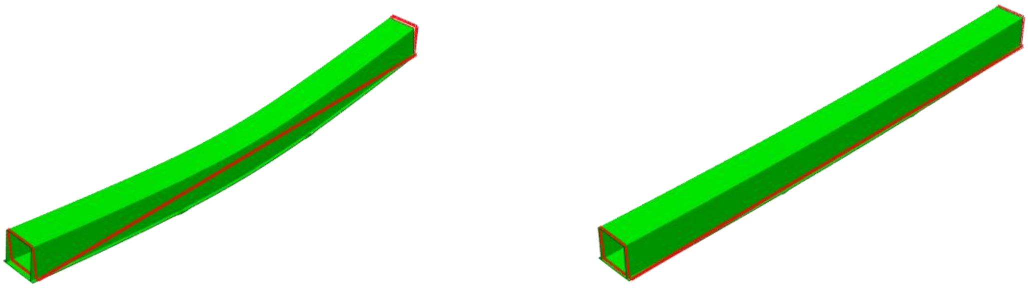

Figure 108 - Steel integrated beam with concrete floor left uninsulated, right insulated.

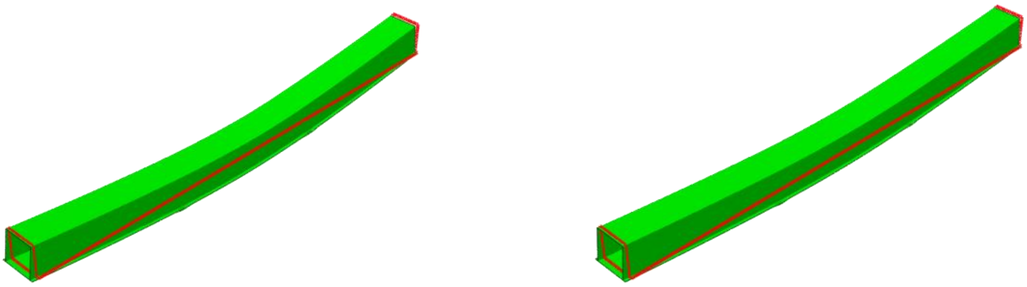

Figure 109 - Steel integrated beam with lightweight floor, left uninsulated, right insulated.

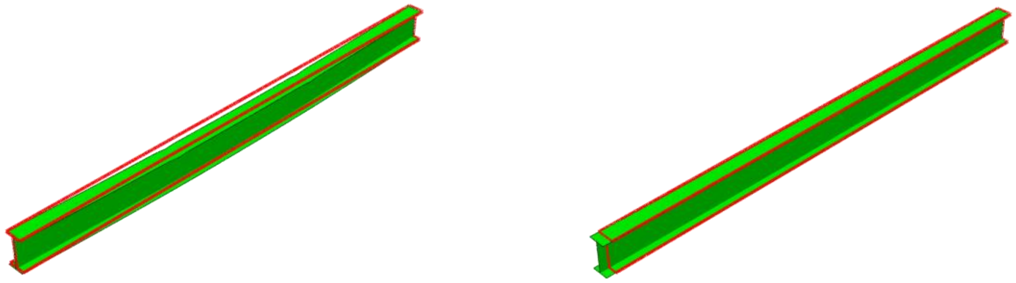

Figure 110 - Aluminium 3-sided beam with concrete floor left uninsulated, right insulated.

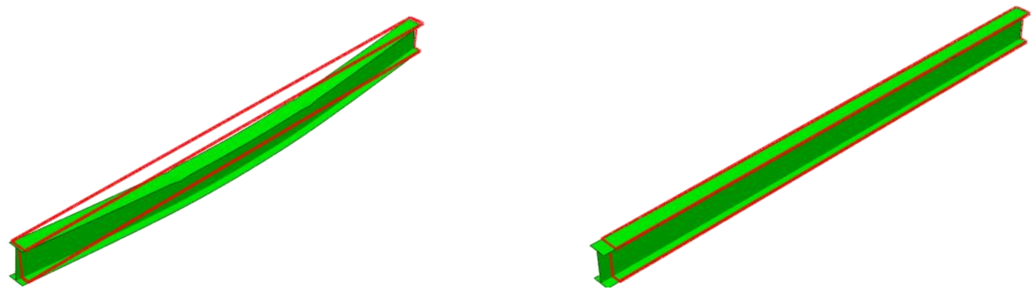

Figure 111 - Aluminium 3-sided beam with lightweight floor, left uninsulated, right insulated.

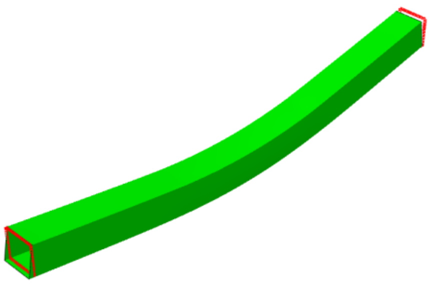

Figure 112 - Aluminium 3-sided beam with concrete floor insulated

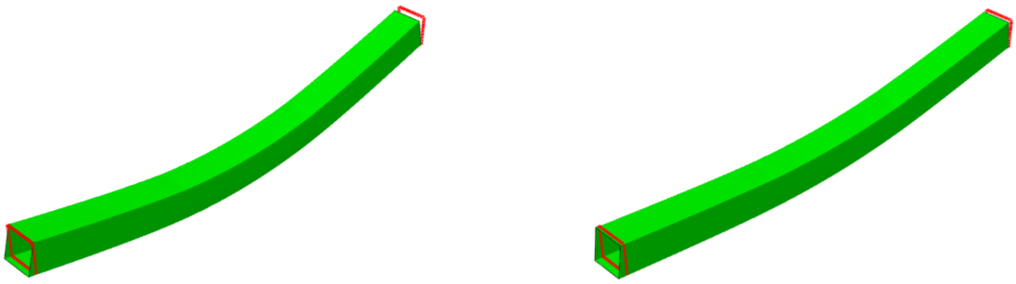

Figure 113 - Aluminium 3-sided beam with lightweight floor, left uninsulated, right insulated.

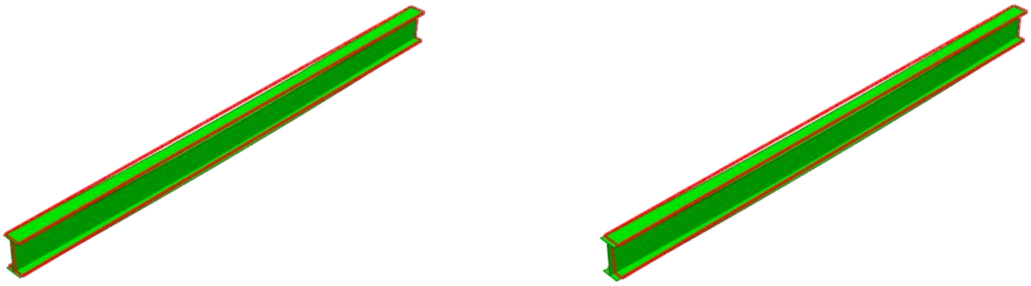

Figure 114 - Steel 3-sided beam with concrete floor, left uninsulated right insulated.

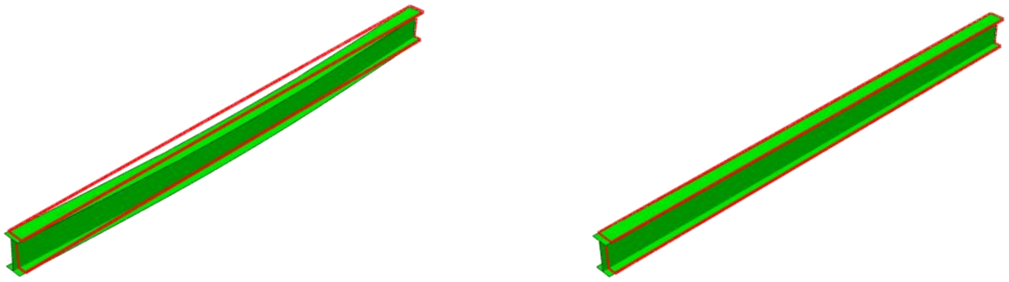

Figure 115 - Steel 3-sided beam with lightweight floor, left uninsulated, right insulated.

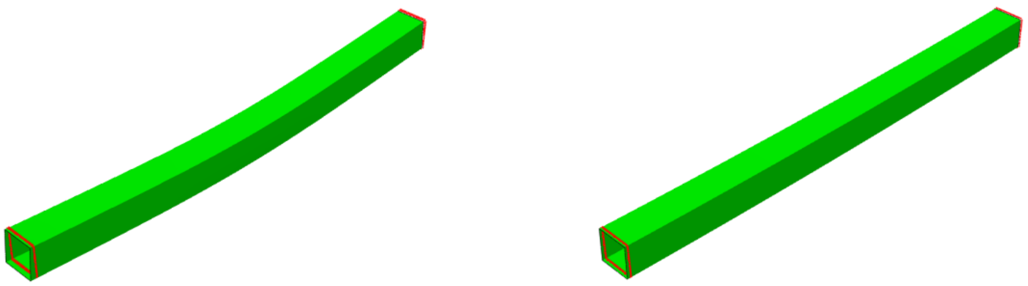

Figure 116 - Steel 3-sided beam with concrete floor, left uninsulated, right insulated.

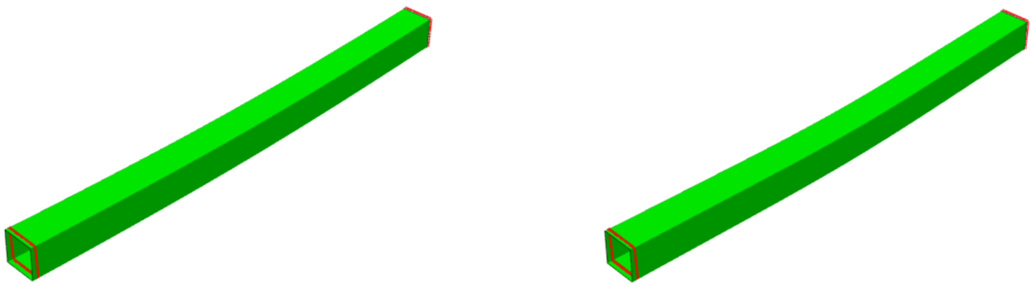

Figure 117 - Steel 3-sided beam with lightweight floor and left uninsulated and right insulated.

E: FEM THERMAL ANALYSIS SCRIPT

```python
#R.M. van der Wurff
#Date

# -*- coding: mbcs -*-
# Abaqus works in true strains and stresses, absolute temperatures in Celsius

#import extensions
from abaqus import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from datetime import *
from odbAccess import *

import os
import csv
# sys.path.append(r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Scripts")
import input_variables
import csv_writer_thermal_analysis
import executable_time

def main(material, insulated, interface, floor, mesh_size):
    # ------------------------------------------------------------------------------------------------------------
    ## model Parameters ##
    analysis         = "Thermal_Analysis_Beam3_IPE_"+material+"_"+floor+'_'+insulated+'_alt'
    cwd              = os.getcwd()
    filelocation     = str(cwd)+"\\"+str(date.today())+"_"+analysis+"\\"
    name_model       = analysis


    session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=COORDINATE)
    Path_Data_Files = r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS"+'\\'+analysis

    #error check on file location
    if not os.path.exists(Path_Data_Files):
        try:
            os.makedirs(Path_Data_Files)
        except OSError as exc:
            if exc.errno != errno.EEXIST:
                raise
    os.chdir(Path_Data_Files)
    Scratch = Path_Data_Files
    myModel_1 = mdb.Model(name = name_model)
    if "Model-1" in mdb.models:
        del mdb.models["Model-1"]
    # -----------------------------------------------------------------------------------------------------
    #Popening input variables
    section=    'I-section'
    Fire_Load = 'Standard_Fire'
    model_values, geometry, Emissivity, Poisons_Alu, contactResistance = input_variables.main(myModel_1, section, material, Fire_Load, insulated, interface)
    T, Step_time, Conv_hot, Conv_ambient = model_values
    H,W,tf,tw,tp, Ws, Hs = geometry
    Emissivity_metal, Emissivity_Ins, Emissivity_Floor = Emissivity
```

```
    # -------------------------------------------------------------------------------------------------
    ## Sketch + Part ##
    # geometry values are inputted in mm
    # Part 1 - Rectangular hollow section #
    mySketch_1 = myModel_1.ConstrainedSketch(name=section, sheetSize=0.2)
    xyCoords = ((-(0.5*W), -tf), (-(0.5*W), 0), (0.5*W, 0), (0.5*W, -tf),
        ((0.5*tw), -tf), ((0.5*tw), -(H-tf)), (0.5*W, -(H-tf)), (0.5*W, -H),
        (-(0.5*W), -H), (-(0.5*W), -(H-tf)), (-(0.5*tw), -(H-tf)), (-(0.5*tw), -
        tf), (-(0.5*W), -tf))
    # Please note: Coordinates have to be such order that section can be drawn fluently
    for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
    myPart_1 = myModel_1.Part(name = section, dimensionality = TWO_D_PLANAR, type=DEFORMABLE_BODY)
    myPart_1.BaseShell(sketch = mySketch_1)
    del mySketch_1

    #Floor slab
    if floor =='Concrete':
        XYcoords = ((-0.5*Ws,Hs),(0.5*Ws,Hs),(0.5*Ws,0),(0.5*W+tp,0),(0.5*W,0),(-0.5*W,0),
            (-0.5*W-tp,0),(-0.5*Ws,0),(-0.5*Ws,Hs))
        mySketch_3 = myModel_1.ConstrainedSketch(name="Slab", sheetSize=0.2)
        for i in range(len(XYcoords)-1): mySketch_3.Line(point1=XYcoords[i], point2=XYcoords[i+1])
        myPart_3 = myModel_1.Part(name='Slab', dimensionality=TWO_D_PLANAR,type=DEFORMABLE_BODY)
        myPart_3.BaseShell(sketch=mySketch_3)

    elif floor =='Lightweight':
        mySketch_3 = myModel_1.ConstrainedSketch(name='Floor_top', sheetSize=0.2)
        XYcoords = ((-0.5*Ws,Hs),(0.5*Ws,Hs),(0.5*Ws,Hs-10),(-0.5*Ws,Hs-10),(-0.5*Ws,Hs))
        for i in range(len(XYcoords)-1): mySketch_3.Line(point1=XYcoords[i], point2=XYcoords[i+1])
        myPart_3 = myModel_1.Part(name='Floor_top', dimensionality=TWO_D_PLANAR,type=DEFORMABLE_BODY)
        myPart_3.BaseShell(sketch=mySketch_3)

        mySketch_4 = myModel_1.ConstrainedSketch(name='Floor_middle', sheetSize=0.2)
        XYcoords = ((-0.5*Ws,Hs-10),(0.5*Ws,Hs-10),(0.5*Ws,1./2*Hs),(-0.5*Ws,1./2*Hs),(-0.5*Ws,Hs-10))
        for i in range(len(XYcoords)-1): mySketch_4.Line(point1=XYcoords[i], point2=XYcoords[i+1])
        myPart_4 = myModel_1.Part(name='Floor_middle', dimensionality=TWO_D_PLANAR,type=DEFORMABLE_BODY)
        myPart_4.BaseShell(sketch=mySketch_4)

        mySketch_5 = myModel_1.ConstrainedSketch(name='Floor_concrete', sheetSize=0.2)
        XYcoords = ((-0.5*Ws,1./2*Hs),(0.5*Ws,1./2*Hs),(0.5*Ws,1./4*Hs),(-0.5*Ws,1./4*Hs),(-0.5*Ws,1./2*Hs))
        for i in range(len(XYcoords)-1): mySketch_5.Line(point1=XYcoords[i], point2=XYcoords[i+1])
        myPart_5 = myModel_1.Part(name='Floor_concrete', dimensionality=TWO_D_PLANAR,type=DEFORMABLE_BODY)
        myPart_5.BaseShell(sketch=mySketch_5)

        mySketch_6 = myModel_1.ConstrainedSketch(name='Floor_ins', sheetSize=0.2)
        XYcoords = ((-0.5*Ws,1./4*Hs),(0.5*Ws,1./4*Hs),(0.5*Ws,0),(0.5*W+tp,0),(0.5*W,0),(-0.5*W,0),(-0.5*W-tp,0),(-0.5*Ws,0),(-0.5*Ws,1./4*Hs))
        for i in range(len(XYcoords)-1): mySketch_6.Line(point1=XYcoords[i], point2=XYcoords[i+1])
        myPart_6 = myModel_1.Part(name='Floor_ins', dimensionality=TWO_D_PLANAR,type=DEFORMABLE_BODY)
```

```
        myPart_6.BaseShell(sketch=mySketch_6)

    # Part 2 - Insulation #
    if insulated=='yes':
        mySketch_2 = myModel_1.ConstrainedSketch(name='Insulation', sheetSize = 0.2)
        xyCoords_out_Ins = ((-(0.5*W+tp), 0), (-(0.5*W), 0),((-0.5*W,-tf)),(-0.5*tw,-tf),(-0.5*tw,-(H-tf)),(-0.5*W,-H+tf),
            (-0.5*W,-H),(0.5*W,-H),(0.5*W,-H+tf),(0.5*tw,-H+tf),(0.5*tw,-tf),(0.5*W,-tf),(0.5*W,0),(0.5*W+tp,0),
            ((0.5*W+tp), -tf-tp), ((0.5*tw+tp), -tf-tp),((0.5*tw+tp), -H+tf+tp), ((0.5*W+tp), (-H+tp+tf)), (0.5*W+tp, (-H-tp)), (-(0.5*W+tp), -H-tp),
            (-(0.5*W+tp), -H+tf+tp), (-(0.5*tw+tp), -H+tf+tp), (-(0.5*tw+tp), -(tf+tp)), (-(0.5*W+tp), -tf-tp), (-(0.5*W+tp), 0))
        # Please note: Coordinates have to be such order that section can be drawn fluently
        for i in range (len(xyCoords_out_Ins)-1): mySketch_2.Line(point1 = xyCoords_out_Ins [i],point2 = xyCoords_out_Ins [i+1])
        myPart_2 = myModel_1.Part(name='insulation', dimensionality = TWO_D_PLANAR, type=DEFORMABLE_BODY)
        myPart_2.BaseShell(sketch = mySketch_2)
        del mySketch_2

    # -----------------------------------------------------------------------------------------------
    ## Sets ##
    #all sets based on geometry are copied into assembly
    Outer_edge_IPE  = myPart_1.edges.findAt(((-(0.5*W), -tf*0.5,0),), ((0.5*W, -0.5*tf,0),), ((tw, -tf,0),),
        (((0.5*tw), -0.5*H,0),), (((tw), -(H-tf),0),), ((0.5*W, -(H-0.5*tf),0),), ((0, -H,0),),
        ((-(0.5*W), -H+0.5*tw,0),), ((-(tw), -(H-tf),0),), ((-(0.5*tw), -(0.5*H),0),), ((-(tw), -tf,0),),)
    IPE_Floor_edge = myPart_1.edges.findAt(((0,0,0),))
    Surface_IPE     = myPart_1.faces.findAt(((0,-0.5*tf,0),))

    mySet_11 = myPart_1.Set(edges=Outer_edge_IPE, name='Outside_IPE') #contact edge
    mySet_13 = myPart_1.Set(name='IPE',faces = Surface_IPE)
    mySurface_11 = myPart_1.Surface(name='Outside_IPE', side1Edges =Outer_edge_IPE) #contact surface
    mySurface_12 = myPart_1.Surface(name='IPE_Floor', side1Edges=IPE_Floor_edge)

    Floor_fire = myPart_6.edges.findAt(((-0.5*Ws+1,0,0),),((0.5*Ws-1,0,0),),)
    Floor_top = myPart_3.edges.findAt(((0,Hs,0),),)
    Floor_ins = myPart_6.edges.findAt(((-0.5*W-tp+1,0,0),),((0.5*W+1,0,0),),)
    Floor_IPE = myPart_6.edges.findAt(((0,0,0),),)
    mySet_31 = myPart_3.Set(name='Floor_top', edges = Floor_top)
    mySet_32 = myPart_6.Set(name='Floor_fire', edges = Floor_fire)
    mySurface_31 = myPart_6.Surface(name='Floor_fire', side1Edges=Floor_fire)
    mySurface_32 = myPart_6.Surface(name='Floor_ins', side1Edges=Floor_ins)
    mySurface_33 = myPart_6.Surface(name='Floor_IPE', side1Edges=Floor_IPE)
    mySurface_34 = myPart_3.Surface(name='Floor_top', side1Edges=Floor_top)

    Floor_body_top = myPart_3.faces.findAt(((0,Hs-1,0),),)
    Floor_body_middle = myPart_4.faces.findAt(((0,1./2*Hs+1,0),) ,)
    Floor_body_concrete = myPart_5.faces.findAt(((0,1./4*Hs+1,0),),)
    Floor_body_ins = myPart_6.faces.findAt(((0,1,0),),)
    mySet_33a = myPart_3.Set(name='Floor_body_top', faces = Floor_body_top)
    mySet_33b = myPart_4.Set(name='Floor_body_middle', faces = Floor_body_middle)
    mySet_33c = myPart_5.Set(name='Floor_body_concrete', faces = Floor_body_concrete)
    mySet_33d = myPart_6.Set(name='Floor_body_ins', faces = Floor_body_ins)

    if floor =='Lightweight':
        Floor_top       = myPart_3.faces.findAt( ((0,Hs-1,0),),)
        Floor_ins       = myPart_6.faces.findAt( ((0,1,0),),)
        Floor_middle    = myPart_4.faces.findAt( ((0,1./2*Hs+1,0),),)
        Floor_concrete  = myPart_5.faces.findAt( ((0,1./4*Hs+1,0),),)
        Floor1a = myPart_3.edges.findAt( ((-0.5*Ws,Hs-10,0),),)
        Floor2a = myPart_4.edges.findAt( ((-0.5*Ws,1./2*Hs,0),),)
        Floor3a = myPart_5.edges.findAt( ((-0.5*Ws,1./4*Hs,0),),)
        Floor1b = myPart_4.edges.findAt( ((-0.5*Ws,Hs-10,0),),)
```

```
        Floor2b = myPart_5.edges.findAt( ((-0.5*Ws,1./2*Hs,0),),)
        Floor3b = myPart_6.edges.findAt( ((-0.5*Ws,1./4*Hs,0),),)

        myFloor_top = myPart_3.Set(name='Floor_top', faces=Floor_top)
        myFloor_ins = myPart_6.Set(name='Floor_ins', faces = Floor_ins)
        myFloor_middle = myPart_4.Set(name='Floor_middle', faces = Floor_middle)
        myFloor_concrete = myPart_5.Set(name='Floor_concrete', faces = Floor_concrete)
        mySurface_Floor_1a = myPart_3.Surface(name='Floor1a', side1Edges=Floor1a)
        mySurface_Floor_2a = myPart_4.Surface(name='Floor2a', side1Edges=Floor2a)
        mySurface_Floor_3a = myPart_5.Surface(name='Floor3a', side1Edges=Floor3a)
        mySurface_Floor_1b = myPart_4.Surface(name='Floor1b', side1Edges=Floor1b)
        mySurface_Floor_2b = myPart_5.Surface(name='Floor2b', side1Edges=Floor2b)
        mySurface_Floor_3b = myPart_6.Surface(name='Floor3b', side1Edges=Floor3b)

    if insulated=='yes':
        Fire_Ins     = myPart_2.edges.findAt((((0.5*W+tp), -tp,0),), (((tw+tp), -tf-tp,0),), (((0.5*tw+tp), -0.5*H,0),),
            (((tw+tp), -H+tf+tp,0),), (((0.5*W+tp), (-H),0),), ((0, (-H-tp),0),), ((-(0.5*W+tp), -H,0),),
            ((-(tw+tp), -H+tf+tp,0),), ((-(0.5*tw+tp), -0.5*H,0),), ((-(tw+tp), -(tf+tp),0),), ((-(0.5*W+tp), -tp,0),),)
        IPE_Ins = myPart_2.edges.findAt(((-(0.5*W), -tf*0.5,0),), ((0.5*W, -0.5*tf,0),), ((tw, -tf,0),),
        (((0.5*tw), -0.5*H,0),), (((tw), -(H-tf),0),), ((0.5*W, -(H-0.5*tf),0),), ((0, -H,0),),
        ((-(0.5*W), -H+0.5*tw,0),), ((-(tw), -(H-tf),0),), ((-(0.5*tw), -(0.5*H),0),), ((-(tw), -tf,0),),)
        Surface_Ins = myPart_2.faces.findAt(((-0.5*W-tp+1,-1,0),),)
        Ins_Floor = myPart_2.edges.findAt(((-0.5*W-tp+1,0,0),),((0.5*W+1,0,0),),)

        mySet_21 = myPart_2.Set(name='Outside_Ins', edges=Fire_Ins)
        mySet_22 = myPart_2.Set(name='Inside_Ins', edges=IPE_Ins) #contact edge
        mySet_23 = myPart_2.Set(name='Blanket_1', faces=Surface_Ins)
        mySurface_21 = myPart_2.Surface(name='Outside_Ins', side1Edges=Fire_Ins) #fire side
        mySurface_22 = myPart_2.Surface(name='Inside_Ins', side1Edges=IPE_Ins) #contact surface
        mySurface_23 = myPart_2.Surface(name='Ins_Floor', side1Edges=Ins_Floor)

    # -----------------------------------------------------------------------------------------
    ## Section ##
    myModel_1.HomogeneousSolidSection(material=material, name='IPE',thickness= None)
    myModel_1.HomogeneousSolidSection(material='Insulation', name='Blanket',thickness=None)
    myModel_1.HomogeneousSolidSection(material=floor, name='Slab', thickness=None)
    myModel_1.HomogeneousSolidSection(material='air-alu', name='air-alu', thickness=None)
    # ---------------------------------------------------------------------------------------------------
    ## Section Assignment ##
    myPart_1.SectionAssignment(offset = 0.0, offsetField = " ", offsetType = MIDDLE_SURFACE,
        region = myPart_1.sets['IPE'], sectionName = "IPE", thicknessAssignment = FROM_SECTION)
    if insulated=='yes':
        myPart_2.SectionAssignment(offset = 0.0, offsetField = " ", offsetType = MIDDLE_SURFACE,
        region = myPart_2.sets['Blanket_1'], sectionName = "Blanket", thicknessAssignment = FROM_SECTION)
    if floor =='Concrete':
        myPart_3.SectionAssignment(offset = 0.0, offsetField = " ", offsetType = MIDDLE_SURFACE,
            region = myPart_3.sets['Floor_body'], sectionName = "Slab", thicknessAssignment= FROM_SECTION)
    elif floor=='Lightweight':
        myPart_3.SectionAssignment(offset=0.0, offsetField="", offsetType=MIDDLE_SURFACE,
            region = myPart_3.sets['Floor_top'], sectionName='IPE', thicknessAssignment=FROM_SECTION)
```

```
    myPart_4.SectionAssignment(offset=0.0, offsetField="", offsetType=MIDDLE_SURFACE,
        region = myPart_4.sets['Floor_middle'], sectionName='air-alu', thicknessAssignment=FROM_SECTION)
    myPart_6.SectionAssignment(offset=0.0, offsetField="", offsetType=MIDDLE_SURFACE,
        region = myPart_6.sets['Floor_ins'], sectionName='Blanket', thicknessAssignment=FROM_SECTION)
    myPart_5.SectionAssignment(offset=0.0, offsetField="", offsetType=MIDDLE_SURFACE,
        region = myPart_5.sets['Floor_concrete'], sectionName='Slab', thicknessAssignment=FROM_SECTION)

# ------------------------------------------------------------------------------------------------------------------
## Step ##
myModel_1.HeatTransferStep (timePeriod = T, deltmx = 50, initialInc = 5, maxInc = T,
    maxNumInc = 10000, minInc = 0.001, name = "Heat Transfer", previous = "Initial", response = TRANSIENT)

# ------------------------------------------------------------------------------------------------------------------
## Mesh ##
# Mesh IPE #
myPart_1.setMeshControls(algorithm=MEDIAL_AXIS, minTransition =ON,
    technique = FREE, regions = Surface_IPE)
myPart_1.setElementType (regions = mySet_13, elemTypes = (ElemType( elemCode =
    DC2D4,elemLibrary = STANDARD), )) #2D linear heat transfer blocks, 4 nodes per element
myPart_1.seedPart (deviationFactor = 1, minSizeFactor = 1, size = mesh_size)
myPart_1.generateMesh()

if insulated=='yes':
    # Mesh Insulation #
    myPart_2.setMeshControls(algorithm=MEDIAL_AXIS, minTransition =ON,
        technique = FREE, regions = Surface_Ins)
    myPart_2.setElementType (regions = mySet_23, elemTypes = (ElemType( elemCode =DC2D8,
        elemLibrary = STANDARD), )) #2D quadratic heat transfer blocks, 8 nodes per element
    myPart_2.seedPart (deviationFactor = 1, minSizeFactor = 1, size = tp/4)
    myPart_2.generateMesh()

#Mesh Slab
myPart_3.setMeshControls(algorithm = MEDIAL_AXIS, minTransition=ON,
    technique = FREE, regions = Floor_body_top)
myPart_3.setElementType(regions = mySet_33a, elemTypes=(ElemType(elemCode =
    DC2D8, elemLibrary = STANDARD),))
myPart_3.seedPart (deviationFactor=1, minSizeFactor =1 , size = tp/4)
myPart_3.generateMesh()
myPart_4.setMeshControls(algorithm = MEDIAL_AXIS, minTransition=ON,
    technique = FREE, regions = Floor_body_middle)
myPart_4.setElementType(regions = mySet_33b, elemTypes=(ElemType(elemCode =
    DC2D8, elemLibrary = STANDARD),))
myPart_4.seedPart (deviationFactor=1, minSizeFactor =1 , size = tp/4)
myPart_4.generateMesh()
myPart_5.setMeshControls(algorithm = MEDIAL_AXIS, minTransition=ON,
    technique = FREE, regions = Floor_body_concrete)
myPart_5.setElementType(regions = mySet_33c, elemTypes=(ElemType(elemCode =
    DC2D8, elemLibrary = STANDARD),))
myPart_5.seedPart (deviationFactor=1, minSizeFactor =1 , size = tp/4)
myPart_5.generateMesh()
myPart_6.setMeshControls(algorithm = MEDIAL_AXIS, minTransition=ON,
    technique = FREE, regions = Floor_body_ins)
myPart_6.setElementType(regions = mySet_33d, elemTypes=(ElemType(elemCode =
    DC2D8, elemLibrary = STANDARD),))
myPart_6.seedPart (deviationFactor=1, minSizeFactor =1 , size = tp/4)
myPart_6.generateMesh()
#
```

```
------------------------------------------------------------------------------------------------------------
## Assembly ##
myAssembly = myModel_1.rootAssembly
myAssembly.DatumCsysByDefault (CARTESIAN)
myAssembly.Instance(dependent = ON, part = myPart_1, name = "IPE-1")
myAssembly.Instance(dependent = ON, part = myPart_3, name = "Slab_top")
myAssembly.Instance(dependent = ON, part = myPart_4, name = "Slab_middle")
myAssembly.Instance(dependent = ON, part = myPart_5, name = "Slab_concrete")
myAssembly.Instance(dependent = ON, part = myPart_6, name = "Slab_ins")
if insulated=='yes':
    myAssembly.Instance(dependent = ON, part = myPart_2, name = "Blanket_1")
#all previously made sets are copied into assembly, only applicable to geometry dependent sets

#--------------------------------------------------------------------------------------------------------------
## Fire Loads ##
if insulated=='yes':
    region = myAssembly.instances['Blanket_1'].surfaces['Outside_Ins']
    Emissivity = Emissivity_Ins
else:
    region = myAssembly.instances['IPE-1'].surfaces['Outside_IPE']
    Emissivity = Emissivity_metal

if Fire_Load == 'Standard_Fire':
    # Convection Fire Side #
    myModel_1.FilmCondition(createStepName = 'Heat Transfer', definition = EMBEDDED_COEFF,
        filmCoeff = Conv_hot, name = 'Convection_Fire_Side', sinkDistributionType = UNIFORM,
        sinkTemperature = 1, sinkAmplitude = "Standard Fire", surface = region)
    myModel_1.FilmCondition(createStepName = 'Heat Transfer', definition = EMBEDDED_COEFF,
        filmCoeff = Conv_hot, name = 'Convection_Fire_Side_Floor', sinkDistributionType = UNIFORM,
        sinkTemperature = 1, sinkAmplitude = "Standard Fire", surface = myAssembly.instances['Slab_ins'].surfaces['Floor_fire'])

    # Radiation Fire Side #
    myModel_1.RadiationToAmbient (ambientTemperature = 1, ambientTemperatureAmp = 'Standard Fire',
        createStepName = 'Heat Transfer', emissivity = Emissivity_Ins, name = 'Radiation_Fire_Side',
        distributionType = UNIFORM, surface=region)
    myModel_1.RadiationToAmbient (ambientTemperature =1, ambientTemperatureAmp = 'Standard Fire',
        createStepName = 'Heat Transfer', emissivity = Emissivity_Floor, name = 'Radiation_Fire_Side_Floor',
        distributionType = UNIFORM, surface = myAssembly.instances['Slab_ins'].surfaces['Floor_fire'])

    # Ambient side
    myModel_1.FilmCondition (createStepName = 'Heat Transfer', definition = EMBEDDED_COEFF,
        filmCoeff = Conv_ambient, name = 'Convection_Ambient_Side', sinkDistributionType = UNIFORM,
        sinkTemperature = 20, surface = myAssembly.instances['Slab_top'].surfaces['Floor_top'])
    myModel_1.RadiationToAmbient (ambientTemperature = 20, createStepName = 'Heat Transfer',
        emissivity = Emissivity_Floor, name = 'Radiation_Ambient_Side',
        distributionType = UNIFORM,surface = myAssembly.instances['Slab_top'].surfaces['Floor_top'])

if Fire_Load == 'Hydrocarbon':
    myModel_1.EdgeHeatFlux(name = 'heatflux on insulation', createStepName =
        'Heat Transfer', region = region, magnitude = myModel_1.TabularAmplitude['Hydrocarbon'])
    myModel_1.EdgeHeatFlux(name = 'heatflux on Floor', createStepName = 'Heat Transfer',
```

```
            region = myAssembly.instances['Slab_ins'].surfaces['Floor_fire'],
            magnitude = myModel_1.TabularAmplitude['Hydrocarbon'])

    if insulated=='yes':
        # Contact Resistance insulation - RHS #
        myModel_1.ContactProperty ('Contact_Resistance_IPE_ins')

        myModel_1.interactionProperties['Contact_Resistance_IPE_ins'].ThermalConductance(
            clearanceDepTable =((contactResistance, 0), (0, 1)), clearanceDependency = ON, definition = TABULAR)
        myModel_1.SurfaceToSurfaceContactStd (name = 'Contact_Resistance', master =myAssembly.instances['IPE-1'].surfaces['Outside_IPE'],
            slave = myAssembly.instances['Blanket_1'].surfaces['Inside_Ins'], createStepName = 'Heat Transfer', interactionProperty ='Contact_Resistance_IPE_ins',
            sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)
        # Contact Resistance insulation - Floor #
        myModel_1.ContactProperty ('Contact_Resistance_Slab_Ins')

        myModel_1.interactionProperties['Contact_Resistance_Slab_Ins'].ThermalConductance(
            clearanceDepTable =((contactResistance, 0), (0, 1)), clearanceDependency = ON, definition = TABULAR)
        myModel_1.SurfaceToSurfaceContactStd (name = 'Contact_Resistance_Ins_Floor', master =myAssembly.instances['Slab_ins'].surfaces['Floor_ins'],
            slave = myAssembly.instances['Blanket_1'].surfaces['Ins_Floor'], createStepName = 'Heat Transfer', interactionProperty ='Contact_Resistance_Slab_Ins',
            sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)

    # Contact Floor - RHS #
    myModel_1.ContactProperty ('Contact_Resistance_Floor_IPE')

    myModel_1.interactionProperties['Contact_Resistance_Floor_IPE'].ThermalConductance(
        clearanceDepTable =((200e-3, 0), (0, 1)), clearanceDependency = ON, definition = TABULAR)
    myModel_1.SurfaceToSurfaceContactStd (name = 'Contact_Resistance_Floor_IPE', master =myAssembly.instances['IPE-1'].surfaces['IPE_Floor'],
        slave = myAssembly.instances['Slab_ins'].surfaces['Floor_IPE'], createStepName = 'Heat Transfer', interactionProperty ='Contact_Resistance_Floor_IPE',
        sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)

    # Contact in floor
    myModel_1.ContactProperty('Contact_Floor_top')
    myModel_1.interactionProperties['Contact_Floor_top'].ThermalConductance(
        clearanceDepTable=((0,0),(0,1)), clearanceDependency=ON, definition=TABULAR)
    myModel_1.SurfaceToSurfaceContactStd(name='Contact_Floor_top', master = myAssembly.instances['Slab_top'].surfaces['Floor1a'],
        slave=myAssembly.instances['Slab_middle'].surfaces['Floor1b'], createStepName='Heat Transfer', interactionProperty= 'Contact_Floor_top',
        sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)
    myModel_1.ContactProperty('Contact_Floor_middle')
    myModel_1.interactionProperties['Contact_Floor_middle'].ThermalConductance(
        clearanceDepTable=((0,0),(0,1)), clearanceDependency=ON, definition=TABULAR)
    myModel_1.SurfaceToSurfaceContactStd(name='Contact_Floor_middle', master = myAssembly.instances['Slab_middle'].surfaces['Floor2a'],
        slave=myAssembly.instances['Slab_concrete'].surfaces['Floor2b'], createStepName='Heat Transfer', interactionProperty= 'Contact_Floor_middle',
        sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)
    myModel_1.ContactProperty('Contact_Floor_bottom')
    myModel_1.interactionProperties['Contact_Floor_bottom'].ThermalConductance(
        clearanceDepTable=((0,0),(0,1)), clearanceDependency=ON, definition=TABULAR)
    myModel_1.SurfaceToSurfaceContactStd(name='Contact_Floor_bottom', master = myAssembly.instances['Slab_concrete'].surfaces['Floor3a'],
        slave=myAssembly.instances['Slab_ins'].surfaces['Floor3b'], createStepName='Heat Transfer', interactionProperty= 'Contact_Floor_bottom',
        sliding=FINITE, surfaceSmoothing=NONE, thickness=ON)

    # fire if not insulated
```

```
    if insulated!='yes':
        myModel_1.FilmCondition(createStepName = 'Heat Transfer', definition = EMBEDDED_COEFF,
            filmCoeff = Conv_hot, name = 'Convection_Fire_Floor',
            sinkDistributionType = UNIFORM,
            sinkTemperature = 1, sinkAmplitude = "Standard Fire", surface = myAssembly.instances['Slab_ins'].surfaces['Floor_ins'])
        myModel_1.RadiationToAmbient (ambientTemperature = 1, ambientTemperatureAmp = 'Standard Fire',
            createStepName = 'Heat Transfer', emissivity = Emissivity_Floor, name = 'Radiation_Fire_Floor',
            distributionType = UNIFORM,
            surface=myAssembly.instances['Slab_ins'].surfaces['Floor_ins'])

    # -------------------------------------------------------------------------------------------------------------
    ## BCs ##
    # Predifined field - constant initial temperature of 20 C #
    myModel_1.Temperature (createStepName = "Initial", crossSectionDistribution =
        CONSTANT_THROUGH_THICKNESS, distributionType = UNIFORM, magnitudes =(20, ), name = "Initial TemperatureIPE",
        region = myAssembly.instances["IPE-1"].sets["IPE"])
    myModel_1.Temperature ( createStepName="Initial", crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
        distributionType = UNIFORM, magnitudes=(20,), name = "Initial Temperature Floor1",
        region = myAssembly.instances["Slab_top"].sets['Floor_body_top'])
    myModel_1.Temperature ( createStepName="Initial", crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
        distributionType = UNIFORM, magnitudes=(20,), name = "Initial Temperature Floor2",
        region = myAssembly.instances["Slab_middle"].sets['Floor_body_middle'])
    myModel_1.Temperature ( createStepName="Initial", crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
        distributionType = UNIFORM, magnitudes=(20,), name = "Initial Temperature Floor3",
        region = myAssembly.instances["Slab_concrete"].sets['Floor_body_concrete'])
    myModel_1.Temperature ( createStepName="Initial", crossSectionDistribution=CONSTANT_THROUGH_THICKNESS,
        distributionType = UNIFORM, magnitudes=(20,), name = "Initial Temperature Floor4",
        region = myAssembly.instances["Slab_ins"].sets['Floor_body_ins'])
    if insulated=='yes':
        myModel_1.Temperature (createStepName = "Initial", crossSectionDistribution =
            CONSTANT_THROUGH_THICKNESS,
            distributionType = UNIFORM, magnitudes =(20, ), name = "Initial Temperature insulation",
            region = myAssembly.instances["Blanket_1"].sets["Blanket_1"])

    # -----------------------------------------------------------------------------------------------------------
    ## Output Request ##
    myModel_1.fieldOutputRequests['F-Output-1'].setValues(variables = ('NT','COORD'),
        frequency = 1, region = myAssembly.instances['IPE-1'].sets['IPE'])
    myModel_1.FieldOutputRequest (name = 'Temperature_XY_Output_Surface', createStepName =
        'Heat Transfer', timeInterval = Step_time, variables = ('COORD', 'NT'), region =
        myAssembly.instances['IPE-1'].sets['IPE'])

    # -----------------------------------------------------------------------------------------------------------
    ## Job ##
    myJob_1 = mdb.Job(name = name_model, model = myModel_1, type = ANALYSIS,scratch = Scratch)
    myJob_1.submit(consistencyChecking=OFF)
    myJob_1.waitForCompletion()

    odb = session.openOdb(name = name_model+'.odb')
```

```
    frames = odb.steps['Heat Transfer'].frames
    numFrames = int(len(frames))
    # mySurface_odb = odb.rootAssembly.instances['IPE-1'].nodeSets['OUTSIDE_IPE']
    csv_writer_thermal_analysis.csv_coordinates(odb, name_model)
    csv_writer_thermal_analysis.csv_temperatures(odb, name_model, numFrames)
    csv_writer_thermal_analysis.csv_thermal_result(name_model)
    #odb.close()
    executable_time.ExecTime(name_model)
    return numFrames
```

F: FEM MECHANICAL ANALYSIS SCRIPT

```python
#Mechanical test for abaqus run

#import extensions
from abaqus import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from datetime import *
from odbAccess import *

import os
import csv
sys.path.append(r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Scripts")
import input_variables, csv_writer_thermal_analysis

def main(type, material, elements, section, floor, insulated, interface, numFrames, mesh_size):
    analysis        = 'Mech_' + type + '_' + material + '_' + elements + '_' + section + '_' + floor + '_' + insulated + '_'+str(mesh_size)
    cwd             = os.getcwd()
    filelocation    = str(cwd)+"\\"+str(date.today())+"_"+analysis+"\\"
    myModel_1       = mdb.Model(name=analysis)
    session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=COORDINATE)
    Path_Data_Files = r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS"+'\\'+analysis

    #error check on file location
    if not os.path.exists(Path_Data_Files):
        try:
            os.makedirs(Path_Data_Files)
        except OSError as exc:
            if exc.errno != errno.EEXIST:
                raise
    os.chdir(Path_Data_Files)
    Scratch = Path_Data_Files
    myModel_1 = mdb.Model(name = analysis)
    if "Model-1" in mdb.models:
        del mdb.models["Model-1"]
    #------------------------------------------------------------------------------------------------------------
    ## First model setup ##
    #------------------------------------------------------------------------------------------------------------
    ## getting input variables - material properties, general model data ##
    Fire_Load = 'Standard_Fire'
    # input variables has to be editted to include mechanical properties!!!!
    if section=='IPE': section='I-section'
    else: pass
    model_values, geometry, Emissivity, Poisons_Alu, contactResistance = input_variables.main(myModel_1, section, material, Fire_Load, insulated, interface)
    T, Step_time, Conv_hot, Conv_ambient = model_values
    Emissivity_metal, Emissivity_Ins, Emissivity_Concrete = Emissivity
    if type=='Column': L=1000 #mm
    else: L=3000 #mm
    if floor=='Concrete': load=36000 #Newton
    elif floor=='Lightweight': load=20000

```

```
    if section=='I-section':
        section='IPE'
        H,W,tf,tw,tp,Ws,Hs = geometry
    elif section=='RHS':
        H,W,t,tp,Ws,Hs = geometry

    #---------------------------------------------------------------------------------------------------------------
    ## Sketch + Part ##
    if elements=='volume':
        if section=='RHS':
            if type!='Beam1':
                e=0
            else: #integrated RHS beam
                e=16
            # RHS part
            mySketch_1 = myModel_1.ConstrainedSketch(name='RHS', sheetSize=0.2)
            xyCoords = ((W,H),(W,Hs),(W,t),(W+e,t),(W+e,0),(-e,0),(-e,t),(0,t),(0,Hs),(0,H),(W,H))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1=xyCoords[i], point2=xyCoords[i+1])
            mySketch_1.rectangle(point1=(t,t), point2=(W-t,H-t))
            myPart_1 = myModel_1.Part(name = 'RHS', dimensionality = THREE_D, type=DEFORMABLE_BODY)
            myPart_1.BaseShellExtrude(depth=L  ,sketch = mySketch_1)
        elif section=='Decking':
            # Part 1 - Decking #
            mySketch_1 = myModel_1.ConstrainedSketch(name=section, sheetSize=0.2)
            side = c*tw
            xyCoords_outer = ((0,0), (W+10*side,0), (W+10*side, -tf-side), ((0.9*W+11*side-c*tf), -H),
                ((0.1*W+c*tf-side), -H), (0, -tf-side), (0,0))
            # Please note: Coordinates have to be such order that section can be drawn fluently
            for i in range (len(xyCoords_outer)-1): mySketch_1.Line(point1 = xyCoords_outer [i],point2 = xyCoords_outer [i+1])
            #first cut out
            xyCoords = ((2*side,-tf),(0.2*W,-tf),(0.1*W+side,-H+tf),(2*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #second cut out
            xyCoords = ((0.2*W+2*side,-tf),(0.3*W+side,-H+tf),(0.1*W+3*side,-H+tf),(0.2*W+2*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #third cutout
            xyCoords = ((0.2*W+4*side,-tf),(0.4*W+2*side,-tf),(0.3*W+3*side,-H+tf),(0.2*W+4*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #fourth cutout
            xyCoords = ((0.4*W+4*side,-tf),(0.5*W+3*side,-H+tf),(0.3*W+5*side,-H+tf),(0.4*W+4*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #fifth cutout
            xyCoords = ((0.4*W+6*side,-tf), (0.5*W+5*side,-H+tf),(0.6*W+4*side,-tf),(0.4*W+6*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #sixth cutout
            xyCoords = ((0.6*W+6*side,-tf),(0.5*W+7*side,-H+tf),(0.7*W+5*side,-H+tf),(0.6*W+6*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #seventh cutout
            xyCoords =
```

```
            ((0.6*W+8*side,-tf),(0.8*W+6*side,-tf),(0.7*W+7*side,-H+tf),(0.6*W+8*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #eighth cutout
            xyCoords = ((0.8*W+8*side,-tf),(0.7*W+9*side,-H+tf),(0.9*W+7*side,-H+tf),(0.8*W+8*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            #ninth cutout
            xyCoords = ((0.8*W+10*side,-tf),(1*W+8*side,-tf),(0.9*W+9*side,-H+tf),(0.8*W+10*side,-tf))
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            myPart_1 = myModel_1.Part(name = section, dimensionality = THREE_D, type=DEFORMABLE_BODY)
            myPart_1.BaseShellExtrude(depth= L, sketch = mySketch_1)
        elif section=='IPE':
            mySketch_1 = myModel_1.ConstrainedSketch(name=section, sheetSize=0.2)
            xyCoords = ((-(0.5*W), -tf), (-(0.5*W), 0), (0.5*W, 0), (0.5*W, -tf),
                ((0.5*tw), -tf), ((0.5*tw), -H+tf+Hs),(0.5*tw,-H+tf), (0.5*W, -(H-tf)), (0.5*W, -H),
                (-(0.5*W), -H), (-(0.5*W), -(H-tf)), (-(0.5*tw), -(H-tf)), (-0.5*tw,-H+Hs+tf),
                (-(0.5*tw), -tf), (-(0.5*W), -tf))
            # Please note: Coordinates have to be such order that section can be drawn fluently
            for i in range (len(xyCoords)-1): mySketch_1.Line(point1 = xyCoords [i],point2 = xyCoords [i+1])
            myPart_1 = myModel_1.Part(name = section, dimensionality = THREE_D, type=DEFORMABLE_BODY)
            myPart_1.BaseShellExtrude(depth=L, sketch = mySketch_1)

    elif elements=='shell':
        if section=='RHS':
            e=-0.5*t #mm
            if type=='Beam1':
                e=16 #mm
            # RHS member section
            mySketch_1 = myModel_1.ConstrainedSketch(name='RHS', sheetSize=0.2)
            mySketch_1.Line(point1=(-e,0.5*t),point2=(W+e,0.5*t))
            mySketch_1.Line(point1=(0.5*t,0.5*t),point2=(0.5*t,H-0.5*t))
            mySketch_1.Line(point1=(0.5*t,H-0.5*t), point2=(W-0.5*t,H-0.5*t))
            mySketch_1.Line(point1=(W-0.5*t,H-0.5*t), point2=(W-0.5*t,0.5*t))
            myPart_1 = myModel_1.Part(dimensionality=THREE_D, name='RHS', type = DEFORMABLE_BODY)
            myPart_1.BaseShellExtrude(depth=L,sketch=mySketch_1)
        elif section=='IPE':
            # member section IPE
            mySketch_1 = myModel_1.ConstrainedSketch(name='IPE', sheetSize =0.2)
            mySketch_1.Line(point1=(-0.5*W,-0.5*tf), point2=(0.5*W,-0.5*tf))
            mySketch_1.Line(point1=(0,-0.5*tf), point2=(0,-(H-(0.5*tf))))
            mySketch_1.Line(point1=(-0.5*W,-(H-(0.5*tf))), point2=(0.5*W,-(H-(0.5*tf))))
            myPart_1 = myModel_1.Part(dimensionality=THREE_D, name='IPE', type=DEFORMABLE_BODY)
            myPart_1.BaseShellExtrude(depth=L, sketch=mySketch_1)

        elif section=='Decking':
            pass
    # -----------------------------------------------------------------------------------------------
    ## Section ##
    # integration points over thickness can be inputted here, default at 5IP's
    if elements =='shell':
        if section=='IPE':
            Flanges = myPart_1.faces.findAt(
                ((-(0.2*W),-(0.5*tf),(0.1*L)),),((0.2*W,-0.5*tf,0.1*L),),((-0.2*W,-(H-(0.5*tf)),0.1*L),),((0.2*W,-(H-(0.5*tf)),0.1*L),),
```

```
                ((-(0.2*W),-(0.5*tf),(0.5*L)),),((0.2*W,-0.5*tf,0.5*L),),((-0.2*W,-(H-(0.5*tf)),0.5*L),),((0.2*W,-(H-(0.5*tf)),0.5*L),),
                ((-(0.2*W),-(0.5*tf),(0.8*L)),),((0.2*W,-0.5*tf,0.8*L),),((-0.2*W,-(H-(0.5*tf)),0.8*L),),((0.2*W,-(H-(0.5*tf)),0.8*L),),)
            Set_11  = myPart_1.Set(name='Flanges', faces = (Flanges,))
            myModel_1.HomogeneousShellSection(material=material, name ='Flanges', thickness=tf)
            myModel_1.parts['IPE'].SectionAssignment(offset=0.0, offsetType = MIDDLE_SURFACE,
                region = myModel_1.parts['IPE'].sets['Flanges'], sectionName = 'Flanges',
                thicknessAssignment = FROM_SECTION)

            Web = myPart_1.faces.findAt( ((0,-0.5*H,0.5*L),),)
            Set_12 = myPart_1.Set(name='Web', faces = (Web,))
            myModel_1.HomogeneousShellSection(material=material, name='Web', thickness = tw)
            myModel_1.parts['IPE'].SectionAssignment(offset=0.0, offsetType=MIDDLE_SURFACE,
                region = myModel_1.parts['IPE'].sets['Web'], sectionName='Web',
                thicknessAssignment = FROM_SECTION)
            ## partitions for loading
            myPart_1.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=1./3*L)
            myPart_1.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=2./3*L)
            myPart_1.PartitionFaceByDatumPlane(datumPlane=myPart_1.datums[4], faces =
                myPart_1.faces.findAt(((-0.2*W,-0.5*tf,0.5*L),),((0.2*W,-0.5*tf,0.5*L),),
                    ((-0.2*W,-H+0.5*tf,0.5*L),),((0.2*W,-H+0.5*tf,0.5*L),),))
            myPart_1.PartitionFaceByDatumPlane(datumPlane=myPart_1.datums[5], faces =
                myPart_1.faces.findAt(((-0.2*W,-0.5*tf,0.5*L),),((0.2*W,-0.5*tf,0.5*L),),
                    ((-0.2*W,-H+0.5*tf,0.5*L),),((0.2*W,-H+0.5*tf,0.5*L),),))
            myPart_1.Set(name='Load_top1', vertices = myPart_1.vertices.findAt(
                ((0,-0.5*tf,1./3*L),),((-0.5*W,-0.5*tf,1./3*L),),((0.5*W,-0.5*tf,1./3*L),),))
            myPart_1.Set(name='Load_top2', vertices = myPart_1.vertices.findAt(
                ((0,-0.5*tf,2./3*L),),((-0.5*W,-0.5*tf,2./3*L),),((0.5*W,-0.5*tf,2./3*L),),))
            myPart_1.Set(name='Load_bottom1', vertices = myPart_1.vertices.findAt(
                ((0,-H+0.5*tf,1./3*L),),((-0.5*W,-H+0.5*tf,1./3*L),),((0.5*W,-H+0.5*tf,1./3*L),),))
            myPart_1.Set(name='Load_bottom2', vertices = myPart_1.vertices.findAt(
                ((0,-H+0.5*tf,2./3*L),),((-0.5*W,-H+0.5*tf,2./3*L),),((0.5*W,-H+0.5*tf,2./3*L),),))

        elif section=='RHS' :
            if type=='Beam1':
                Sides = myPart_1.faces.findAt((((W-0.5*t),0.5*H,0.1*L),),((0.5*W,(H-0.5*t),0.1*L),),
                    ((0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.5*L),),((-0.5*e,0.5*t,0.1*L),),((W+0.5*e,0.5*t,0.1*L),),
                    (((W-0.5*t),0.5*H,0.5*L),),((0.5*W,(H-0.5*t),0.5*L),),
                    ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),((-0.5*e,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),
                    (((W-0.5*t),0.5*H,0.8*L),),((0.5*W,(H-0.5*t),0.8*L),),
                    ((0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),((-0.5*e,0.5*t,0.8*L),),((W+0.5*e,0.5*t,0.8*L),),)
            else:
                Sides = myPart_1.faces.findAt(
                    ((W-0.5*t,0.5*H,0.5*L),),((0.5*W,H-0.5*t,0.5*L),),
                    ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),
                    ((W-0.5*t,0.5*H,0.1*L),),((0.5*W,(H-0.5*t),0.1*L),),
                    ((0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.1*L),),
                    ((W-0.5*t,0.5*H,0.8*L),),((0.5*W,(H-0.5*t),0.8*L),),
                    ((0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),)
            Set_11 = myPart_1.Set(name='Flanges', faces = (Sides,))
```

```
            myModel_1.HomogeneousShellSection(material=material, name ='Flanges', thickness=t)
            myModel_1.parts['RHS'].SectionAssignment(offset=0.0, offsetType=MIDDLE_SURFACE,
                region = myModel_1.parts['RHS'].sets['Flanges'],
                sectionName='Flanges',
                thicknessAssignment = FROM_SECTION)
            ## partitions for loading
            myPart_1.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=1./3*L)
            myPart_1.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=2./3*L)
            myPart_1.PartitionFaceByDatumPlane(datumPlane=myPart_1.datums[3], faces =
                myPart_1.faces.findAt((((W-0.5*t),0.5*H,0.5*L),),((0.5*W,(H-0.5*t),0.5*L),),
                    ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),
                    ((-0.5*e,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),))
            myPart_1.PartitionFaceByDatumPlane(datumPlane=myPart_1.datums[4], faces =
                myPart_1.faces.findAt(((W-0.5*t,0.5*H,0.5*L),),((0.5*W,(H-0.5*t),0.5*L),),
                    ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),
                    ((-0.5*e,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),))
            myPart_1.Set(name='Load_top1', vertices = myPart_1.vertices.findAt(
                ((0.5*t,H-0.5*t, 1./3*L),),((W-0.5*t,H-0.5*t,1./3*L),),))
            myPart_1.Set(name='Load_top2', vertices = myPart_1.vertices.findAt(
                ((0.5*t,H-0.5*t, 2./3*L),),((W-0.5*t,H-0.5*t,2./3*L),),))
            myPart_1.Set(name='Load_bottom1', vertices = myPart_1.vertices.findAt(
                ((-e,0.5*t, 1./3*L),),((W+e,0.5*t,1./3*L),),
                ((0.5*t,0.5*t, 1./3*L),),((W-0.5*t,0.5*t,1./3*L),),))
            myPart_1.Set(name='Load_bottom2', vertices = myPart_1.vertices.findAt(
                ((-e,0.5*t, 2./3*L),),((W+e,0.5*t,2./3*L),),
                ((0.5*t,0.5*t, 2./3*L),),((W-0.5*t,0.5*t,2./3*L),),))
        elif section=='Decking':
            pass

    # section if volume elements
    elif elements =='volume':
        pass
    #-----------------------------------------------------------------------------------------------------------
    ## Assembly ##
    myAssembly = myModel_1.rootAssembly
    myAssembly.DatumCsysByDefault (CARTESIAN)
    myAssembly.Instance(dependent = ON, part = myPart_1, name = (section+'-1'))
    #-----------------------------------------------------------------------------------------------------------
    ## Mesh ##
    if elements=='shell':
        if section=='IPE':
            mySet_19 = myAssembly.Set(name='IPE', faces = myAssembly.instances['IPE-1'].faces.findAt(
                ((0.1*W,-0.5*tf,0.1*L),),((-0.1*W,-0.5*tf,0.1*L),),((0.1*W,-H+0.5*tf,0.1*L),),((-0.1*W,-H+0.5*tf,0.1*L),),
                ((0.1*W,-0.5*tf,0.5*L),),((-0.1*W,-0.5*tf,0.5*L),),((0.1*W,-H+0.5*tf,0.5*L),),((-0.1*W,-H+0.5*tf,0.5*L),),
                ((0.1*W,-0.5*tf,0.8*L),),((-0.1*W,-0.5*tf,0.8*L),),((0.1*W,-H+0.5*tf,0.8*L),),((-0.1*W,-H+0.5*tf,0.8*L),),
                ((0,-0.5*H,0.1*L),),))
        elif section=='RHS':
            if type=='Beam1':
                mySet_19 = myAssembly.Set(name='RHS', faces = myAssembly.instances['RHS-1'].faces.findAt(
                    ((-0.5*e,0.5*t,0.1*L),),((0.5*W,0.5*t,0.1*L),),((W+0.5*e,0.5*t,0.1*L),),
                    ((0.5*t,0.5*H,0.1*L),),((0.5*W,H-0.5*t,0.1*L),),((W-0.5*t,0.5*H,0.
```

```
                        1*L),),
                        ((-0.5*e,0.5*t,0.5*L),),((0.5*W,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),
                        ((0.5*t,0.5*H,0.5*L),),((0.5*W,H-0.5*t,0.5*L),),((W-0.5*t,0.5*H,0.5*L),),
                        ((-0.5*e,0.5*t,0.8*L),),((0.5*W,0.5*t,0.8*L),),((W+0.5*e,0.5*t,0.8*L),),
                        ((0.5*t,0.5*H,0.8*L),),((0.5*W,H-0.5*t,0.8*L),),((W-0.5*t,0.5*H,0.8*L),),))
                else:
                    mySet_19 = myAssembly.Set(name='RHS', faces =
                    myAssembly.instances['RHS-1'].faces.findAt(
                        ((0.5*t,0.5*H,0.1*L),),((0.5*W,H-0.5*t,0.1*L),),((W-0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.1*L),),
                        ((0.5*t,0.5*H,0.5*L),),((0.5*W,H-0.5*t,0.5*L),),((W-0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),
                        ((0.5*t,0.5*H,0.8*L),),((0.5*W,H-0.5*t,0.8*L),),((W-0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),))
            elif section=='Decking':
                pass
        elif elements=='volume':
            pass
        myAssembly.setElementType (elemTypes= (ElemType( elemCode = S4R, elemLibrary =
            STANDARD),), regions = mySet_19)
        myModel_1.parts[section].seedPart(deviationFactor=0.1, minSizeFactor = 0.1,
        size=mesh_size)
        myModel_1.parts[section].generateMesh()


        #-----------------------------------------------------------------------------------------------------------
        ## Start incrementation ##
        increment = 0
        while increment<=numFrames:
            print('Start incrementation '+ analysis+': '+str(increment))
            model_Name_2 = str(increment) +'_'+ analysis
            myModel_2 = mdb.Model(name=model_Name_2)
            myModel_2.setValues(absoluteZero = -273.15, stefanBoltzmann = 5.67e-11)
            # Load part / material / Section #
            myPart_21 = myModel_2.Part(section, myModel_1.parts[section])
            myModel_2.Material(material, myModel_1.materials[material])
            myModel_2.Section('Flanges', myModel_1.sections['Flanges'])
            if section=='IPE':
                myModel_2.Section('Web', myModel_1.sections['Web'])
            myModel_2.Instance(section+'-1', myAssembly.instances[section+'-1'])

            if increment>1: step = 5
            else: step = 1
            NewJob = '3D_Model_GA_new'+str(increment)
            PrevJob = '3D_Model_GA_new'+str(increment-step)
            NewStep = 'General_Analysis_'+str(increment)
            PrevStep = 'General_Analysis_'+str(increment-step)
            # Loading Restart File #
            if increment>0:
                myModel_2.setValues(restartJob = PrevJob, restartStep = PrevStep,
                    restartIncrement = STEP_END)


            #-------------------------------------------------------------------------------------------------------
            ## Assembly ##
            #print('Start assembly')
            myAssembly = myModel_2.rootAssembly
            #
            ---------------------------------------------------------------------------------
```

```
------------------
## Surfaces ##
#print('Start surfaces')
if elements=='shell':
    if section=='RHS':
        mySurface_11 = myAssembly.Surface(name='Top Beam', side2Faces=
            myAssembly.instances['RHS-1'].faces.findAt( ((t, H-0.5*t,0.1*L),),
                ((t, H-0.5*t,0.5*L),),((t, H-0.5*t,0.8*L),),))
        if type=='Beam1':
            mySurface_13 = myAssembly.Surface(name='Side face1 RHS', side1Edges =
                myAssembly.instances['RHS-1'].edges.findAt(
                    ((0.5*t, 0.5*H,0),),((0.1*W,H-0.5*t,0),),((W-0.5*t,0.5*H,0),),
                    ((0.1*W,0.5*t,0),),((-0.5*e,0.5*t,0),),((W+0.5*e,0.5*t,0),),))
            mySurface_12 = myAssembly.Surface(name='Bottom flange', side2Faces=
                myAssembly.instances['RHS-1'].faces.findAt(
                    ((-0.25*e,0.5*t,0.1*L),),((W+0.25*e,0.5*t,0.1*L),),
                    ((-0.25*e,0.5*t,0.5*L),),((W+0.25*e,0.5*t,0.5*L),),
                    ((-0.25*e,0.5*t,0.8*L),),((W+0.25*e,0.5*t,0.8*L),),))
        else:
            mySurface_13 = myAssembly.Surface(name='Side face1 RHS', side1Edges =
                myAssembly.instances['RHS-1'].edges.findAt(
                    ((0.5*t, 0.5*H,0),),((0.1*W,H-0.5*t,0),),
                    ((W-0.5*t,0.5*H,0),),((0.1*W,0.5*t,0),),))
    elif section=='IPE':
        mySurface_11 = myAssembly.Surface(name='Top beam', side2Faces =
            myAssembly.instances['IPE-1'].faces.findAt(
                ((0.1*W,-0.5*tf,0.1*L),),((-0.1*W,-0.5*tf,0.1*L),),
                ((0.1*W,-0.5*tf,0.5*L),),((-0.1*W,-0.5*tf,0.5*L),),
                ((0.1*W,-0.5*tf,0.8*L),),((-0.1*W,-0.5*tf,0.8*L),),))
        mySurface_13 = myAssembly.Surface(name='Side face1 IPE', side1Edges =
            myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-0.5*tf,0),),((0.1*W,-H+0.5*tf,0),),((0,-0.5*H,0),),
                ((-0.1*W,-0.5*tf,0),),((-0.1*W,-H+0.5*tf,0),),))
        if type=='Beam1':
            mySurface_12 = myAssembly.Surface(name='Bottom Flange beam', side2Faces=
                myAssembly.instances['IPE-1'].faces.findAt(
                ((-0.2*W,-H+0.5*tf,0.1*L),),((0.2*W,-H+0.5*tf,0.1*L),),
                ((-0.2*W,-H+0.5*tf,0.5*L),),((0.2*W,-H+0.5*tf,0.5*L),),
                ((-0.2*W,-H+0.5*tf,0.8*L),),((0.2*W,-H+0.5*tf,0.8*L),)))
    elif section=='Decking':
        pass
elif elements=='volume':
    pass
#-----------------------------------------------------------------------------------------------
## Sets ##
#print('Start sets')
if elements=='shell':
    if section=='IPE':
        # in part instance
        Flanges = myPart_21.faces.findAt(
          ((-(0.2*W),-(0.5*tf),(0.1*L)),),((0.2*W,-0.5*tf,0.1*L),),((-0.2*W,-(H-(0.5*tf)),0.1*L),),((0.2*W,-(H-(0.5*tf)),0.1*L),),
          ((-(0.2*W),-(0.5*tf),(0.5*L)),),((0.2*W,-0.5*tf,0.5*L),),((-0.2*W,-(H-(0.5*tf)),0.5*L),),((0.2*W,-(H-(0.5*tf)),0.5*L),),
          ((-(0.2*W),-(0.5*tf),(0.8*L)),),((0.2*W,-0.5*tf,0.8*L),),((-0.2*W,-(H-(0.5*tf)),0.8*L),),((0.2*W,-(H-(0.5*tf)),0.8*L),),)
        Set_11  = myPart_21.Set(name='Flanges', faces = (Flanges,))
        Web = myPart_21.faces.findAt( ((0,-0.5*H,0.5*L),),)
        Set_12 = myPart_21.Set(name='Web', faces = (Web,))
        # in assembly
```

```
            mySet_11 = myAssembly.Set(name='Top Flange edge1', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-0.5*tf,0),),((-0.1*W,-0.5*tf,0),),))
            mySet_12 = myAssembly.Set(name='Top Flange edge2', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-0.5*tf,L),),((-0.1*W,-0.5*tf,L),),))
            mySet_13 = myAssembly.Set(name='Bottom Flange edge1', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-H+0.5*tf,0),),((-0.1*W,-H+0.5*tf,0),),))
            mySet_14 = myAssembly.Set(name='Bottom Flange edge2', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-H+0.5*tf,L),),((-0.1*W,-H+0.5*tf,L),),))
            mySet_15 = myAssembly.Set(name='Side face1 IPE', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-0.5*tf,0),),((0.1*W,-H+0.5*tf,0),),((0,-0.5*H,0),),((-0.1*W,-0.5*tf,0),),((-0.1*W,-H+0.5*tf,0),),))
            mySet_16 = myAssembly.Set(name='Side face2 IPE', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.1*W,-0.5*tf,L),),((0.1*W,-H+0.5*tf,L),),((0,-0.5*H,L),),((-0.1*W,-0.5*tf,L),),((-0.1*W,-H+0.5*tf,L),),))
            # Edges of flange on one side, left followed by right side
            mySet_17 = myAssembly.Set(name='Side edge IPE', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((-0.5*W,-0.5*tf,0.1*L),),((-0.5*W,-H+0.5*tf,0.1*L),),
                ((-0.5*W,-0.5*tf,0.5*L),),((-0.5*W,-H+0.5*tf,0.5*L),),
                ((-0.5*W,-0.5*tf,0.8*L),),((-0.5*W,-H+0.5*tf,0.8*L),),))
            mySet_18 = myAssembly.Set(name='Side edge2 IPE', edges = myAssembly.instances['IPE-1'].edges.findAt(
                ((0.5*W,-0.5*tf, 0.1*L),),((0.5*W,-H+0.5*tf,0.1*L),),
                ((0.5*W,-0.5*tf, 0.5*L),),((0.5*W,-H+0.5*tf,0.5*L),),
                ((0.5*W,-0.5*tf, 0.8*L),),((0.5*W,-H+0.5*tf,0.8*L),),))
            # full beam
            mySet_19 = myAssembly.Set(name='IPE', faces = myAssembly.instances['IPE-1'].faces.findAt(
                ((0.1*W,-0.5*tf,0.1*L),),((-0.1*W,-0.5*tf,0.1*L),),((0.1*W,-H+0.5*tf,0.1*L),),((-0.1*W,-H+0.5*tf,0.1*L),),
                ((0.1*W,-0.5*tf,0.5*L),),((-0.1*W,-0.5*tf,0.5*L),),((0.1*W,-H+0.5*tf,0.5*L),),((-0.1*W,-H+0.5*tf,0.5*L),),
                ((0.1*W,-0.5*tf,0.8*L),),((-0.1*W,-0.5*tf,0.8*L),),((0.1*W,-H+0.5*tf,0.8*L),),((-0.1*W,-H+0.5*tf,0.8*L),),
                ((0,-0.5*H,0.1*L),),))

        elif section=='RHS':
            mySet_11 = myAssembly.Set(name='Top edge1', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.1*W,H-0.5*t,0),),))
            mySet_12 = myAssembly.Set(name='Top edge2', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.1*W,H-0.5*t,L),),))
            mySet_13 = myAssembly.Set(name='Bottom edge1', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.1*W,0.5*t,0),),((-0.5*e,0.5*t,0),),((W+0.5*e,0.5*t,0),),))
            mySet_14 = myAssembly.Set(name ='Bottom edge2', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.1*W,0.5*t,L),),((-0.5*e,0.5*t,L),),((W+0.5*e,0.5*t,L),),))

            if type=='Beam1':
            # part instance
                Sides = myPart_21.faces.findAt(
                (((W-0.5*t),0.5*H,0.5*L),),((0.5*W,(H-0.5*t),0.5*L),),
                    ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),((-0.5*e,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),
                    (((W-0.5*t),0.5*H,0.1*L),),((0.5*W,(H-0.5*t),0.1*L),),
                    ((0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.1*L),),((-0.5*e,0.5*t,0.1*L),),((W+0.5*e,0.5*t,0.1*L),),
```

```
                (((W-0.5*t),0.5*H,0.8*L),),((0.5*W,(H-0.5*t),0.8*L),),
                ((0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),((-0.5*e,0.5*t,0.8*L),),((W+0.5*e,0.5*t,0.8*L),),)
            Set_11 = myPart_1.Set(name='Flanges', faces = (Sides,))
            mySet_15 = myAssembly.Set(name='Side face1 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.5*t, 0.5*H,0),),((0.1*W,H-0.5*t,0),),((W-0.5*t,0.5*H,0),),((0.1*W,0.5*t,0),),
                ((-0.5*e,0.5*t,0),),((W+0.5*e,0.5*t,0),),))
            mySet_16 = myAssembly.Set(name='Side face2 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.5*t, 0.5*H,L),),((0.1*W,H-0.5*t,L),),((W-0.5*t,0.5*H,L),),((0.1*W,0.5*t,L),),
                ((-0.5*e,0.5*t,L),),((W+0.5*e,0.5*t,L),),))
            mySet_17 = myAssembly.Set(name='Side edge1 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((-e,0.5*t,0.1*L),),((0.5*t,H-0.5*t,0.1*L),),
                ((-e,0.5*t,0.5*L),),((0.5*t,H-0.5*t,0.5*L),),
                ((-e,0.5*t,0.8*L),),((0.5*t,H-0.5*t,0.8*L),),))
            mySet_18 = myAssembly.Set(name='Side edge2 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((W+e,0.5*t,0.1*L),),((W-0.5*t,H-0.5*t,0.1*L),),
                ((W+e,0.5*t,0.5*L),),((W-0.5*t,H-0.5*t,0.5*L),),
                ((W+e,0.5*t,0.8*L),),((W-0.5*t,H-0.5*t,0.8*L),),))
            mySet_19 = myAssembly.Set(name='RHS', faces = myAssembly.instances['RHS-1'].faces.findAt(
                ((-0.5*e,0.5*t,0.1*L),),((0.5*W,0.5*t,0.1*L),),((W+0.5*e,0.5*t,0.1*L),),
                ((0.5*t,0.5*H,0.1*L),),((0.5*W,H-0.5*t,0.1*L),),((W-0.5*t,0.5*H,0.1*L),),
                ((-0.5*e,0.5*t,0.5*L),),((0.5*W,0.5*t,0.5*L),),((W+0.5*e,0.5*t,0.5*L),),
                ((0.5*t,0.5*H,0.5*L),),((0.5*W,H-0.5*t,0.5*L),),((W-0.5*t,0.5*H,0.5*L),),
                ((-0.5*e,0.5*t,0.8*L),),((0.5*W,0.5*t,0.8*L),),((W+0.5*e,0.5*t,0.8*L),),
                ((0.5*t,0.5*H,0.8*L),),((0.5*W,H-0.5*t,0.8*L),),((W-0.5*t,0.5*H,0.8*L),),))

        else:
            Sides = myPart_1.faces.findAt( (((W-0.5*t),0.5*H,0.5*L),),((0.5*W,(H-0.5*t),0.5*L),),
                ((0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),
                (((W-0.5*t),0.5*H,0.1*L),),((0.5*W,(H-0.5*t),0.1*L),),
                ((0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.1*L),),
                (((W-0.5*t),0.5*H,0.8*L),),((0.5*W,(H-0.5*t),0.8*L),),
                ((0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),)
            Set_11 = myPart_1.Set(name='Flanges', faces = (Sides,))
            mySet_15 = myAssembly.Set(name='Side face1 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.5*t, 0.5*H,0),),((0.1*W,H-0.5*t,0),),((W-0.5*t,0.5*H,0),),((0.1*W,0.5*t,0),),
                ))
            mySet_16 = myAssembly.Set(name='Side face2 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.5*t, 0.5*H,L),),((0.1*W,H-0.5*t,L),),((W-0.5*t,0.5*H,L),),((0.1*W,0.5*t,L),),
                ))
            mySet_17 = myAssembly.Set(name='Side edge1 RHS', edges = myAssembly.instances['RHS-1'].edges.findAt(
                ((0.5*t,0.5*t,0.1*L),),((0.5*t,H-0.5*t,0.1*L),),
```

```
                        ((0.5*t,0.5*t,0.5*L),),((0.5*t,H-0.5*t,0.5*L),),
                        ((0.5*t,0.5*t,0.8*L),),((0.5*t,H-0.5*t,0.8*L),),))
                    mySet_18 = myAssembly.Set(name='Side edge2 RHS', edges =
                    myAssembly.instances['RHS-1'].edges.findAt(
                        ((W-0.5*t,0.5*t,0.1*L),),((W-0.5*t,H-0.5*t,0.1*L),),
                        ((W-0.5*t,0.5*t,0.5*L),),((W-0.5*t,H-0.5*t,0.5*L),),
                        ((W-0.5*t,0.5*t,0.8*L),),((W-0.5*t,H-0.5*t,0.8*L),),))
                    mySet_19 = myAssembly.Set(name='RHS', faces =
                    myAssembly.instances['RHS-1'].faces.findAt(

                        ((0.5*t,0.5*H,0.1*L),),((0.5*W,H-0.5*t,0.1*L),),((W-0.5*t,0.5*H,0.1*L),),((0.5*W,0.5*t,0.1*L),),

                        ((0.5*t,0.5*H,0.5*L),),((0.5*W,H-0.5*t,0.5*L),),((W-0.5*t,0.5*H,0.5*L),),((0.5*W,0.5*t,0.5*L),),

                        ((0.5*t,0.5*H,0.8*L),),((0.5*W,H-0.5*t,0.8*L),),((W-0.5*t,0.5*H,0.8*L),),((0.5*W,0.5*t,0.8*L),),))
            elif section=='Decking':
                pass

        elif elements=='volume':
            pass
        #-------------------------------------------------------------------------------------------
        # Local coordinate system #
        #print('start local coordinate')
        if section=='IPE':
            myLocCoor_1 = myAssembly.DatumCsysByThreePoints(coordSysType = CARTESIAN,
                name='Local Coordinates 1',

                origin=myAssembly.instances[section+'-1'].vertices.findAt((0,-0.5*tf,0),),

                point1=myAssembly.instances[section+'-1'].vertices.findAt((0.5*W,-0.5*tf,0),),

                point2=myAssembly.instances[section+'-1'].vertices.findAt((0,-H+0.5*tf,0),),)
            LC_1 = myLocCoor_1.id
            myModel_2.parts[section].DatumCsysByThreePoints(coordSysType = CARTESIAN,
                name='Datum csys-1',
                origin=myModel_2.parts[section].vertices.findAt((0,-0.5*tf,0),),
                point1=myModel_2.parts[section].vertices.findAt((0.5*W,-0.5*tf,0),),
                point2=myModel_2.parts[section].vertices.findAt((0.5*W,-H+0.5*tf,0),))

            myModel_2.parts[section].MaterialOrientation(additionalRotationType=ROTATION_NONE,
                axis=AXIS_2, angle=0.0, localCsys =
                myModel_2.parts[section].datums[15],
                orientationType=SYSTEM, region=Set_11)

            myModel_2.parts[section].MaterialOrientation(additionalRotationType=ROTATION_NONE,
                axis=AXIS_2, angle=0, localCsys=myModel_2.parts[section].datums[15],
                orientationType=SYSTEM, region=Set_12)
        elif section=='RHS':
            if increment==0: count=12
            else: count+=1
            myLocCoor_1 = myAssembly.DatumCsysByThreePoints(coordSysType = CARTESIAN,
                name='Local Coordinates 1',

                origin=myAssembly.instances[section+'-1'].vertices.findAt((0.5*t,0.5*t,0),),

                point1=myAssembly.instances[section+'-1'].vertices.findAt((0.5*t,H-0.5*t,0),),

                point2=myAssembly.instances[section+'-1'].vertices.findAt((W-0.5*t,H-0.5*t,0),))
            LC_1 = myLocCoor_1.id
```

```
            myModel_2.parts[section].DatumCsysByThreePoints(coordSysType = CARTESIAN,
                name='Datum csys-1',
                origin=myModel_2.parts[section].vertices.findAt((0.5*t,0.5*t,0),),
                point1=myModel_2.parts[section].vertices.findAt((0.5*t,H-0.5*t,0),),
                point2=myModel_2.parts[section].vertices.findAt((W-0.5*t,H-0.5*t,0),))

            myModel_2.parts[section].MaterialOrientation(additionalRotationType=ROTATION_NONE,
                axis=AXIS_2, angle=0, localCsys = myModel_2.parts[section].datums[count],
                orientationType=SYSTEM, region=Set_11)

        # Reference Points #
        if section=='IPE':
            myReferencePoint_1 = myAssembly.ReferencePoint(point=(0,-H+0.5*tf,0))
        elif section=='RHS':
            myReferencePoint_1 = myAssembly.ReferencePoint(point=(0.5*t,0.5*t,0))
        elif section=='Decking':
            pass
        RP_1 = myReferencePoint_1.id
        mySet_RP1 = myAssembly.Set(name='Reference Point 1', referencePoints=(myAssembly.referencePoints[RP_1],))

        if section == 'IPE':
            myReferencePoint_2 = myAssembly.ReferencePoint(point=(0,-0.5*tf,1./3*L))
            myReferencePoint_3 = myAssembly.ReferencePoint(point=(0,-H+0.5*tf,1./3*L))
            myReferencePoint_4 = myAssembly.ReferencePoint(point=(0,-0.5*tf,2./3*L))
            myReferencePoint_5 = myAssembly.ReferencePoint(point=(0,-H+0.5*tf,2./3*L))
        elif section=='RHS':
            myReferencePoint_2 = myAssembly.ReferencePoint(point=(-e,0.5*t,1./3*L))
            myReferencePoint_3 = myAssembly.ReferencePoint(point=(W, 0.5*t, 1./3*L))
            myReferencePoint_4 = myAssembly.ReferencePoint(point=(-e,0.5*t,2./3*L))
            myReferencePoint_5 = myAssembly.ReferencePoint(point=(W, 0.5*t, 2./3*L))
        RP2 = myReferencePoint_2.id
        RP3 = myReferencePoint_3.id
        RP4 = myReferencePoint_4.id
        RP5 = myReferencePoint_5.id
        mySet_RP2 = myAssembly.Set(name='RP2', referencePoints=(myAssembly.referencePoints[RP2],))
        mySet_RP3 = myAssembly.Set(name='RP3', referencePoints=(myAssembly.referencePoints[RP3],))
        mySet_RP4 = myAssembly.Set(name='RP4', referencePoints=(myAssembly.referencePoints[RP4],))
        mySet_RP5 = myAssembly.Set(name='RP5', referencePoints=(myAssembly.referencePoints[RP5],))

        # thermal expansion coefficient #
        path_Properties = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\properties'
        if material=='Aluminium':
            with open(path_Properties+'\\'+'ThermalExpAlu EC9.csv','r') as f:
                reader=(csv.reader(f, delimiter=';'))
                Expansion = ()
                for row, column in enumerate(reader):
                    v=[]
                    for value in column:
                        v=v+[float(value),]
                    Expansion = Expansion + (v,)
                f.close()
        elif material=='Steel':
            with open(path_Properties+'\\'+'ThermalExpSteel EC3.csv', 'r') as f:
                reader=(csv.reader(f, delimiter=';'))
                Expansion=()
                for row, column in enumerate(reader):
                    v=[]
                    for value in column:
                        v=v+[float(value),]
                    Expansion = Expansion + (v,)
                f.close()
        myModel_2.materials[material].Expansion(type=ORTHOTROPIC, temperatureDependency=ON, zero=20,
            table=Expansion)
```

```
        del Expansion, v

        # ----------------------------------------------------------------------------------------------------------
        ## Step ##
        #print('start step')
        if increment== 0:
            myModel_2.StaticStep(name='General_Analysis_0', nlgeom = ON, previous='Initial',
                maxNumInc=1000, initialInc=1, minInc=1e-9)
            # creating restart file
            myModel_2.steps['General_Analysis_0'].Restart(frequency=1, numberIntervals=0,
                overlay=ON, timeMarks=OFF)
        elif increment==1:
            myModel_2.StaticStep(name='General_Analysis_0', nlgeom=ON, previous='Initial',
                maxNumInc=1000, initialInc=1, minInc=1e-9)
            myModel_2.StaticStep(name=NewStep, nlgeom=ON, previous='General_Analysis_0',
                maxNumInc=1000, initialInc=1, minInc=1e-9)
            # creating restart file
            myModel_2.steps[NewStep].Restart(frequency = 1, numberIntervals=0,
                overlay=ON, timeMarks=OFF)
        elif increment>1:
            myModel_2.StaticStep(name=PrevStep, nlgeom = ON, previous='Initial',
                maxNumInc=1000, initialInc=1, minInc=1e-9)
            myModel_2.StaticStep(name=NewStep, nlgeom=ON, previous=PrevStep,
                maxNumInc=1000, initialInc=1, minInc=1e-9)
            # creating restart file
            myModel_2.steps[NewStep].Restart(frequency = 1, numberIntervals=0,
                    overlay=ON, timeMarks=OFF)
        # ----------------------------------------------------------------------------------------------------------
        ## Ties ##
        #print('start ties')
        myModel_2.Coupling(controlPoint=mySet_RP1, couplingType = KINEMATIC,
            influenceRadius = WHOLE_SURFACE, name = 'CP-1', surface=mySurface_13,
            u1=ON,u2=ON, u3=ON,ur1=ON,ur2=ON,ur3=ON)
        if section=='IPE':
            mySurface_4a = myAssembly.Set(name='Top flange', faces = myAssembly.instances['IPE-1'].faces.findAt(
                ((-0.1*W,-0.5*tf,1/4.*L),),((-0.1*W,-0.5*tf,1/2.*L),),((-0.1*W,-0.5*tf,3/4.*L),),
                ((0.1*W,-0.5*tf,1/4.*L),),((0.1*W,-0.5*tf,1/2.*L),),((0.1*W,-0.5*tf,3/4.*L),),))
            mySurface_4b = myAssembly.Set(name='Bottom flange', faces = myAssembly.instances['IPE-1'].faces.findAt(
                ((-0.1*W,-H+0.5*tf,1/4.*L),),((-0.1*W,-H+0.5*tf,1/2.*L),),((-0.1*W,-H+0.5*tf,3/4.*L),),
                ((0.1*W,-H+0.5*tf,1/4.*L),),((0.1*W,-H+0.5*tf,1/2.*L),),((0.1*W,-H+0.5*tf,3/4.*L),),))
        elif section =='RHS':
            mySurface_4a = myAssembly.Set(name='Top flange', faces = myAssembly.instances['RHS-1'].faces.findAt(
                ((0.5*W,H-0.5*t,1/4.*L),),((0.5*W,H-0.5*t,1/2.*L),),((0.5*W,H-0.5*t,3/4.*L),),))
            if type=='Beam3':
                mySurface_4b = myAssembly.Set(name='Bottom flange', faces = myAssembly.instances['RHS-1'].faces.findAt(
                    ((0.5*W,0.5*t,1/4.*L),),((0.5*W,0.5*t,1/2.*L),),((0.5*W,0.5*t,3/4.*L),),))
            elif type=='Beam1':
                mySurface_4b = myAssembly.Set(name='Bottom flange', faces =
```

```
            myAssembly.instances['RHS-1'].faces.findAt(
                ((-0.5*e,0.5*t,1/4.*L),),((0.5*W,0.5*t,1/4.*L),),((W+0.5*e,0.5*t,1/4.*L),),
                ((-0.5*e,0.5*t,1/2.*L),),((0.5*W,0.5*t,1/2.*L),),((W+0.5*e,0.5*t,1/2.*L),),
                ((-0.5*e,0.5*t,3/4.*L),),((0.5*W,0.5*t,3/4.*L),),((W+0.5*e,0.5*t,3/4.*L),),))

    all_nodes = myAssembly.instances[section+'-1'].nodes
    left_nodes_top = []
    length_left = []
    length_right = []
    right_nodes_top = []
    left_nodes_bottom = []
    right_nodes_bottom = []
    for Length in list(range(-5,6,1)):
        length_left.append( L/3.+Length*mesh_size )
        length_right.append( 2*L/3.+Length*mesh_size )

    for n in all_nodes:
        ycoord = n.coordinates[1]
        zcoord = n.coordinates[2]
        if section =='IPE':
            if ycoord == -0.5*tf:
                if zcoord in length_left: left_nodes_top.append(n)
                elif zcoord in length_right: right_nodes_top.append(n)
            elif ycoord==-H+0.5*tf:
                if zcoord in length_left: left_nodes_bottom.append(n)
                elif zcoord in length_right: right_nodes_bottom.append(n)
        else:
            if ycoord== H-0.5*t:
                if zcoord in length_left: left_nodes_top.append(n)
                elif zcoord in length_right: right_nodes_top.append(n)
            elif ycoord == 0.5*t:
                if zcoord in length_left: left_nodes_bottom.append(n)
                elif zcoord in length_right: right_nodes_bottom.append(n)

    left_top = myAssembly.Set(nodes=MeshNodeArray(left_nodes_top), name='left_top')
    left_bottom = myAssembly.Set(nodes=MeshNodeArray(left_nodes_bottom), name='left_bottom')
    right_top = myAssembly.Set(nodes=MeshNodeArray(right_nodes_top), name='right_top')
    right_bottom = myAssembly.Set(nodes=MeshNodeArray(right_nodes_bottom), name='right_bottom')

    myModel_1.RigidBody(name='left_top', refPointRegion=mySet_RP2,
        tieRegion=left_top, refPointAtCOM=ON)
    myModel_1.RigidBody(name='left_bottom', refPointRegion=mySet_RP3,
        tieRegion=left_bottom, refPointAtCOM=ON)
    myModel_1.RigidBody(name='right_top', refPointRegion=mySet_RP4,
        tieRegion=right_top, refPointAtCOM=ON)
    myModel_1.RigidBody(name='right_bottom', refPointRegion=mySet_RP5,
        tieRegion=right_bottom, refPointAtCOM=ON)

    ## Boundary conditions ##
    # these can differ between top edge, bottom edge or end face
    if type=='Column':
        myModel_2.DisplacementBC (createStepName='Initial', name= 'Bottom',
            u1=0,u2=0,u3=0,ur1=0, ur2=0, ur3=0, region= mySet_RP1)

    else:
        myModel_2.DisplacementBC (createStepName='Initial', name = 'Hinge',
            u1=UNSET, u2=0, u3=0, ur1=UNSET, ur2=UNSET, ur3=UNSET, region= mySet_RP1)
        myModel_2.DisplacementBC (createStepName='Initial', name = 'Roller',
            u1=UNSET,u2=0,u3=UNSET,ur1=UNSET,ur2=UNSET,ur3=UNSET, region= mySet_12)
        myModel_2.DisplacementBC (createStepName= 'Initial', name='Lateral',
```

```python
                u1=0,u2=UNSET,u3=UNSET,ur1=UNSET,ur2=UNSET,ur3=UNSET, region=mySet_17)

        # -----------------------------------------------------------------------------------------------------------
        ## Loads ##
        if section == 'RHS': divide = 2
        elif section =='RHS' and type=='Beam1': divide=4
        else: divide = 3
        if type=="Beam3":
            myModel_2.ConcentratedForce(name='Load1', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_top1'],
                cf2 = -load/divide, distributionType=UNIFORM, field='', localCsys=None)
            myModel_2.ConcentratedForce(name='Load2', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_top2'],
                cf2 = -load/divide, distributionType=UNIFORM, field='', localCsys=None)
        elif type=='Beam1':
            myModel_2.ConcentratedForce(name='Load1', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_top1'],
                cf2 = -0.25*load/divide, distributionType=UNIFORM, field='', localCsys=None)
            myModel_2.ConcentratedForce(name='Load2', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_top2'],
                cf2 = -0.25*load/divide, distributionType=UNIFORM, field='', localCsys=None)
            myModel_2.ConcentratedForce(name='Load3', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_bottom1'],
                cf2 = -0.75*load/divide, distributionType=UNIFORM, field='', localCsys=None)
            myModel_2.ConcentratedForce(name='Load4', createStepName = NewStep,
                region = myAssembly.instances[section+'-1'].sets['Load_bottom2'],
                cf2 = -0.75*load/divide, distributionType=UNIFORM, field='', localCsys=None)

        # -----------------------------------------------------------------------------------------------------------
        ## Predifined field ##
        ## Initial ##
        #print('Start predefened field: '+str(increment))
        if increment==0:
            myModel_2.Temperature (createStepName = 'Initial', crossSectionDistribution =
                CONSTANT_THROUGH_THICKNESS, distributionType=UNIFORM, magnitudes=(20.0,),
                name = 'Initial Temperature', region = mySet_19)
        if increment>0:
            myModel_2.InitialState(createStepName='Initial', endIncrement=STEP_END, endStep=LAST_STEP,
                fileName=PrevJob, instances=(myAssembly.instances[section+'-1'],),
                name='Initial Temperature', updateReferenceConfiguration=OFF)
            if type!='Column':
                Thermal = 'Thermal_Analysis_'+type+'_'+section+'_'+material+'_Ins_'+insulated+'_'+floor
            elif type=='Column':
                Thermal = 'Thermal_Analysis_'+type+'_'+section+'_'+material+'_Ins_'+insulated
            path = r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS"+"\\"+Thermal
            myList = input_variables.Temperature_field(path,L, Thermal, increment, mesh_size)

            myModel_2.MappedField(description = 'midside', fieldDataType = SCALAR, localCsys = None,
                name = 'Coord_Temp_field', partLevelData = False, pointDataFormat = XYZ,
                regionType = POINT, xyzPointData = myList)
            myModel_2.Temperature(createStepName=NewStep, crossSectionDistribution=
```

```
        CONSTANT_THROUGH_THICKNESS, distributionType=FIELD, field = 'Coord_Temp_field',
        interpolate=MIDSIDE_ONLY, magnitudes=(1.0,), name = 'Temperature', region = mySet_19)
    del myList

# -----------------------------------------------------------------------------------------------------------
## Output request ##
Variables = ('COORD','TEMP','S', 'U','LE','PE','E')
myModel_2.fieldOutputRequests['F-Output-1'].setValues(variables = Variables, frequency = Step_time)
# -----------------------------------------------------------------------------------------------------------
print('Start job of increment '+analysis+': '+str(increment))
## Job ##
try:
    try:
        if increment==0:
            myModel_2.keywordBlock.synchVersions(storeNodesAndElements=False)
            myJob = mdb.Job(name = NewJob, model = myModel_2, type = ANALYSIS, scratch = Scratch)
        else:
            myJob = mdb.Job(name = NewJob, model = myModel_2, type = RESTART, scratch = Scratch)
        myJob.submit(consistencyChecking=OFF)
        myJob.waitForCompletion()

        # result writing to csv
        if increment==0: step_name='General_Analysis_0'
        else: step_name = NewStep
        odb = session.openOdb(name = NewJob +'.odb')
        csv_writer_thermal_analysis.CSV_writer_mechanical(odb, NewJob, step_name)
        odb.close()

        if myJob.status != ABORTED:
            if increment == 0: increment+=1
            else: increment+=5
        else:
            odb = session.openOdb(name = NewJob + '.odb', readOnly=False)
            odb.save()
            break
    except:
        odb = session.openOdb(name = NewJob + '.odb', readOnly=False)
        odb.save()
        break
except OdbError, error:
    print(error)
    break
```

G: POSTPROCESSING SCRIPT

```python
##  figure plotting  ##
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.ticker as ticker
import matplotlib.gridspec as gridspec
import matplotlib.patches as patches
from matplotlib.lines import Line2D
from matplotlib.ticker import AutoMinorLocator
from mpl_toolkits.axisartist.axislines import SubplotZero
import scipy
from scipy import ndimage
from scipy import interpolate
from scipy.signal import savgol_filter
from statsmodels.nonparametric.smoothers_lowess import lowess
import matplotlib.transforms as mtransforms
import pandas as pd
import numpy as np
import copy
import sys
import csv
import cowsay
sys.path.append(r"D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Scripts\Intumescent paint")

# Mechanische analyse
def read_data(numframes, model, variable, location): #variable of type ['Coord', 'Temp','Stresses','Displacements','Rotations','Log_strains','Plastic_strains']
    try:
        if location=='': location==r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS'
        path = location+'\\'+str(model)+'\\3D_Model_GA_new'
        if variable == 'Temp':
            name=['']
            dict1 = pd.read_csv(path+str(0)+'_'+variable+'.txt', header=None, names=name)
        elif variable == 'Coord' or variable == 'Displacements' or variable =='Rotations':
            name=['X', 'Y', 'Z']
            dict1 = pd.read_csv(path+str(0)+'_'+variable+'.txt', delimiter=',', header=None, names=name)
        else:  #Stresses, Log Strains and Plastic strains
            name=['S11','S12', 'S13', 'S22']
            dict1 = pd.read_csv(path+str(0)+'_'+variable+'.txt', delimiter=',', header=None, names=name)
        #data = pd.DataFrame(dict1.items(), columns=[0], copy=True)
        data = {0: dict1}
        for frame in range(1,902,5):
            try:
                dict1 = pd.read_csv(path+str(frame)+'_'+variable+'.txt', header=None, names=name)
                data[frame] = dict1
            except IOError:
                break
        return data
    except FileNotFoundError: return None

def find_paint_strain(item):
    # find first occurance where temperature value is above a certain value
    try:
        LE = pd.DataFrame()
        PE = pd.DataFrame()
        stress_temp = pd.DataFrame()
        for L, P, T in zip(item[3], item[4], item[2]):
            LE = pd.concat([LE, item[3][L]], axis=1)
            PE = pd.concat([PE, item[4][P]], axis=1)
            stress_temp = pd.concat([stress_temp, item[2][T]], axis=1)
        location = (stress_temp.min().values > 250).argmax()
        paint_strain = ((LE['S11'].max()).iloc[location] + (PE['S11'].max()).iloc[location])*100+1.3
        return paint_strain
    except: []

```

```python
def Temp_time(dictionary):
    plt.figure()
    plt.figsize=(6.27,3.5)
    plt.subplots_adjust(left=0.14,bottom=0.14,right=0.96,top=0.92,hspace=0)
    plt.ylabel('T$_{FEM}$ [Celsius]')
    plt.xlabel('Time [min]')
    plt.grid(lw=0.3, which='major', axis='both')
    plt.grid(lw=0.1, which='minor', axis='both')
    plt.xlim(right=90, left =0)
    errorticks = [i for i,item in zip(range(30,100,5), range(40)) if item<(len(dictionary[0][1])/2)] # get list of differing tick spaces so lines don't overlap
    plt.ylim(bottom=0,top=800)
    counter=0
    for item in dictionary:
        try:
            if item[1] == None: continue
            else:
                if 'Column' in item[0]:
                    type = 'Columns'
                    item[0] = item[0].replace('Mech_Column_', '')
                    item[0] = item[0].replace('_Concrete','')
                elif 'Beam3' in item[0]:
                    type='3-sided Beams'
                    item[0] = item[0].replace('Mech_Beam3_', '')
                elif 'Beam1' in item[0]:
                    type = 'Integrated Beams'
                    item[0] = item[0].replace('Mech_Beam1_', '')
                if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                item[0] = item[0].replace('_', ' ')

                Label = item[0]
                data = pd.DataFrame()
                for j in item[1]:
                    data = pd.concat([data,item[1][j]],axis=1)
                y = data.mean()
                x = list(range(0,len(y), 1))
                x = [item / 2 for item in x]
                lowerlim = y - data.min()
                upperlim = data.max() - y
                limits=[lowerlim, upperlim]
                plt.errorbar(x,y,yerr=limits, label = Label, lw=0.8,
                    elinewidth=0.4, errorevery=errorticks[counter])
                counter+=1
        except TypeError: continue

    plt.title('Temperature - Time curve '+type)
    plt.minorticks_on()
    plt.legend(loc='lower right', fontsize=9, frameon=True, shadow=False, framealpha=0.5)
    plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'temp.png', dpi=400)
    plt.show()
    plt.close()

def Strain_time(dictionary):
    fig = plt.figure(figsize=(8,6))
    grid = fig.add_gridspec(nrows=2,ncols=2)
    fig.subplots_adjust(left=0.1,bottom=0.14,right=0.9,top=0.9, wspace=0.3, hspace=.5)
    strain = fig.add_subplot(grid[0])
    deflect = fig.add_subplot(grid[1])
    stress = fig.add_subplot(grid[2])

    strain.grid(lw=0.3, which='major', axis='both')
    strain.grid(lw=0.1, which='minor', axis='both')
    deflect.grid(lw=0.3, which='major', axis='both')
    deflect.grid(lw=0.1, which='minor', axis='both')
    stress.grid(lw=0.3, which='major', axis='both')
```

```python
    stress.grid(lw=0.1, which='minor', axis='both')

    for item in dictionary:
        try:
            if item[1] == None: continue
            else:
                if 'Column' in item[0]:
                    type = 'Columns'
                    item[0] = item[0].replace('Mech_Column_', '')
                    item[0] = item[0].replace('_Concrete','')
                elif 'Beam3' in item[0]:
                    type='3-sided Beams'
                    item[0] = item[0].replace('Mech_Beam3_', '')
                elif 'Beam1' in item[0]:
                    type = 'Integrated Beams'
                    item[0] = item[0].replace('Mech_Beam1_', '')
                if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                item[0] = item[0].replace('_', ' ')

                Label= item[0]
                LE = pd.DataFrame()
                PE = pd.DataFrame()
                disp = pd.DataFrame()
                Strs = pd.DataFrame()
                for L, P, D, S in zip(item[1], item[2],item[3], item[4]):
                    LE= pd.concat([LE,item[1][L]], axis=1)
                    PE= pd.concat([PE,item[2][P]], axis=1)
                    disp = pd.concat([disp, item[3][D]], axis=1)
                    Strs = pd.concat([Strs, item[4][S]], axis=1)

                y1 = LE['S11'].max()+PE['S11'].max()
                y2 = disp['Y'].max()
                y3 = Strs['S11'].max()
                x= list(range(0,len(y1),1))
                x= [i / 2 for i in x]
                strain.plot(x,y1, label=Label, lw=0.8)
                deflect.plot(x,y2, label=Label, lw=0.8)
                stress.plot(x,y3, label=Label, lw=0.8)
        except TypeError: continue

    strain.set(ylabel=r'$\epsilon_{FEM}$ ', xlabel='Time [min]',
        title='Strain - Time curve '+type, xlim=(0,90))
    deflect.set(ylabel='u$_{FEM}$ [mm]', xlabel='Time [min]',
        title='Deflection - Time curve '+type, xlim=(0,90))
    stress.set(ylabel= r'$\sigma_{FEM}$ [MPa]', xlabel='Time [min]',
        title='Stress - Time curve '+type, xlim=(0,90))
    deflect.minorticks_on()
    strain.minorticks_on()
    stress.minorticks_on()

    handles,labels = stress.get_legend_handles_labels()
    legend = fig.add_subplot(grid[3])
    legend.axis('off')
    legend.legend(handles,labels, loc='center left', fontsize=9, frameon=True, shadow=False,
                      framealpha =0.5)

    plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'stress-disp-strain.png', dpi=400)
    plt.show()
    plt.close()

def stress_strain(dictionary):
    fig = plt.figure(figsize=(6.27,3.5))
    fig.subplots_adjust(left=0.10,bottom=0.14,right=0.94,top=0.9, wspace=0)
    strain = fig.add_subplot()
    strain.grid(lw=0.3, which='major', axis='both')
    strain.grid(lw=0.1, which='minor', axis='both')
```

```python

    for item in dictionary:
        try:
            if item[1] == None: continue
            else:
                if 'Column' in item[0]:
                    type = 'Columns'
                    item[0] = item[0].replace('Mech_Column_', '')
                    item[0] = item[0].replace('_Concrete','')
                elif 'Beam3' in item[0]:
                    type='3-sided Beams'
                    item[0] = item[0].replace('Mech_Beam3_', '')
                elif 'Beam1' in item[0]:
                    type = 'Integrated Beams'
                    item[0] = item[0].replace('Mech_Beam1_', '')
                if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                item[0] = item[0].replace('_', ' ')

                Label= item[0]
                LE = pd.DataFrame()
                PE = pd.DataFrame()
                Strs = pd.DataFrame()
                Temp = pd.DataFrame()
                proof_stress = list()
                for L, P, S, T in zip(item[1], item[2],item[3], item[4]):
                    LE= pd.concat([LE,item[1][L]], axis=1)
                    PE= pd.concat([PE,item[2][P]], axis=1)
                    Strs = pd.concat([Strs, item[3][S]], axis=1)
                    Temp = pd.concat([Temp, item[4][T]], axis=1)
                for T in Temp.max():
                    if 'Alu' in Label:
                        if T<175: proof_stress.append(((120-205)/(175-20))*T+205)
                        elif T<200: proof_stress.append(((110-120)/25)*T+(120+70))
                        elif T<225: proof_stress.append(((100-110)/25)*T+(110+80))
                        elif T<250: proof_stress.append( ((88-100)/25)*T+(100+108))
                        elif T<275: proof_stress.append( ((75-88)/25)*T+(88+130))
                        elif T<300: proof_stress.append( ((60-75)/25)*T+(75+165))
                        elif T<325: proof_stress.append( ((46-60)/25)*T+(60+168))
                        elif T<350: proof_stress.append( ((34-46)/25)*T+(46+156))
                        elif T<450: proof_stress.append( ((1-34)/100)*T+(34+462))
                        else: proof_stress.append(0)
                    else:
                        if T<200: proof_stress.append(((203.9-800)/180)*T+800)
                        elif T<300: proof_stress.append(((137.6-203.9)/100)*T+(203.9+132.6))
                        elif T<400: proof_stress.append(((112.5-137.6)/100)*T+(137.6+75.3))
                        elif T<500: proof_stress.append(((89.3-112.5)/100)*T+(112.5+92.8))
                        elif T<600: proof_stress.append(((47.5-89.3)/100)*T+(89.3+209))
                        elif T<800: proof_stress.append(((0.1-47.5)/200)*T+(47.5+142.2))
                        else: proof_stress.append(0)
                y=list()
                #print(proof_stress)
                #print(Strs['S11'].max())
                #print(len(proof_stress),len(Strs['S11'].max()), len(Temp.max()))
                for s1, s2 in zip(Strs['S11'].max(), proof_stress):
                    try: y.append(s1/s2)
                    except : y.append(0)
                x = LE['S11'].max() + PE['S11'].max()
                strain.plot(x,y, label=Label, lw=0.8)
        except TypeError: continue

    strain.set(ylabel=r'$\sigma_{0.2\theta}$ / $\sigma_{FEM}$ ', xlabel=r'$\epsilon_{FEM}$',
        title='Stress - Strain curve '+type, )
```

```
    strain.minorticks_on()
    strain.legend(loc='upper right', fontsize=9, frameon=True, shadow=False, framealpha=0.5)

    plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'stress-strain.png', dpi=400)
    plt.show()
    plt.close()

def strain_and_rate(dictionary):
    fig = plt.figure(figsize=(6.27,3.5))
    grid = fig.add_gridspec(nrows=1,ncols=2)
    fig.subplots_adjust(left=0.1,bottom=0.14,right=0.98,top=0.9, wspace=0.3, hspace=.5)
    strain = fig.add_subplot(grid[0])
    strain_rate = fig.add_subplot(grid[1])

    strain.grid(lw=0.3, which='major', axis='both')
    strain.grid(lw=0.1, which='minor', axis='both')
    strain_rate.grid(lw=0.3, which='major', axis='both')
    strain_rate.grid(lw=0.1, which='minor', axis='both')

    # if 'paint' in dictionary[0][0]:
        # strain.plot([0,90],[1.3,1.3], 'k-.', label='Paint strain limit', lw=0.6) # need to add strain at 120 degrees celsius
    strain.plot([0,90],[3.75,3.75], 'k-.', label='Limit value', lw=0.6)
    strain_rate.plot([0,90],[1.7,1.7], 'k-.', lw=0.6)
    linenumber = 1
    for item in dictionary:
        try:
            if item[1] == None: continue
            else:
                if 'Column' in item[0]:
                    type = 'Columns'
                    item[0] = item[0].replace('Mech_Column_', '')
                    item[0] = item[0].replace('_Concrete','')
                elif 'Beam3' in item[0]:
                    type='3-sided Beams'
                    item[0] = item[0].replace('Mech_Beam3_', '')
                elif 'Beam1' in item[0]:
                    type = 'Integrated Beams'
                    item[0] = item[0].replace('Mech_Beam1_', '')
                if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                item[0] = item[0].replace('_', ' ')

                Label= item[0]
                LE = pd.DataFrame()
                PE = pd.DataFrame()
                for L, P, in zip(item[1], item[2]):
                    LE= pd.concat([LE,item[1][L]], axis=1)
                    PE= pd.concat([PE,item[2][P]], axis=1)

                y1 = LE['S11'].max()+PE['S11'].max() #logarithmic true strains
                x= list(range(0,len(y1),1))
                x= [i / 2 for i in x]
                #y2 = LE['S11'].max()) # LE strains and plastic strains
                y3 = np.zeros(y1.shape, np.float)
                y3[0:-1] = np.diff(y1)/np.diff(x)# derivative of a fitted polyline to y2
                y3[-1] = (y1[-1] - y1[-2]) / (x[-1] * x[-2])
                strain.plot(x,y1*100, label=Label, lw=0.8)
                Color = strain.get_lines()[linenumber].get_color()
                #strain.plot(x,y2, '--', color = Color, lw=0.8)
                del x[-1]
                strain_rate.plot(x,y3*100, color = Color, label=Label, lw=0.8)
                linenumber+=1
        except TypeError: continue

```

```python
    if 'type' in locals():
        strain.set(ylabel=r'$\epsilon_{FEM}$ $\u2030$ ', xlabel='Time [min]',
            xlim=(0,90))
        strain_rate.set(ylabel=r'$\delta\epsilon_{FEM}$/$\delta$t', xlabel='Time [min]',
            xlim=(0,90))
        strain_rate.set_title('Strain rate '+type, fontsize=9)
        strain.set_title('True Strain '+type, fontsize=9)
        if type=='3-sided Beams':
            strain.set_ylim(0,5)
            strain_rate.set_ylim(0,2)
    else: Label=''

    strain.minorticks_on()
    strain_rate.minorticks_on()
    strain_rate.plot([],[],'k--', lw=0.6, label='True Logarithmic strain')
    strain_rate.legend(loc='best', fontsize=6, frameon=True, shadow=False,
                      framealpha =0.5)

    plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'strain_rate.png', dpi=400)
    plt.show()
    plt.close()

def stress_strain_normalised(dictionary):
    fig = plt.figure(figsize=(6.27,3.5))
    fig.subplots_adjust(left=0.10,bottom=0.14,right=0.94,top=0.9, wspace=0.3)
    grid = fig.add_gridspec(nrows=1, ncols=2)
    strain_stress = fig.add_subplot(grid[0])
    strain_normalised = fig.add_subplot(grid[1])

    strain_stress.grid(lw=0.3, which='major', axis='both')
    strain_stress.grid(lw=0.1, which='minor', axis='both')
    strain_normalised.grid(lw=0.3, which='major', axis='both')
    strain_normalised.grid(lw=0.1, which='minor', axis='both')

    # if 'paint' in dictionary[0][0]:
        # strain.plot([1.3,1.3],[0,3], 'k-.', label='Paint strain limit', lw=0.6) # need to add strain at 120 degrees celsius
    #strain_stress.plot([3.75,3.75],[0,3], 'k-.', label='Strain limit', lw=0.6)
    #strain_normalised.plot([3.75,3.75],[0,1], 'k-.', label='Strain limit', lw=0.6)
    for item in dictionary:
        try:
            if item[1] == None: continue
            else:
                if 'Column' in item[0]:
                    type = 'Columns'
                    item[0] = item[0].replace('Mech_Column_', '')
                    item[0] = item[0].replace('_Concrete','')
                elif 'Beam3' in item[0]:
                    type='3-sided Beams'
                    item[0] = item[0].replace('Mech_Beam3_', '')
                elif 'Beam1' in item[0]:
                    type = 'Integrated Beams'
                    item[0] = item[0].replace('Mech_Beam1_', '')
                if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                item[0] = item[0].replace('_', ' ')

                Label= item[0]
                LE = pd.DataFrame()
                PE = pd.DataFrame()
                Strs = pd.DataFrame()
                Temp = pd.DataFrame()
                proof_stress = list()
                for L, P, S, T in zip(item[1], item[2],item[3], item[4]):
                    LE= pd.concat([LE,item[1][L]], axis=1)
                    PE= pd.concat([PE,item[2][P]], axis=1)
                    Strs = pd.concat([Strs, item[3][S]], axis=1)
```

```
                    Temp = pd.concat([Temp, item[4][T]], axis=1)
                for T in Temp.max():
                    if 'Alu' in Label:
                        if T<175: proof_stress.append(((120-205)/(175-20))*T+205)
                        elif T<200: proof_stress.append(((110-120)/25)*T+(120+70))
                        elif T<225: proof_stress.append(((100-110)/25)*T+(110+80))
                        elif T<250: proof_stress.append( ((88-100)/25)*T+(100+108))
                        elif T<275: proof_stress.append( ((75-88)/25)*T+(88+130))
                        elif T<300: proof_stress.append( ((60-75)/25)*T+(75+165))
                        elif T<325: proof_stress.append( ((46-60)/25)*T+(60+168))
                        elif T<350: proof_stress.append( ((34-46)/25)*T+(46+156))
                        elif T<450: proof_stress.append( ((1-34)/100)*T+(34+462))
                        else: proof_stress.append(0)
                    else:
                        if T<200: proof_stress.append(((203.9-800)/180)*T+800)
                        elif T<300: proof_stress.append(((137.6-203.9)/100)*T+(203.9+132.6))
                        elif T<400: proof_stress.append(((112.5-137.6)/100)*T+(137.6+75.3))
                        elif T<500: proof_stress.append(((89.3-112.5)/100)*T+(112.5+92.8))
                        elif T<600: proof_stress.append(((47.5-89.3)/100)*T+(89.3+209))
                        elif T<800: proof_stress.append(((0.1-47.5)/200)*T+(47.5+142.2))
                        else: proof_stress.append(0)
                y = Strs['S11'].max()
                x = LE['S11'].max() + PE['S11'].max()
                strain_stress.plot(x*100,y, label=Label,lw=0.8)
                y = list()
                for s1, s2 in zip(Strs['S11'].max(), proof_stress):
                    try: y.append(s1/s2)
                    except : y.append(0)

                strain_normalised.plot(x*100,y, label=Label, lw=0.8)
        except TypeError: continue

    if 'type' in locals():
        strain_normalised.set(ylabel=r'$\sigma_{FEM}$ / $\sigma_{0.2\Theta}$', xlabel=r'$\epsilon_{FEM}$ $\u2030$',)
        strain_normalised.set_title('Normalised stress - strain curve '+type,fontsize=9)
        strain_stress.set(ylabel=r'$\sigma_{True,FEM}$ [MPa]', xlabel=r'$\epsilon_{FEM}$ $\u2030$')
        strain_stress.set_title('True stress - strain curve '+type, fontsize=9)
    else: Label=''
    strain_normalised.minorticks_on()
    strain_stress.minorticks_on()
    strain_normalised.legend(loc='best', fontsize=6, frameon=True, shadow=False, framealpha=0.5)

    plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'stress-strain-normalised.png', dpi=400)
    plt.show()
    plt.close()

def Stress_deflect(dictionary):
    fig = plt.figure(figsize=(6.27,3.5))
    grid = fig.add_gridspec(nrows=1,ncols=2)
    fig.subplots_adjust(left=0.14,bottom=0.14,right=0.98,top=0.9, wspace=0.3, hspace=.5)
    stress = fig.add_subplot(grid[1])
    deflect = fig.add_subplot(grid[0])

    stress.grid(lw=0.3, which='major', axis='both')
    stress.grid(lw=0.1, which='minor', axis='both')
    deflect.grid(lw=0.3, which='major', axis='both')
    deflect.grid(lw=0.1, which='minor', axis='both')

    path_Properties = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\properties'
    Stress_Alu = pd.read_csv(path_Properties+'\proof stress6060-T66 20-TRUE.csv', delimiter=';', header=None, names=['Stress', 'Strain', 'Temperature'],
```

```
dtype=np.float64)
Stress_Steel = pd.read_csv(path_Properties+'\\proof stressSteel EC3.csv',
delimiter=';', header=None, names=['Stress', 'Strain', 'Temperature'],
dtype=np.float64)

stress_alu = Stress_Alu.sort_values(by=['Strain', 'Stress'], ascending =False)
stress_alu = stress_alu.reset_index(drop=True)
stress_alu.loc[90]=[1,0,450]
stress.plot(stress_alu['Temperature'][81:91], stress_alu['Stress'][81:91],'k--',
label = r'$\sigma_{0.2}$ Aluminium', lw=0.8)
stress_steel = Stress_Steel.sort_values(by=['Strain','Temperature'])
stress_steel = stress_steel.reset_index(drop=True)
stress.plot(stress_steel['Temperature'][0:7], stress_steel['Stress'][0:7],
'k-.', label = r'$\sigma_{0.2}$ Steel', lw=0.8)

for item in dictionary:
    try:
        if item[1] == None: continue
        else:
            if 'Column' in item[0]:
                type = 'Columns'
                item[0] = item[0].replace('Mech_Column_', '')
                item[0] = item[0].replace('_Concrete','')
            elif 'Beam3' in item[0]:
                type='3-sided Beams'
                item[0] = item[0].replace('Mech_Beam3_', '')
            elif 'Beam1' in item[0]:
                type = 'Integrated Beams'
                item[0] = item[0].replace('Mech_Beam1_', '')
            if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
            elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
            if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
            if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', '
            IPE')
            item[0] = item[0].replace('_', ' ')

            Label= item[0]
            disp = pd.DataFrame()
            Strs = pd.DataFrame()
            Temp = pd.DataFrame()
            for D, S, T in zip(item[1], item[2], item[3]):
                disp = pd.concat([disp, item[1][D]], axis=1)
                Strs = pd.concat([Strs, item[2][S]], axis=1)
                Temp = pd.concat([Temp, item[3][T]], axis=1)
            if type=='Columns': y1=disp['Z'].max()
            else: y1 = disp['Y'].max()
            y2 = Strs['S11'].max()
            x = list(range(0,len(y1),1))
            x = [i / 2 for i in x]
            x2 = Temp.max()
            deflect.plot(x,y1, label=Label, lw=0.8)
            stress.plot(x2,y2, label=Label, lw=0.8)
    except TypeError: continue

deflect.set(ylabel='u$_{FEM}$ [mm]', xlabel='Time [min]',
    xlim=(0,90))
stress.set(ylabel= r'True $\sigma_{FEM}$ [MPa]', xlabel='T$_{member}$ [Celsius]',
    xlim=(0,700), ylim=(0,300))
if 'type' in locals():
    deflect.set_title('Deflection - Time curve '+type, fontsize=9)
    stress.set_title('Stress - Time curve '+type, fontsize=9)
else: Label=''
deflect.minorticks_on()
stress.minorticks_on()
stress.legend(loc='best', fontsize=6, frameon=True, shadow=False,
                  framealpha =0.5)

plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis
figures'+'\\'+Label+'stress-disp.png', dpi=400)
plt.show()
plt.close()
```

```
# create list with all model names and corresponding data
# possible variables ['Coord', 'Temp','Stresses','Displacements','Rotations','Log_strains','Plastic_strains']
column_list = ['Mech_Column_Steel_shell_RHS_Concrete_no_10',
               'Mech_Column_Steel_shell_RHS_Concrete_yes_10',
               'Mech_Column_Steel_shell_I-section_Concrete_no_10',
               'Mech_Column_Steel_shell_I-section_Concrete_yes_10',
               'Mech_Column_Aluminium_shell_RHS_Concrete_no_10',
               'Mech_Column_Aluminium_shell_RHS_Concrete_yes_10',
               'Mech_Column_Aluminium_shell_I-section_Concrete_no_10',
               'Mech_Column_Aluminium_shell_I-section_Concrete_yes_10',
               ]
beam3_ins_list = ['Mech_Beam3_Steel_shell_RHS_Concrete_yes_10',
                  'Mech_Beam3_Steel_shell_RHS_Lightweight_yes_10',
                  'Mech_Beam3_Steel_shell_I-section_Concrete_yes_10',
                  'Mech_Beam3_Steel_shell_I-section_Lightweight_yes_10',
                  'Mech_Beam3_Aluminium_shell_RHS_Concrete_yes_10',
                  'Mech_Beam3_Aluminium_shell_RHS_Lightweight_yes_10',
                  'Mech_Beam3_Aluminium_shell_I-section_Concrete_yes_10',
                  'Mech_Beam3_Aluminium_shell_I-section_Lightweight_yes_10',
                  ]
beam1_ins_list = ['Mech_Beam1_Aluminium_shell_RHS_Concrete_yes_10',
                  'Mech_Beam1_Aluminium_shell_RHS_Lightweight_yes_10',
                  'Mech_Beam1_Aluminium_shell_I-section_Lightweight_yes_10',
                  'Mech_Beam1_Aluminium_shell_I-section_concrete_yes_10',
                  'Mech_Beam1_Steel_shell_RHS_Lightweight_yes_10',
                  'Mech_Beam1_Steel_shell_RHS_Concrete_yes_10',
                  'Mech_Beam1_Steel_shell_I-section_Lightweight_yes_10',
                  'Mech_Beam1_Steel_shell_I-section_Concrete_yes_10',
                  ]

beam3_noins_list = list()
beam1_noins_list = list()
for K, Z in zip(beam3_ins_list, beam1_ins_list):
    beam3_noins_list.append( K.replace('yes', 'no'))
    beam1_noins_list.append( Z.replace('yes', 'no'))

def temps(myList, numframes):
    dictionary_Temp=list()
    for model in myList:
        Temp = read_data(numframes,model,'Temp', '')
        dictionary_Temp.append([model,Temp])
    Temp_time(dictionary_Temp)
def stresses(myList, numframes):
    dictionary_stress = list()
    for model in myList:
        LE = read_data(numframes,model,'Log_strains', '')
        PE = read_data(numframes,model,'Plastic_strains','')
        stress = read_data(numframes, model, 'Stresses','')
        Temp = read_data(numframes,model,'Temp','')
        dictionary_stress.append([model,LE,PE,stress,Temp])
    stress_strain(dictionary_stress)
def strains(myList, numframes):
    dictionary_strain =list()
    for model in myList:
        LE = read_data(numframes,model,'Log_strains','')
        PE = read_data(numframes,model,'Plastic_strains','')
        disp = read_data(numframes,model,'Displacements','')
        stress = read_data(numframes, model, 'Stresses','')
        dictionary_strain.append([model,LE,PE,disp,stress])
    Strain_time(dictionary_strain)

def strains2(myList, numframes):
    dictionary = list()
    for model in myList:
        LE = LE = read_data(numframes,model,'Log_strains','')
        PE = read_data(numframes,model,'Plastic_strains','')
        if 'four_point' in model: model = model[24:]
        dictionary.append([model,LE,PE])
    strain_and_rate(dictionary)
    del LE, PE
    dictionary.clear()
```

```
    for model in myList:
        LE = read_data(numframes,model,'Log_strains','')
        PE = read_data(numframes,model,'Plastic_strains','')
        stress = read_data(numframes, model, 'Stresses','')
        Temp = read_data(numframes,model,'Temp','')
        if 'four_point' in model: model = model[24:]
        dictionary.append([model,LE,PE,stress,Temp])
    stress_strain_normalised(dictionary)
    del LE, PE, stress, Temp
    dictionary.clear()
    for model in myList:
        disp = read_data(numframes,model,'Displacements','')
        stress = read_data(numframes, model, 'Stresses','')
        Temp = read_data(numframes,model,'Temp','')
        if 'four_point' in model: model = model[24:]
        dictionary.append([model,disp,stress, Temp])
    Stress_deflect(dictionary)

# Plot_Emod()
# Plot_conductivity()
# Plot_specific_heat()
# Plot_proofStress()

def mechanical_plotting_columns():
    strains(column_list, 901)
    stresses(column_list,901)
    #temps(column_list, 901)
def mechanical_plotting_beam3():
    strains(beam3_ins_list, 901)
    stresses(beam3_ins_list,901)
    #temps(beam3_ins_list, 901)
def mechanical_plotting_beam1():
    strains(beam1_ins_list, 901)
    stresses(beam1_ins_list,901)
    #temps(beam1_ins_list, 901)

def four_point_bending():
    #mechanical_plotting_columns()
    #mechanical_plotting_beam3()
    #mechanical_plotting_beam1()
    strains2(column_list, 901)
    counter=0
    for item in beam3_ins_list:
        beam3_ins_list[counter] = 'four_point_bending_test\\'+item
        counter+=1
    strains2(beam3_ins_list, 901)
    counter =0
    for item in beam1_ins_list:
        beam1_ins_list[counter] = 'four_point_bending_test\\'+item
        counter+=1
    strains2(beam1_ins_list, 901)
#four_point_bending()
def distributed_load():
    for item in column_list: item = 'Combined odb\\'+item
    for item in column_list: item = 'Combined odb\\'+item
    for item in column_list: item = 'Combined odb\\'+item
    mechanical_plotting_columns()
    mechanical_plotting_beam3()
    mechanical_plotting_beam1()
# distributed_load()

def thermal_expansion(dictionary):
    x_steel = list(range(20,1205,5))
    y_steel = list()
    y_aluminium =list()
    x_aluminium = list(range(20,505,5))
    for item in x_steel:
        if item-x_steel[0]==0: y_steel.append(0)
        else:
            if item<750:
            y_steel.append((1.2e-5*item+0.4e-8*item*item-2.416e-4)/(item-x_steel[0]))
            elif item<861: y_steel.append(1.1e-2/(item-x_steel[0]))
```

```python
                else: y_steel.append((2e-5*item-6.2e-3)/(item-x_steel[0]))
        for item in x_aluminium:
            if item-x_aluminium[0]==0: y_aluminium.append(0)
            else: y_aluminium.append((0.1e-7*item*item+22.5e-6*item-4.5e-4)/(item-x_aluminium[0]))
        if 'Column' in dictionary[0][0]: L=1e3
        else: L=3e3

        fig = plt.figure(figsize=(8,4))
        grid = fig.add_gridspec(nrows=1,ncols=2)
        fig.subplots_adjust(left=0.1,bottom=0.14,right=0.9,top=0.9, wspace=0.5, hspace=.5)
        expand = fig.add_subplot(grid[0])
        displace = expand.twinx() #fig.add_subplot(grid[1], sharex=expand)
        left = fig.add_subplot(grid[1])
        left2 = left.twinx()

        expand.grid(lw=0.3, which='major', axis='both')
        expand.grid(lw=0.1, which='minor', axis='both')
        left.grid(lw=0.3, which='major', axis='both')
        left.grid(lw=0.1, which='minor', axis='both')

        expand.plot(x_steel,y_steel, 'k--', label=r'$\alpha_{L}$', lw=0.6)
        left.plot(x_aluminium, y_aluminium,'k--', label=r'$\alpha_{L}$',  lw=0.6)

        # thermal expansion coefficient #
        path_Properties = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\properties'
        with open(path_Properties+'\\'+'ThermalExpAlu EC9.csv','r') as f:
            reader=(csv.reader(f, delimiter=';'))
            Expansion_alu = ()
            x_alu = list()
            y_alu = list()
            for row, column in enumerate(reader):
                v=[]
                for value in column:
                    v=v+[float(value),]
                Expansion_alu = Expansion_alu + (v,)
            f.close()
            for item in Expansion_alu:
                x_alu.append(item[3])
                y_alu.append(item[0])
        with open(path_Properties+'\\'+'ThermalExpSteel EC3.csv', 'r') as f:
            reader=(csv.reader(f, delimiter=';'))
            Expansion_steel=()
            x_ste = list()
            y_ste = list()
            for row, column in enumerate(reader):
                v=[]
                for value in column:
                    v=v+[float(value),]
                Expansion_steel = Expansion_steel + (v,)
            f.close()
            for item in Expansion_steel:
                x_ste.append(item[3])
                y_ste.append(item[0])
        disp_steel = list()
        disp_alu = list()
        for item, temp in zip(y_ste,x_ste):
            if temp-x_ste[0]>0: disp_steel.append(item*(temp-x_ste[0]) * L)
            else: disp_steel.append(0)
        for item, temp in zip(y_alu, x_alu):
            if temp-x_alu[0]>0: disp_alu.append(item*(temp-x_alu[0])*L)
            else: disp_alu.append(0)
        left.plot(x_alu,y_alu,'r--', lw=0.6, label=r'$\alpha_{L,FEM}$')
        expand.plot(x_ste,y_ste,'r--', lw=0.6, label=r'$\alpha_{L,FEM}$')
        displace.plot(x_ste, disp_steel, label='Theoretical', lw=2)
        left2.plot(x_alu, disp_alu, label='Theoretical', lw=2)

        for item in dictionary:
            try:
                if item[1] == None: continue
                else:
```

```python
                    if 'Column' in item[0]:
                        type = 'Columns'
                        item[0] = item[0].replace('Mech_Column_', '')
                        item[0] = item[0].replace('_Concrete','')
                    elif 'Beam3' in item[0]:
                        type='3-sided Beams'
                        item[0] = item[0].replace('Mech_Beam3_', '')
                    elif 'Beam1' in item[0]:
                        type = 'Integrated Beams'
                        item[0] = item[0].replace('Mech_Beam1_', '')
                    if 'yes' in item[0]: item[0]=item[0].replace('_yes_10', ' insulated')
                    elif 'no' in item[0]: item[0]=item[0].replace('_no_10',' uninsulated')
                    if 'shell' in item[0]: item[0]=item[0].replace('_shell', '')
                    if 'I-section' in item[0]: item[0]=item[0].replace('_I-section', ' IPE')
                    item[0] = item[0].replace('_', ' ')

                    Label= item[0]
                    disp = pd.DataFrame()
                    Temp = pd.DataFrame()
                    for D, T in zip(item[1], item[2]):
                        disp = pd.concat([disp, item[1][D]], axis=1)
                        Temp = pd.concat([Temp, item[2][T]], axis=1)

                    y = disp['Z'].max()
                    x = Temp.max()
                    if 'Steel' in Label:
                        Label = Label.replace('Steel ', '')
                        displace.plot(x,y, label=Label,lw=0.8)
                    else:
                        Label = Label.replace('Aluminium ','')
                        left2.plot(x,y, label=Label, lw=0.8)

            except TypeError: continue

        fig.suptitle('Thermal expansion '+type)
        expand.set(title='Steel', ylabel=r'$\alpha_{L}$ [K$^{-1}$]',ylim=(0,1.6e-5), xlabel='T$_{MAX}$ [Celsius]',xlim=(0,1000))
        displace.set(ylabel= 'Displacement [mm]',ylim=(0,16))
        left2.set(ylabel= 'Displacement [mm]',ylim=(0,16))
        left.set(title='Aluminium', ylabel=r'$\alpha_{L}$ [K$^{-1}$]', ylim=(0,3e-5),xlabel='T$_{MAX}$ [Celsius]',xlim=(0,500))

        start1, end1 = expand.get_ylim()
        expand.yaxis.set_ticks(np.arange(start1,end1,end1/5))
        left.yaxis.set_ticks(np.arange(start1,3e-5,(3e-5)/5))
        #expand.yaxis.set_major_formatter(ticker.FormatStrFormatter('%0.1f')) # set major ticks
        expand.ticklabel_format(axis='y', style='sci', scilimits=(-5,-5))
        left.ticklabel_format(axis='y', style='sci', scilimits=(-5,-5))

        start2, end2 = displace.get_ylim()
        displace.yaxis.set_ticks(np.arange(start2,end2,end2/5))
        left2.yaxis.set_ticks(np.arange(start2,end2,end2/5))
        displace.minorticks_on()
        expand.minorticks_on()
        left.minorticks_on()
        left2.minorticks_on()

        handles1,labels1 = expand.get_legend_handles_labels()
        handles2,labels2 = displace.get_legend_handles_labels()
        for item in handles2: handles1.append(item)
        for item in labels2: labels1.append(item)
        expand.legend(handles1,labels1,loc='best', fontsize=7, frameon=True, shadow=False,
                          framealpha =0.5)
        left.legend(handles1,labels1,loc='best', fontsize=7, frameon=True, shadow=False,
                          framealpha =0.5)

        plt.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+Label+'thermal_expand.png', dpi=400)
        plt.show()
        plt.close()
```

```python

# dictionary=list()
# for model in column_list:
    # disp = read_data(901,model,'Displacements')
    # Temp = read_data(901,model,'Temp')
    # if 'four_point' in model: model = model[24:]
    # dictionary.append([model,disp, Temp])
# thermal_expansion(dictionary)

def beams_validation(dictionary,loading):
    q_load_steel = 0.2*100 #N/mm 100 is width of I-section
    q_load_aluminium = 0.2*100
    P_concrete = 49.5e6
    P_lightweight = 36e6
    P_alternate = 20e6

    # material properties
    path_Properties = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\properties'
    Emod_Alu = pd.read_csv(path_Properties+'\\EmodAlu 6060-T66 18.csv', delimiter=';', header=None, names=['fraction', 'Temperature', 'Emod'], dtype=np.float64, decimal=',')
    stress_alu = pd.read_csv(path_Properties+'\\proofstressAlu.csv', delimiter=';', header=None, names=['Temperature', '0.2', 'yield'], dtype=np.float64, decimal=',')
    Emod_Steel = pd.read_csv(path_Properties+'\\EmodSteel.csv', delimiter=';', header=None, names=['Temperature','Emod'], dtype=np.float64, decimal=',')
    stress_steel = pd.read_csv(path_Properties+'\\proofstressSteel.csv', delimiter=';', header=None, names=['Temperature','yield', '0.2', 'hi'], dtype=np.float64, decimal=',')
    Expansion_Alu = pd.read_csv(path_Properties+'\\ThermalExpAlu EC9.csv', delimiter=';', header=None, names=['Alpha', 'beta', 'c', 'Temperature'], dtype=np.float64)
    Expansion_Ste = pd.read_csv(path_Properties+'\\ThermalExpSteel EC3.csv', delimiter=';', header=None, names=['Alpha', 'beta', 'c','Temperature'], dtype=np.float64)

    # figure plotting stresses
    fig = plt.figure(figsize=(8,4))
    grid = fig.add_gridspec(nrows=1,ncols=2)
    fig.subplots_adjust(left=0.1,bottom=0.14,right=0.96,top=0.87, wspace=0.22, hspace=.5)
    steel = fig.add_subplot(grid[0])
    aluminium = fig.add_subplot(grid[1])

    steel.grid(lw=0.3, which='major', axis='both')
    steel.grid(lw=0.1, which='minor', axis='both')
    aluminium.grid(lw=0.3, which='major', axis='both')
    aluminium.grid(lw=0.1, which='minor', axis='both')

    #second image strains
    fig2 = plt.figure(figsize=(8,4))
    grid2 = fig2.add_gridspec(nrows=1, ncols=2)
    fig2.subplots_adjust(left=0.1,bottom=0.14,right=0.96,top=0.87, wspace=0.22,hspace=0.5)
    steel2 = fig2.add_subplot(grid2[0])
    aluminium2 = fig2.add_subplot(grid2[1])

    steel2.grid(lw=0.3, which='major', axis='both')
    steel2.grid(lw=0.1, which='minor', axis='both')
    aluminium2.grid(lw=0.3, which='major', axis='both')
    aluminium2.grid(lw=0.1, which='minor', axis='both')

    #fourth image displacement
    fig4 = plt.figure(figsize=(8,4))
    grid4 = fig4.add_gridspec(nrows=1,ncols=2)
    fig4.subplots_adjust(left=0.1,bottom=0.14,right=0.96,top=0.87, wspace=0.22,hspace=0.5)
    steel4 = fig4.add_subplot(grid4[0])
    aluminium4 = fig4.add_subplot(grid4[1])

    steel4.grid(lw=0.3, which='major', axis='both')
    steel4.grid(lw=0.1, which='minor', axis='both')
    aluminium4.grid(lw=0.3, which='major', axis='both')
```

```
        aluminium4.grid(lw=0.1, which='minor', axis='both')

        # plot stress-temp curve
        steel.plot(stress_steel['Temperature'], stress_steel['yield'], 'k--', label='Yield Stress', lw=0.8)
        aluminium.plot(stress_alu['Temperature'], stress_alu['yield'], 'k--', label='Yield Stress', lw=0.8)
        steel.plot(stress_steel['Temperature'], stress_steel['0.2'], 'k-.', label='Proportional Stress', lw=0.8)
        aluminium.plot(stress_alu['Temperature'], stress_alu['0.2'], 'k-.', label='Proof Stress', lw=0.8)

        # plot strain limits
        steel2.plot([0,1000],[3.75,3.75], 'k--', label = 'Ultimate strain limit', lw=0.8)
        aluminium2.plot([0,1000],[3.75,3.75], 'k--', label= 'Ultimate strain limit', lw=0.8)
        steel2.plot([0,1000],[2.,2.], 'k-.', label = 'Yield limit', lw=0.8)
        #aluminium2.plot([0,1000],[2.,2.], 'k-.', label='Yield limit', lw=0.8)
        # plot deflection limit
        if 'Column' in dictionary[0][0]: deflect = 1000**2 / (400 * 200)
        else: deflect = 3000**2 / (400*200)
        steel4.plot([0,1000],[deflect,deflect],'k--', label = 'Deflection limit', lw=0.8)
        aluminium4.plot([0,1000],[deflect,deflect],'k--', label = 'Deflection limit', lw=0.8)

        for item in dictionary:
            try:
                # theoretical stress
                if loading=='q_load':
                    if 'Steel' in item[0]: M=q_load_steel
                    else: M=q_load_aluminium
                elif loading =='Lowered':
                    if 'oncrete' in item[0]: M=P_lightweight
                    else: M=P_alternate
                else:
                    if 'oncrete' in item[0]: M=P_concrete
                    else: M=P_lightweight
                if 'Steel' in item[0]:
                    Emod = Emod_Steel['Emod']
                    Temp = Emod_Steel['Temperature']
                    t,tf,tw,b,h = [6.0,8.5,5.6,100.,200.]
                elif 'Aluminium' in item[0]:
                    Emod = Emod_Alu['Emod']
                    Temp = Emod_Alu['Temperature']
                    t,tf,tw,b,h = [9.0,9.0,5.,100.,200.]
                if 'RHS' in item[0]: I= 1/12*(b*2)*h**3 - 1/12*((b*2)-2*t)*(h-2*t)**3#mm4
                elif 'I-section' in item[0] or 'IPE' in item[0]: I= 1/12*b*(tf**3) + b*tf*(1/2*h-1/2*tf)**2 + 1/12*tw*((h-2*tf)**3) + 1/12*b*(tf**3) + b*tf*(1/2*h-1/2*tf)**2 #mm4
                stress=list()
                stress_temp = list()
                z=1/2*h #mm
                counter = 0
                if 'Column' in item[0]: L=1e3
                else: L=3e3
                if loading =='q_load': M = 1/8*M*L**2
                else: M = M*1/2*L - M*1/6*L
                for E in Emod:
                    stress.append( M*z/I)
                    stress_temp.append(Temp[counter])
                    counter+=1
                if item[1] == None: continue
                else:
                    if 'Column' in item[0]:
                        type = 'Columns'
                        Label = item[0].replace('Mech_Column_', '')
                        Label = Label.replace('_Concrete','')
                    elif 'Beam3' in item[0]:
                        type='3-sided Beams'
                        Label = item[0].replace('Mech_Beam3_', '')
                    elif 'Beam1' in item[0]:
                        type = 'Integrated Beams'
```

```python
            Label = item[0].replace('Mech_Beam1_', '')
        if 'yes' in item[0]: Label=Label.replace('_yes_10', ' insulated')
        elif 'no' in item[0]: Label=Label.replace('_no_10',' uninsulated')
        if 'shell' in item[0]: Label=Label.replace('_shell', '')
        if 'I-section' in item[0]: Label=Label.replace('_I-section', ' IPE')
        Label = Label.replace('_', ' ')
        if 'Steel' in Label:
            Label = Label.replace('Steel ', '')
            steel.plot(stress_temp,stress, '--', label='Theoretical',lw=0.8) #plot theoretical stress
            Label = Label.replace(' ', 'Steel')
            Color = steel.get_lines()[-1].get_color()

        else:
            Label = Label.replace('Aluminium ','')
            aluminium.plot(stress_temp, stress, '--', label='Theoretical', lw=0.8) #plot theoretical stress
            Label = Label.replace(' ','Aluminium')
            Color = aluminium.get_lines()[-1].get_color()

        # plot abaqus stress-temp curve (has 3 directions)
        print('stress')
        stress = pd.DataFrame()
        stress_temp = pd.DataFrame()
        for S, T in zip(item[1], item[2]):
            stress = pd.concat([stress, item[1][S]], axis=1)
            stress_temp = pd.concat([stress_temp, item[2][T]], axis=1)

        if 'Column' in item[0]:
            loc_middle = 334
            loc_top = 2
            loc_bottom = 7
        else:
            loc_middle = 6938
            loc_top = 163
            loc_bottom = 1112
        middle_node = abs(stress['S11'].loc[loc_middle])
        top_node =  abs(stress['S11'].loc[loc_top])
        bottom_node = abs(stress['S11'].loc[loc_bottom])
        print(middle_node,top_node,bottom_node)

        x_top = stress_temp.loc[loc_middle]
        x_bottom = stress_temp.loc[loc_bottom]
        x_middle = stress_temp.loc[loc_middle]

        if 'Steel' in Label:
            Label = Label.replace('Steel', ' ')
            #steel.plot(x_middle,middle_node, label=Label+ ' centroid', lw=0.8, alpha=0.5)#color = Color,
            #steel.plot(x_top,top_node, label=Label+ ' top centre', lw=0.8, alpha=0.5)#color = Color,
            steel.plot(x_bottom,bottom_node, color = Color, label=Label+ ' bottom centre', lw=0.8, alpha=0.5)#color = Color,

        else:
            Label = Label.replace('Aluminium',' ')
            #aluminium.plot(x_middle,middle_node, label=Label+ ' centroid', lw=0.8, alpha=0.5)#color = Color,
            #aluminium.plot(x_top,top_node, label=Label+ ' top centre', lw=0.8, alpha=0.5)#color = Color,
            aluminium.plot(x_bottom,bottom_node, color=Color,  label=Label+ ' bottom centre', lw=0.8, alpha=0.5)#color = Color,

        # strain plotting
        print('strain plotting')
        # try: paint_strain = find_paint_strain(item)
        # except:pass

        if 'Steel' in item[0]:
            input_stress = stress_steel
            input_emod = Emod_Steel
        else:
```

```
    input_stress = stress_alu
    input_emod = Emod_Alu

input_stress.sort_values(['Temperature'], ascending=True, inplace=True)
input_stress = input_stress.loc[input_stress['Temperature'].isin(input_emod['Temperature'])].reset_index(drop=True)
input_emod = input_emod.loc[input_emod['Temperature'].isin(input_stress['Temperature'])]

temp = input_stress['Temperature']
if 'Steel' in item[0]: proofstress = input_stress['yield']
else: proofstress = input_stress['0.2']
emod = input_emod['Emod'].reset_index(drop=True)
disp_proof = np.argwhere( (M*z/I) > input_stress['0.2'] )
disp_yield = np.argwhere( (M*z/I) > input_stress['yield'] )

if 'Steel' in item[0]: expansion = Expansion_Ste
else: expansion = Expansion_Alu
alpha = copy.deepcopy(expansion['Alpha'].loc[expansion['Temperature'].isin(temp)].reset_index(drop=True))
temp = copy.deepcopy(temp.loc[temp.isin(expansion['Temperature'])].reset_index(drop=True))

strain_total = list()
deltaT = []
deflect_total = list()
elastic_strain_all = list()
for item2 in temp:
    location = np.argwhere(stress_temp.mean() >item2)
    deltaT.append(0)
    if len(location) == 0:
        try: deltaT[-1] = (stress_temp.max().iloc[-1] - stress_temp.min().iloc[-1])
        except: pass
    else: deltaT[-1] = (stress_temp.max().iloc[location[0]] - stress_temp.min().iloc[location[0]])

for i in range(0,len(temp),1):
    n = proofstress[i] / 10 # reference 13
    if type!='Columns': L = 3e3
    else: L=1e3
    try: temp_diff = deltaT[i]
    except: temp_diff=0
    if item[6]=='expansion':
        elastic_strain =0
        elastic_disp = 0
        bowing_strain = alpha[i]*(temp[i]-temp[0]) + alpha[i] * temp_diff / 4
        bowing_disp = ((alpha[i] * temp_diff * (L**2)) / (8*h))
        Label2 = 'Theoretical Thermal bowing'
    elif item[6]=='elastic':
        elastic_strain = (M*z/I) / emod[i] + 2e-3 * ((M*z/I) / proofstress[i])**n
        elastic_disp = (5/384) * ((M*8/(L**2))*(L**4))/(emod[i]*I)
        bowing_strain = 0
        bowing_disp = 0

        if (M*z/I)>proofstress[i]:
            print('proofstress has been exceeded')
        Label2 = 'Theoretical Mechanical only'
    else:
        elastic_strain = (M*z/I) / emod[i] + 2e-3 * ((M*z/I) / proofstress[i])**n
        elastic_disp = (5/384) * ((M*8/(L**2))*(L**4))/(emod[i]*I)
        bowing_strain =  alpha[i]*(temp[i]-temp[0])+ alpha[i] * temp_diff / 4
        bowing_disp = ((alpha[i] * temp_diff * (L**2)) / (8*h))
```

```
                if (M*z/I)>proofstress[i]:
                    print('proofstress has been exceeded')
                Label2 = 'Theoretical total'
            elastic_strain_all.append(elastic_strain)
            strain_total.append(   (elastic_strain + bowing_strain)*100 )
            deflect_total.append( elastic_disp + bowing_disp )
            if elastic_strain>(4/100): break

        if 'Steel' in item[0]:
            steel2.plot(temp[0:len(strain_total)],strain_total, '--', label = Label2, lw=0.8)
            Color = steel2.get_lines()[-1].get_color()
            steel4.plot(temp[0:len(deflect_total)],deflect_total, '--',color = Color, label = Label2, lw=0.8)
        else:
            aluminium2.plot(temp[0:len(strain_total)],strain_total, '--', label =Label2, lw=0.8)
            Color = aluminium2.get_lines()[-1].get_color()
            aluminium4.plot(temp[0:len(deflect_total)],deflect_total,  '--', color=Color,  label =Label2, lw=0.8)

        LE = pd.DataFrame()
        PE = pd.DataFrame()
        disp = pd.DataFrame()
        for L, P, D in zip(item[3], item[4], item[5]):
            LE = pd.concat([LE, item[3][L]], axis=1)
            PE = pd.concat([PE, item[4][P]], axis=1)
            disp = pd.concat([disp, item[5][D]], axis=1)

        if 'Column' in item[0]:
            loc_middle = 334
            loc_top = 2
            loc_bottom = 7
        else:
            loc_middle = 6938
            loc_top = 163
            loc_bottom = 1112
        middle_node = abs(LE['S11'].loc[loc_middle]) + abs(PE['S11'].loc[loc_middle])
        top_node =  abs(LE['S11'].loc[loc_top]) + abs(PE['S11'].loc[loc_top])
        bottom_node = abs(LE['S11'].loc[loc_bottom]) + abs(PE['S11'].loc[loc_bottom])

        x_top = stress_temp.loc[loc_middle]
        x_bottom = stress_temp.loc[loc_bottom]
        x_middle = stress_temp.loc[loc_middle]

        if 'Steel' in item[0]:
            #steel2.plot(x_middle,middle_node*100, label=Label+ ' middle', lw=0.8, alpha=0.5)
            #steel2.plot(x_top,top_node*100, label=Label+ ' top', lw=0.8, alpha=0.5)
            steel2.plot(x_bottom,bottom_node*100, color=Color, label=Label+ ' bottom', lw=0.8, alpha=0.5)

        else:
            #aluminium2.plot(x_middle,middle_node*100, label=Label+ ' middle', lw=0.8, alpha=0.5)
            #aluminium2.plot(x_top,top_node*100,  label=Label+ ' top', lw=0.8, alpha=0.5)
            aluminium2.plot(x_bottom,bottom_node*100, color=Color, label=Label+ ' bottom', lw=0.8, alpha=0.5)

        # deflection
        if 'Column' in item[0]:
            loc_middle = 334
            loc_top = 2
            loc_bottom = 7
            disp = disp['Z']
        else:
            loc_middle = 6938
            loc_top = 163
```

```python
                    loc_bottom = 1112
                    disp= disp['Y']

                middle_node = abs(disp.loc[loc_middle])
                top_node =  abs(disp.loc[loc_top])
                bottom_node = abs(disp.loc[loc_bottom])
                print(middle_node)

                x_top = stress_temp.loc[loc_middle]
                x_bottom = stress_temp.loc[loc_bottom]
                x_middle = stress_temp.loc[loc_middle]

                if item[6]=='expansion':
                    itera = 0
                    for Q in bottom_node:
                        if Q>20: break
                        itera+=1

                    middle_node = middle_node[0:itera]
                    top_node =  top_node[0:itera]
                    bottom_node = bottom_node[0:itera]

                    x_top = x_top[0:len(top_node)]
                    x_bottom = x_bottom[0:len(bottom_node)]
                    x_middle = x_middle[0:len(middle_node)]

                first_yield = input_stress['Temperature'].loc[disp_proof[0][0]]
                ultimate_yield = input_stress['Temperature'].loc[disp_yield[0][0]]
                print(first_yield,ultimate_yield)

                if 'Steel' in item[0]:
                    #steel4.plot(x_middle,middle_node, label=Label+ ' middle', lw=0.8, alpha=0.5)
                    #steel4.plot(x_top,top_node, label=Label+ ' top', lw=0.8, alpha=0.5)
                    steel4.plot(x_bottom,bottom_node, color=Color,label=Label+ ' bottom', lw=0.8, alpha=0.5)

                    if item[6]!='expansion':
                        steel4.plot([first_yield,first_yield],[0,200], 'k-.', label = 'First yield', lw=0.8)
                        steel4.plot([ultimate_yield,ultimate_yield],[0,200], 'k-.', label = 'Ultimate yield', lw=0.8)

                else:
                    #aluminium4.plot(x_middle,middle_node, label=Label+ ' middle', lw=0.8, alpha=0.5)
                    #aluminium4.plot(x_top,top_node,  label=Label+ ' top', lw=0.8, alpha=0.5)
                    aluminium4.plot(x_bottom,bottom_node, color=Color, label=Label+ ' bottom', lw=0.8, alpha=0.5)

                    if item[6]!='expansion':
                        aluminium4.plot([220,220],[0,200], 'k-.', label = 'First yield', lw=0.8)
                        aluminium4.plot([280,280],[0,200], 'k-.', label = 'Ultimate yield', lw=0.8)

        except TypeError: continue
    if 'type' in locals(): pass
    else: type =''

    fig.suptitle('Stress - Temperature curves '+type)
    steel.set(title='Steel', ylabel='\u03C3  [MPa]', xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,300))
    aluminium.set(title='Aluminium', ylabel='\u03C3  [MPa]', xlabel='Temperature [Celsius]',xlim=(0,550),ylim=(0,300))

    steel.minorticks_on()
    aluminium.minorticks_on()
    steel.legend(loc='best', fontsize=7, frameon=True, shadow=False,framealpha =0.5)
    aluminium.legend(loc='best', fontsize=7, frameon=True, shadow=False,
```

```
        framealpha=0.5)

        fig2.suptitle('Strain - Temperature curves '+type)
        steel2.set(title='Steel', ylabel= '$\epsilon$ [\u2030]', xlabel='Temperature [Celsius]',xlim=(0,800), ylim=(0,4))
        aluminium2.set(title='Aluminium', ylabel= '$\epsilon$ [\u2030]', xlabel='Temperature [Celsius]',xlim=(0,550), ylim=(0,4))
        steel2.minorticks_on()
        aluminium2.minorticks_on()
        steel2.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha=0.5)
        aluminium2.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha=0.5)

        fig4.suptitle('Displacement - Temperature curves '+type)
        steel4.set(title='Steel', ylabel='Deflection [mm]', xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,120))
        aluminium4.set(title='Aluminium', ylabel='Delfection [mm]', xlabel='Temperature [Celsius]',xlim=(0,550),ylim=(0,120))
        steel4.minorticks_on()
        aluminium4.minorticks_on()
        steel4.legend(loc='upper left', fontsize=7, frameon=True, shadow=False, framealpha=0.5)
        aluminium4.legend(loc='upper left', fontsize=7, frameon=True, shadow=False, framealpha=0.5)

        fig.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+'Stress - Temperature curves Elastic '+type+'.png', dpi=400)
        fig2.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+'Strain - Temperature curves Elastic '+type+'.png', dpi=400)
        fig4.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+'Deflect = Temperature curves Elastic '+type+'.png', dpi=400)

        plt.show()
        plt.close()

    def checker():
        myList = ['Mech_Beam3_Aluminium_shell_I-section_Concrete_no_10','Mech_Beam3_Steel_shell_I-section_Concrete_no_10']
        dictionary=list()
        for item in myList:
            print(item)
            check = 'expansion'
            path = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\Validation\noloading'
            item1 = item
            stress=read_data(901,item1,'Stresses',path)
            temp = read_data(901, item1, 'Temp',path)
            LE = read_data(901, item1, 'Log_strains',path)
            PE = read_data(901, item1, 'Plastic_strains',path)
            disp = read_data(901,item1,'Displacements',path)
            dictionary.append([item1, stress, temp, LE, PE, disp,check])
        #beams_validation(dictionary,'q_load')
        myList = ['Mech_Beam3_Aluminium_shell_I-section_Concrete_yes_10','Mech_Beam3_Steel_shell_I-section_Concrete_yes_10']

        for item in myList:
            print(item)
            item1=item
            path= r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\Validation\noexp'
            check= 'elastic'
            stress=read_data(901,item1,'Stresses',path)
            temp = read_data(901, item1, 'Temp',path)
            LE = read_data(901, item1, 'Log_strains',path)
            PE = read_data(901, item1, 'Plastic_strains',path)
            disp = read_data(901,item1,'Displacements',path)
            dictionary.append([item1,stress,temp,LE,PE,disp,check])
        #beams_validation(dictionary,'q_load')
        for item in myList:
            print(item)
```

```
        item1 =item
        path= r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\Validation\norm'
        check = ''
        stress=read_data(901,item1,'Stresses',path)
        temp = read_data(901, item1, 'Temp',path)
        LE = read_data(901, item1, 'Log_strains',path)
        PE = read_data(901, item1, 'Plastic_strains',path)
        disp = read_data(901,item1,'Displacements',path)
        dictionary.append([item1,stress,temp,LE,PE,disp,check])
    loading = 'q_load'
    beams_validation(dictionary, loading)

#checker()

def postprocessing(dictionary, name):
    # material properties
    path_Properties = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\properties'
    stress_alu = pd.read_csv(path_Properties+'\\proofstressAlu.csv', delimiter=';', header=None, names=['Temperature', '0.2', 'yield'], dtype=np.float64, decimal=',')
    stress_steel = pd.read_csv(path_Properties+'\\proofstressSteel.csv', delimiter=';', header=None, names=['Temperature','yield', '0.2', 'hi'], dtype=np.float64, decimal=',')

    # figure plotting stresses
    fig = plt.figure(figsize=(8,8))
    grid = fig.add_gridspec(nrows=3,ncols=2)
    fig.subplots_adjust(left=0.08,bottom=0.08,right=0.92,top=0.9, wspace=0.45, hspace=.5)
    plot11 = fig.add_subplot(grid[0])   #stress-time
    plot111 = plot11.twinx()            #utilization-time
    plot12 = fig.add_subplot(grid[1])   #temp-time
    plot21 = fig.add_subplot(grid[2])   #strain-time
    plot31 = fig.add_subplot(grid[4])   #deflect-time
    plot311 = plot31.twinx()            #deflection rate - time

    # image print
    item_counter = 0
    for item in dictionary:
        if item[1]==None: continue
        else: item_counter+=1

    if item_counter==0:
        plot22 = fig.add_subplot(grid[3])   #legend
        plot32 = fig.add_subplot(grid[5])
    elif item_counter>0 and item_counter<5:
        #plot22 = fig.add_subplot(grid[3])  #legend
        grid2 = grid[1:,-1].subgridspec(item_counter+1,1)
    elif item_counter>4 and item_counter<7:
        #plot22 = fig.add_subplot(grid[3])  #legend
        grid2 = grid[1:,-1].subgridspec(item_counter/2+1,2)
    elif item_counter>6:
        grid2 = grid[1:,-1].subgridspec(5,2)

    for plots in [plot11,plot111,plot12,plot21,plot31,plot311]:
        plots.grid(lw=0.3, which='major', axis='both')
        plots.grid(lw=0.1, which='minor', axis='both')

    # plot strain limits
    plot21.plot([0,800],[3.75,3.75], 'k--', label = 'Ultimate strain limit', lw=0.5)
    if 'no' in dictionary[1][0]:
        plot21.text(12,3.5, '$\epsilon_{ultimate,limit}$', fontsize=9)
    else:
        plot21.text(60,3.5, '$\epsilon_{ultimate,limit}$', fontsize=9)
    # plot deflection limit
    if 'Column' in dictionary[0][0]: deflect = 1000**2 / (400 * 200)
    else: deflect = 3000**2 / (400*200)
    plot31.plot([0,800],[deflect,deflect],'k--', label = 'Deflection limit', lw=0.5 )
    if 'no' in dictionary[1][0]:
        plot31.text(12,deflect, 'Deflection limit', fontsize=7)
    else: plot31.text(60,deflect, 'Deflection limit', fontsize=7)
    # Deflection rate limit
```

```python
        if 'Column' in dictionary[0][0]: deflect_rate = 1000**2 / (9000*200)
        else: deflect_rate = 3000**2 / (9000*200)
        plot311.plot([0,800], [deflect_rate,deflect_rate], 'k-.', label='Deflection rate limit', lw=0.5)
        if 'no' in dictionary[1][0]:
            plot311.text(12,deflect_rate, 'Deflection rate limit', fontsize=7)
        else: plot311.text(55,deflect_rate, 'Deflection rate limit', fontsize=7)

        item_counter = 0
        for item in dictionary:
            try:
                if item[1] == None: continue #if empty dataset don't run it
                else:
                    print(item[0])
                    # create a label
                    if 'Column' in item[0]:
                        type = 'Columns'
                        Label = item[0].replace('Mech_Column_', '')
                        Label = Label.replace('_Concrete','')
                        length = 1e3
                    elif 'Beam3' in item[0]:
                        type='3-sided Beams'
                        Label = item[0].replace('Mech_Beam3_', '')
                        length=3e3
                    elif 'Beam1' in item[0]:
                        type = 'Integrated Beams'
                        Label = item[0].replace('Mech_Beam1_', '')
                        length=3e3
                    if 'yes' in item[0]: Label=Label.replace('_yes_10', ' insulated')
                    elif 'no' in item[0]: Label=Label.replace('_no_10',' uninsulated')
                    if 'shell' in item[0]: Label=Label.replace('_shell', '')
                    if 'I-section' in item[0]: Label=Label.replace('_I-section', ' IPE')
                    if 'Beam' in item[0] and 'uninsulated' in Label:
                        Label=Label.replace(' uninsulated', '')
                        type = type + ' uninsulated'

                    Label = str(item_counter+1)+' '+Label.replace('_', ' ')

                    #plot temperature - time curve
                    print('Temperature plotting')
                    stress_temp = pd.DataFrame()
                    for T in item[2]:
                        stress_temp = pd.concat([stress_temp, item[2][T]], axis=1)

                    y = stress_temp.mean()
                    lowerlim = stress_temp.min()
                    upperlim = stress_temp.max()

                    if 'Aluminium' in item[0]:
                        itera = 0
                        for Q in y:
                            if Q>500:break
                            itera+=1
                    elif 'Steel' in item[0]:
                        itera=0
                        for Q in y:
                            if Q>1200: break
                            itera+=1

                    y= y[0:itera]
                    lowerlim = lowerlim[0:itera]
                    upperlim = upperlim[0:itera]
                    x= [i / 2. for i in list(range(0,len(y),1))]

                    plot12.plot(x,y, label=Label, lw=0.8)
                    Color = plot12.get_lines()[-1].get_color()
                    plot12.plot(x, lowerlim, color=Color,lw=0.6, alpha=0.5)
                    plot12.plot(x,upperlim, color=Color, lw=0.6, alpha=0.5)
                    plot12.fill_between(x,upperlim,lowerlim, color=Color, alpha=0.05)

                    # plot abaqus stress-temp curve (has 3 directions) S11 is in the length direction
```

```python
print('Stress plotting')
stress = pd.DataFrame()
for S in item[1]:
    stress = pd.concat([stress, item[1][S]], axis=1)

if 'Column' in item[0]:
    if 'I-section' in item[0] or 'IPE' in item[0]:
        loc_middle = 334
        loc_top = 2
        loc_bottom = 7
    else:
        loc_middle = 252
        loc_top = 386
        loc_bottom = 18
else:
    if 'I-section' in item[0] or 'IPE' in item[0]:
        loc_middle = 6938
        loc_top = 163
        loc_bottom = 1112
    else:
        if 'Beam3' in item[0]:
            loc_list = [9422,14804,4049,1145,810,176,493]
        else: loc_list = [10625,16008,7495,1751,1416,1060,162]
        loc_middle, loc_top, loc_bottom, edge_top1, edge_top2, edge_bottom1, edge_bottom2 = loc_list

middle_node = abs(stress['S11'].loc[loc_middle])
top_node =  abs(stress['S11'].loc[loc_top])
bottom_node = abs(stress['S11'].loc[loc_bottom])
temp_bottom = abs(stress_temp.loc[loc_bottom])

bottom_node = bottom_node[0:itera]
temp_bottom = temp_bottom[0:itera]

y1 = bottom_node

# if 'Column' in item[0]: y1=stress['S11'].max()
# else:
#    datarange = len(stress['S11'])/300
#    start = 0.5*(len(stress['S11'])-datarange)
#    end = 0.5*(len(stress['S11'])+datarange)
#    y1 = stress['S11'].loc[start:end].max() #stress at midspan

plot11.plot(x,y1, color=Color, label=Label, lw=0.8, alpha =0.5)

# utilization
print('utilization')
if 'Steel' in item[0]:
    input_stress = stress_steel
else:
    input_stress = stress_alu

input_stress.sort_values(['Temperature'], ascending=True, inplace=True)
temp = input_stress['Temperature'].reset_index(drop=True)
if 'Steel' in item[0]:
    proofstress = input_stress['yield'].reset_index(drop=True)
else: proofstress = input_stress['0.2'].reset_index(drop=True)

utilization = []
time = []
counter = 0
for T in temp:
    try:
        location = np.argwhere(temp_bottom>T)
        loc = location[0]
        time.append( x[loc[0]] )
        utilization.append(y1.iloc[loc[0]]/proofstress.iloc[counter])
        counter+=1
    except: continue

plot111.plot(time,utilization, '--', color = Color, label=Label,
```

```
        lw=0.8, alpha=0.5)

        # strain plotting
        print('strain plotting')
        #try:
        #   paint_strain = find_paint_strain(item)
        #   plot21.plot([0,800],[paint_strain, paint_strain], '-.', label='Paint limit', alpha=0.3,lw=0.8)
        #except:pass

        LE = pd.DataFrame()
        PE = pd.DataFrame()
        for L, P in zip(item[3], item[4]):
            LE = pd.concat([LE, item[3][L]], axis=1)
            PE = pd.concat([PE, item[4][P]], axis=1)

        middle_node = abs(LE['S11'].loc[loc_middle] + PE['S11'].loc[loc_middle])
        top_node =  abs(LE['S11'].loc[loc_top] + PE['S11'].loc[loc_top])
        bottom_node = abs(LE['S11'].loc[loc_bottom] + PE['S11'].loc[loc_bottom])

        bottom_node = bottom_node[0:itera]
        y=bottom_node

        # if 'Column' in item[0]:
        #   y = LE['S11'].max()+PE['S11'].max()
        # else:
        #   datarange = len(LE['S11'])/300
        #   start = 0.5*(len(LE['S11'])-datarange)
        #   end = 0.5*(len(LE['S11'])+datarange)
        #   LE1 = copy.deepcopy(LE['S11'].loc[start:end])
        #   PE1 = copy.deepcopy(PE['S11'].loc[start:end])
        #   y1 = LE1.max()+PE1.max()
        #   y = abs(LE1.mean()+PE1.mean())

        if len(x)>len(y): del x[-1]
        if len(y)>len(x): del y[-1]

        plot21.plot(x,y*100, color=Color, label=Label, lw=0.8, alpha=0.5)

        # deflection
        print('deflection plotting')
        disp = pd.DataFrame()
        for D in item[5]:
            disp = pd.concat([disp, item[5][D]], axis=1)

        if 'Column' in item[0]:
            disp = disp['Z']
        else:
            disp = disp['Y']

        middle_node = abs(disp.loc[loc_middle])
        top_node =  abs(disp.loc[loc_top])
        bottom_node = abs(disp.loc[loc_bottom])

        bottom_node = bottom_node[0:itera]
        y=bottom_node

        plot31.plot(x,y, label=Label, color=Color, lw=0.8, alpha=0.5)

        # deflection rate
        print('deflection rate')
        deflect_rate = []
        datarange = np.argwhere(y> (length/30))
        print(datarange)
        if len(datarange)==0: datarange=[[0]]
        for i in range(datarange[0][0],len(y),1):
            deflect_rate.append( (y.iloc[i]-y.iloc[i-1]) / (x[i]-x[i-1]) )
        deflect_x = x[datarange[0][0]:]
        plot311.plot(deflect_x,deflect_rate, '--', color=Color, label=Label, lw=0.8, alpha=0.5)
```

```python

            # image print
            print('image print')
            img = item[6]
            counter = 0
            for item2 in grid2:
                counter+=1
            print(counter)

            if 'plot32' in locals():
                plot32.imshow(img)
            else:
                if counter>5:
                    row = int(item_counter/2+1)
                    column = item_counter % 2
                    print(row,column, item_counter)
                    imager = fig.add_subplot(grid2[item_counter+2])
                else:
                    imager = fig.add_subplot(grid2[item_counter+1])
                imager.imshow(img)
                imager.axis('off')
                imager.text(0.9, 0.1,str(item_counter+1), ha='center', va='center', transform=imager.transAxes, fontsize=7)

            item_counter+=1

    except TypeError: continue
if 'type' in locals(): pass
else: type =''

# manual legend entries
lstyle = ['-', '--']
lines = [Line2D([0],[0], color='k', lw=0.6, ls=style) for style in lstyle]
labels = ['Stress', 'Utilization']
plot11.legend(lines,labels, loc='upper right', fontsize=7, frameon=False)
labels = ['Deflection', 'Deflection rate']
plot31.legend(lines,labels, loc='best', fontsize=7, frameon=False)

fig.suptitle(type)
plot11.set(title='Stress - Time curve', ylabel='\u03C3  [MPa]',xlabel='Time [min]',xlim=(0,90),ylim=(0,250))    #stress-time
plot111.set(ylabel='Utilization', ylim=(0,2.5))              #utilization-time
plot12.set(title='Temperature - Time curve', ylabel='Temperature [Celsius]', xlabel='Time [min]',xlim=(0,90),ylim=(0,1000))      #stress-time    #temp-time

plot21.set(title='Strain - Time curve', ylabel='\u03B5 [\u2030]', xlabel='Time [min]',xlim=(0,90), ylim=(0,4))  #strain-time
handles,labels = plot11.get_legend_handles_labels()
if 'plot22' in locals():
    plot22.axis('off')
    plot22.legend(handles,labels, loc='center left', fontsize=7, frameon=True, shadow=False, framealpha =0.5)
    plot32.set_title('Deformed shape vs Initial state')
else:
    if item_counter<5:
        plot22 = fig.add_subplot(grid2[0,:])
        plot22.axis('off')
        plot22.legend(handles,labels, loc='upper center', fontsize=7, frameon=True, shadow=False, framealpha =0.5)
    else:
        plot22 = fig.add_subplot(grid2[0,:])
        plot22.axis('off')
        plot22.legend(handles,labels, loc='upper left', fontsize=6, frameon=True, shadow=False, framealpha =0.5, ncol=2)
    if 'Aluminium' in dictionary[0][0]:loc_y=-590
    else: loc_y=-990
    if 'no' in dictionary[1][0]: loc_x = 2
    else: loc_x=5
    plot12.text(loc_x,loc_y, 'Deformed shape vs Initial state', fontsize=11)

plot31.set(title='Deflection - Time curve',ylabel='Deflection [mm]', xlabel='Time [min]', xlim=(0,90), ylim=(0,125))     #deflect-time
```

```
    plot311.set(ylabel='Deflection rate [mm/min]', ylim=(0,6.25)) #deflection rate - time
    if 'no' in dictionary[1][0]:
        plot11.set_xlim(0,20)
        plot111.set_xlim(0,20)
        plot12.set_xlim(0,20)
        plot21.set_xlim(0,20)
        plot31.set_xlim(0,20)
        plot311.set_xlim(0,20)
    if 'Aluminium' in dictionary[0][0]:
        plot12.set_ylim(0,600)
    if 'Column' in dictionary[0][0]:
        plot11.set_ylim(0,40)
        plot111.set_ylim(0,4)
        plot21.set_ylim(0,4)
        plot31.set_ylim(0,30)
        plot311.set_ylim(0,3)

    for plots in [plot11,plot111,plot12,plot21,plot31,plot311]:
        plots.minorticks_on()
        plots.minorticks_on()

    fig.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+'Post_processing '+name+'.png', dpi=400)
    #plt.show()
    plt.close()

beam3_ins_list_alu = []
beam3_ins_list_ste = []
beam1_ins_list_alu =[]
beam1_ins_list_ste =[]
beam3_noins_list_alu = []
beam3_noins_list_ste = []
beam1_noins_list_alu =[]
beam1_noins_list_ste =[]
# list names column_list, beam3_ins_list, beam1_ins_list, beam3_noins_list, beam1_noins_list all with 8 items

for Q, W, E, R in zip(beam3_ins_list, beam3_noins_list, beam1_ins_list, beam1_noins_list):
    if 'Alu' in Q: beam3_ins_list_alu.append(Q)
    elif 'Steel' in Q: beam3_ins_list_ste.append(Q)
    if 'Alu' in W: beam3_noins_list_alu.append(W)
    elif 'Steel' in W: beam3_noins_list_ste.append(W)
    if 'Alu' in E: beam1_ins_list_alu.append(E)
    elif 'Steel' in E: beam1_ins_list_ste.append(E)
    if 'Alu' in R: beam1_noins_list_alu.append(R)
    elif 'Steel' in R: beam1_noins_list_ste.append(R)
# list names column_list, beam3_ins_list, beam1_ins_list, beam3_noins_list, beam1_noins_list all with 8 items
def checkers(myList, path, image_name):
    dictionary=list()
    for item in myList:
        stress=read_data(901,item,'Stresses', path)
        temp = read_data(901, item, 'Temp', path)
        LE = read_data(901, item, 'Log_strains', path)
        PE = read_data(901, item, 'Plastic_strains', path)
        disp = read_data(901,item,'Displacements', path)
        try: img = mpimg.imread(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures\mechanical image'+'\\'+copy.deepcopy(item)+image_name+'.png')
        except: img=''
        dictionary.append([item, stress, temp, LE, PE, disp, img])
    name= myList[0] +'_'+ image_name
    postprocessing(dictionary, name)

checkers(column_list, r'E:', '')
# checkers(beam3_ins_list_alu, r'E:\four_point_bending_test',  'Pload')
# checkers(beam3_ins_list_ste, r'E:\four_point_bending_test',  'Pload' )
# checkers(beam3_noins_list, r'E:\four_point_bending_test',  'Pload')
# checkers(beam1_ins_list_alu, r'E:\four_point_bending_test',  'Pload')
# checkers(beam1_ins_list_ste, r'E:\four_point_bending_test',  'Pload')
```

```python
# checkers(beam1_noins_list, r'E:\four_point_bending_test',  'Pload' )

# location = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\paint\P'
# checkers(column_list, location, 'paintP')
# checkers(beam3_ins_list, location,  'paintP')

# location = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\paint\Q'
# checkers(column_list, location, 'paintQ')
# checkers(beam3_ins_list, location,  'paintQ')

# location = r'E:\q_load 12-9-2019'
# checkers(beam3_ins_list_alu, location,  'Qload')
# checkers(beam3_ins_list_ste, location,  'Qload')
# checkers(beam3_noins_list, location,  'Qload')
# checkers(beam1_ins_list_alu, location,  'Qload')
# checkers(beam1_ins_list_ste, location,  'Qload')
# checkers(beam1_noins_list, location,  'Qload')

# location = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\Alt\Q'
# checkers(beam3_ins_list, location, 'AltQ')
# checkers(beam1_ins_list, location, 'AltQ')

# location = r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\ABAQUS\Alt\P'
# checkers(beam3_ins_list, location, 'AltP')
# checkers(beam1_ins_list, location, 'AltP')

# beam3_ins_list_alu
# beam3_ins_list_ste
# beam1_ins_list_alu
# beam1_ins_list_ste
# beam3_noins_list_alu
# beam3_noins_list_ste
# beam1_noins_list_alu
# beam1_noins_list_ste
column_list_no =list()
column_list_yes = list()
for item in column_list:
    if 'yes' in item:
        column_list_yes.append(item)
    else:
        column_list_no.append(item)
# column_list,
# beam3_ins_list,
# beam1_ins_list,
# beam3_noins_list,
# beam1_noins_list

def column_beams(dictionary, name):
    # figure plotting stresses
    fig = plt.figure(figsize=(8,8))
    grid = fig.add_gridspec(nrows=2,ncols=2)
    fig.subplots_adjust(left=0.08,bottom=0.08,right=0.92,top=0.9, wspace=0.45, hspace=.5)
    plot11 = fig.add_subplot(grid[0])
    plot12 = fig.add_subplot(grid[1])
    plot21 = fig.add_subplot(grid[2])
    plot22 = fig.add_subplot(grid[3])

    for plots in [plot11,plot12,plot21,plot22]:
        plots.grid(lw=0.3, which='major', axis='both')
        plots.grid(lw=0.1, which='minor', axis='both')

    item_counter = 0

    for item in dictionary:
        try:
            if item[1] == None: continue #if empty dataset don't run it
            else:
                print(item[0])
                # create a label
                if 'Column' in item[0]:
                    type = 'Columns'
```

```
    Label = item[0].replace('Mech_Column_', '')
    Label = Label.replace('_Concrete','')
    length = 1e3
elif 'Beam3' in item[0]:
    type='3-sided Beams'
    Label = item[0].replace('Mech_Beam3_', '')
    length=3e3
elif 'Beam1' in item[0]:
    type = 'Integrated Beams'
    Label = item[0].replace('Mech_Beam1_', '')
    length=3e3
if 'yes' in item[0]: Label=Label.replace('_yes_10', ' insulated')
elif 'no' in item[0]: Label=Label.replace('_no_10',' uninsulated')
if 'shell' in item[0]: Label=Label.replace('_shell', '')
if 'I-section' in item[0]: Label=Label.replace('_I-section', ' IPE')
if 'Beam' in item[0] and 'uninsulated' in Label:
    Label=Label.replace(' uninsulated', '')
    type = type + ' uninsulated'

Label = Label.replace('_', ' ')

if 'Column' in item[0]:
    if 'I-section' in item[0] or 'IPE' in item[0]:
        loc_middle = 334
        loc_top = 2
        loc_bottom = 7
    else:
        loc_middle = 252
        loc_top = 386
        loc_bottom = 18
else:
    if 'I-section' in item[0] or 'IPE' in item[0]:
        loc_middle = 6938
        loc_top = 163
        loc_bottom = 1112
    else:
        if 'Beam3' in item[0]:
            loc_list = [9422,14804,4049,1145,810,176,493]
        else: loc_list = [10625,16008,7495,1751,1416,1060,162]
        loc_middle, loc_top, loc_bottom, edge_top1, edge_top2, edge_bottom1, edge_bottom2 = loc_list

stress_temp = pd.DataFrame()
for T in item[1]:
    stress_temp = pd.concat([stress_temp, item[1][T]], axis=1)

LE = pd.DataFrame()
PE = pd.DataFrame()
for L, P in zip(item[2], item[3]):
    LE = pd.concat([LE, item[2][L]], axis=1)
    PE = pd.concat([PE, item[3][P]], axis=1)

middle_node = abs(LE['S11'].loc[loc_middle] + PE['S11'].loc[loc_middle])
top_node =  abs(LE['S11'].loc[loc_top] + PE['S11'].loc[loc_top])
bottom_node = abs(LE['S11'].loc[loc_bottom] + PE['S11'].loc[loc_bottom])
bottom_temp = stress_temp.loc[loc_bottom]

if 'Aluminium' in item[0]:
    itera = 0
    for Q in bottom_temp:
        if Q>500:break
        itera+=1
elif 'Steel' in item[0]:
    itera=0
    for Q in bottom_temp:
        if Q>1200: break
        itera+=1

y = bottom_node[0:itera]
x = bottom_temp[0:itera]
```

```python

                if 'Steel' in item[0] and 'no' in item[0]:
                    plot11.plot(x,y*100, label=Label, lw=0.8, alpha=0.8)
                elif 'Steel' in item[0] and 'yes' in item[0]:
                    plot21.plot(x,y*100, label=Label, lw=0.8, alpha=0.8)
                elif 'Aluminium' in item[0] and 'no' in item[0]:
                    plot12.plot(x,y*100, label=Label, lw=0.8, alpha=0.8)
                elif 'Aluminium' in item[0] and 'yes' in item[0]:
                    plot22.plot(x,y*100, label=Label, lw=0.8, alpha=0.8)

        except TypeError: continue
        item_counter+=1

    if 'type' in locals(): pass
    else: type =''

    fig.suptitle(type)
    plot11.set(title='Uninsulated Steel', ylabel='\u03B5 [\u2030]',xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,2))
    plot12.set(title='Uninsulated Aluminium', ylabel='\u03B5 [\u2030]',xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,2))
    plot21.set(title='Insulated Steel', ylabel='\u03B5 [\u2030]',xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,2))
    plot22.set(title='Insulated Aluminium', ylabel='\u03B5 [\u2030]',xlabel='Temperature [Celsius]',xlim=(0,800),ylim=(0,2))

    plot11.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha =0.5)
    plot12.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha =0.5)
    plot21.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha =0.5)
    plot22.legend(loc='best', fontsize=7, frameon=True, shadow=False, framealpha =0.5)

    for plots in [plot11,plot12,plot21, plot22]:
        plots.minorticks_on()
        plots.minorticks_on()

    fig.savefig(r'D:\renee\OneDrive - TU Eindhoven\Studie\Afstuderen\Thesis figures'+'\\'+'column_beam_'+name+'.png', dpi=400)
    plt.show()
    plt.close()

def checkerss(myList, path1, path2, image_name):
    dictionary=list()
    for item in myList:
        if 'Column' in item:
            temp = read_data(901, item, 'Temp', path1)
            LE = read_data(901, item, 'Log_strains', path1)
            PE = read_data(901, item, 'Plastic_strains', path1)
        else:
            temp = read_data(901, item, 'Temp', path2)
            LE = read_data(901, item, 'Log_strains', path2)
            PE = read_data(901, item, 'Plastic_strains', path2)
        dictionary.append([item, temp, LE, PE])
    name = myList[0] +'_'+ image_name
    column_beams(dictionary, name)

# newlist3 = column_list + beam3_ins_list + beam3_noins_list
# checkerss(newlist3, r'E:',  r'E:\four_point_bending_test',  'Pload')
# newlist1 = column_list + beam1_ins_list + beam1_noins_list
# checkerss(newlist1, r'E:',  r'E:\four_point_bending_test',  'Pload')

# location = r'E:\q_load 12-9-2019'
# checkerss(newlist1, r'E:', location, 'Qload')
# checkerss(newlist3, r'E:', location, 'Qload')

```