Martí Torrijos Musach

Bachelor's thesis

Post-tension concrete beam modelling and analysis in DIANA via Python

June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Martí Torrijos Musach

Post-tension concrete beam modelling and analysis in DIANA via Python

Bachelor's thesis
June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

Post-tensioned systems are becoming more popular over time. They permit building long span and slender bridges by compressing the concrete to the point that the effect of the external loads is nearly counteracted.
Studying them is a really difficult and tedious job, making it nearly impossible to do without software. Moreover, sometimes the procedure has some defects, and assessing the level of post-tensioning or the state of the tendons becomes an impossible job.

For those reasons, different Finite Element Software programs are being used. They permit analysing the state of the materials when external loads are applied. This includes displacements, stresses on the concrete and its reinforcement, the cracks on the concrete, the tension of the tendons, and other parameters that are of crucial importance when it comes to the structural element durability.

In this specific project, the program that is going to be used is DIANA FEA, a powerful software that enables the analysis of a wide range of structures, including reinforced and prestressed concrete. Furthermore, a programming language called Python will be used in order to ease the process of modelling the structure that is going to be studied and also to save some time.

NOTE:
There are two kinds of prestressing: pre-tensioning and post-tensioning. For that reason, when prestressing is mentioned in this project, it directly refers to post-tensioning.

Table of Contents

List of Figures

List of Tables

List of Equations

1. Introduction

The main purpose of this project is learning how to use a Finite Element Software called DIANA FEA to model different case scenarios of a reinforced concrete beam and using the programming language Python to do exactly the same models but automating the process. This will allow saving time for the users that have to do a big number of simulations with slight differences between their models.

Learning how to use the software includes the procedures followed to do the 3D modelling, with passive and active reinforcement, defining the properties of the used materials, the restriction of movements, the application of the loads and finally the obtention and interpretation of the results.

The models that will be done are:

1) Reinforced beam with no post-tensioning
2) Reinforced beam with post-tensioning
3) Reinforced beam with post-tensioning and lack of grouting

These three models will allow having a wide vision on how the software works and will permit studying the effects and behavioural changes that post-tensioning[1], as explained in *Post-Tensioning - Methods for Reinforcing Concrete* (Palmer, 2010) (1), has on a structural element such as a beam.

Moreover, the different cases will introduce many difficulties that will have to be dealt with and this will come in really handy when getting familiarized with the software

Python is well-known for its simplicity and helpfulness to automate the procedure if different case scenarios have to be studied. Considering that, this thesis will also try to capture the use of the programming language and help the people who want to continue studying and researching this topic to automate, simplify and speed up all the procedures followed. This will allow them to study a wide range of scenarios while minimizing the amount of time invested. The scripts written to model the different case scenarios will be included in this project and thoroughly explained.

Apart from the goals explained previously, in this thesis, it will also be really important to compare the performances between a case of a beam without post-tensioning and one with.

To end with, the most difficult things when modelling the beam and running the analysis will be explained so that the people who want to run a similar analysis can do so without having to do more research on the internet or in other sources.

NOTE: It is of most importance to say that the geometry and reinforcement have been chosen in an arbitrary way, that is to say they do not follow any code or rule. The goal of this thesis is to model a beam, whichever the dimensions and the reinforcement, and apply load until failure.

Later, doing the same but with a Python script.

The materials, dimensions and other features have been chosen by the author of this document, meaning that it is really likely that the reinforcement amount is not right, among other errors that cannot be ignored when it comes to real cases.

[1] Post tensioning is a technique for reinforcing concrete. Post-tensioning tendons, which are prestressing steel cables inside plastic ducts or sleeves, are positioned in the forms before the concrete is placed. Afterwards, once the concrete has gained strength but before the service loads are applied, the cables are pulled tight, or tensioned, and anchored against the outer edges of the concrete.

2. Modelling of a beam without post-tensioning

The first thing that should be done when modelling a real beam is defining the properties of the materials that will be used, which will be done in section *2.1 Properties of the used materials*. These properties appear in section *3 of the Design of concrete structures in Eurocode 2* (Freeman, 2013) [(2)]. The purpose of every material has to be really clear so that the decision can be made when there is more than one contender.

As will be the case with the geometry of the beam and the reinforcement, the materials will be chosen by the author, always considering that on the second case post-tensioning will be added and, therefore, the concrete will have to be able to withstand the compressing load and the active steel the loads.

To start with the modelling of the simple beam some DIANA FEA tutorials have been used. In the first place, *Creep Response of a Prestressed Concrete Beam under Sustained Load* (DIANA FEA, 2010) [(3)] has been helpful to start getting familiar with the software.
All the following steps will be explained in the order they have been followed so that the procedure can be followed and copied if necessary, but how to use the software will not be explained because the document mentioned before does practically the same.
The part that will be thoroughly explained will be when the programming language Python is being used. This thesis will be useful for the ones who want to start to use this feature from scratch.

2.1. Properties of the used materials

Concrete C45/55	**Value**	**Units**
Young's modulus *E*	3.62832e+10	N/m^2
Compressive strength f_{cm}	5.3e+07	N/m^2
Tensile strength	3.79545e+06	N/m^2
Mass density	2400	kg/m^3
Reinforcement steel (BST 500 S)		
Young's modulus	2e+11	N/m^2
Yield stress	5e+08	N/m^2
Mass density	7850	kg/m^3
Steel plates (S275)		
Young's Modulus	2.1e+11	N/m^2
Yield stress	2.75e+08	N/m^2
Mass density	7850	kg/m^3

Table 1: Properties of the materials. Simple beam without post-tensioning

2.2. Definition of the geometry of the beam

Once the properties of the materials have been defined, the geometry of the beam has to be chosen and so has the reinforcement.

Considering that the measures do not have to fulfil any code, since they are being chosen arbitrarily just to see how the beam reacts to the increase of the applied load, they will be the following:

- The span of the beam will be 6m.
- The cross-section geometry will be 600x400mm^2.

In the following picture appears the completely defined geometry. *All measures are in mm*.

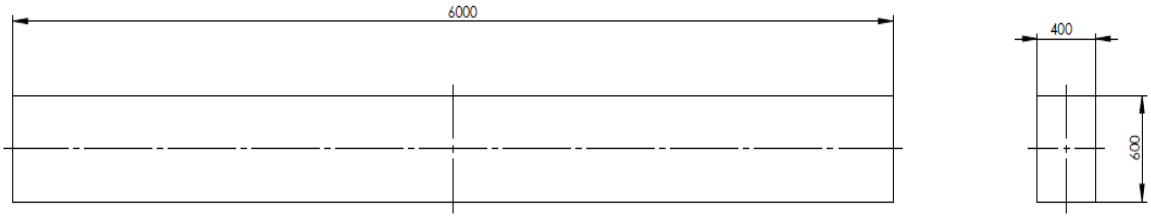

Figure 1: Geometry of the beam

It is of the most importance to remind that this thesis wants the beam to crack, so there is no specific load to apply to the beam. That means that the reinforcement does not have to be calculated and, therefore, there is no specific amount of steel needed.

2.3. Definition and distribution of the passive reinforcement

Three horizontal bars will be used, both in the superior and inferior part of the beam. Some cover is needed so that the steel bars are not affected by corrosion. This cover will be of 10 cm on both sides and also on the superior and inferior part of every bar.

The horizontal bars will be separated 10cm from one another and their diameter will be 12mm.

In the following picture the geometry can be observed. *All measures are in mm.*

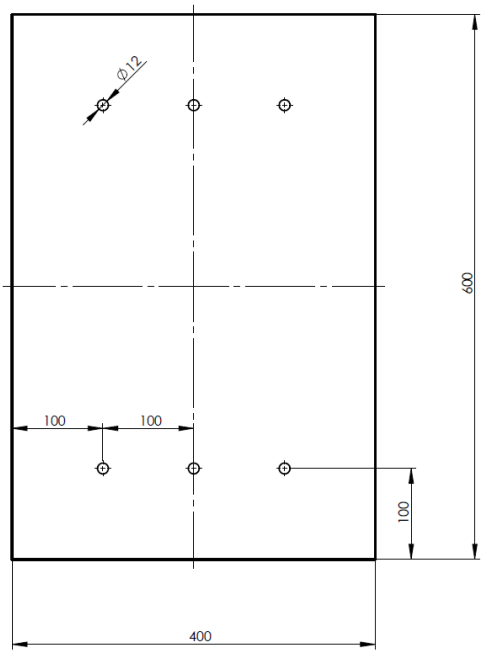

Figure 2: Geometry of the cross-section and distribution of the passive reinforcement

To what the stirrups concern, it has been considered necessary to maintain the same distribution along the whole beam, meaning that the separation between them will always be exactly the same. The diameter will be 8mm, unlike the horizontal rebar that is 12mm.
In the following picture the distribution can be seen. Every discontinuous line is a stirrup. *All measures are in mm.*

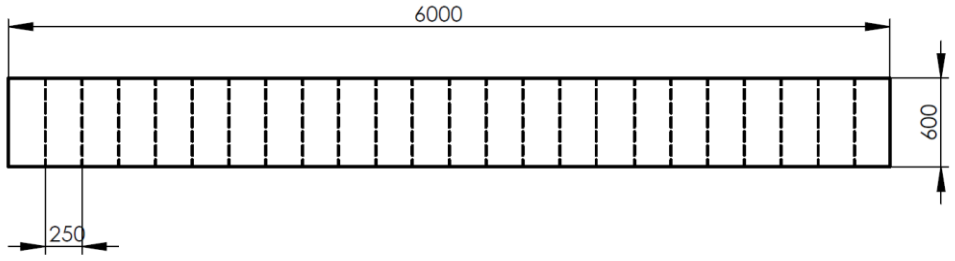

Figure 3: Geometry of the beam and distribution of the stirrups

2.4. Definition and position of the steel plate

The load that will be applied in the first case scenario cannot be applied directly in the beam because that would create a crack in that place. To avoid this problem a steel plate will be placed in the surface of the beam.
Its position will be in the centre of the beam, so that the load is applied in the geometrical centre; x=3m.
In the following picture the geometry of the steel plate is represented. *All measures are in mm.*

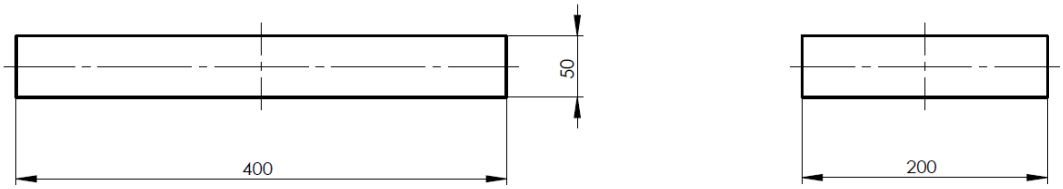

Figure 4: Geometry of the steel plate

The plate not only will keep the beam from cracking, it will also distribute the load in a little surface. This surface is 200x400mm^2.
Once the plate has been added, the complete geometry of the beam is the following:

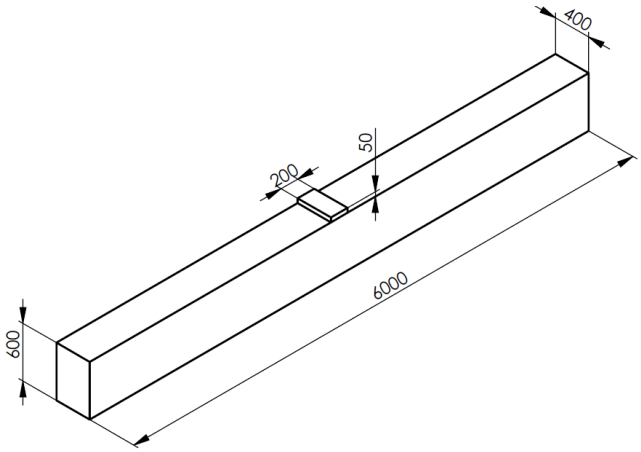

Figure 5: Isometric view of the beam with the steel plate

2.5. Boundary constraints or supports

To make things easier when it comes to defining the boundary constraints, the supports will be exactly the same in both ends of the beam, meaning it will be completely symmetrical.

Neither end of the beam will have the translation in the X direction restricted. There is no load that will act in that direction, so the beam will be able to experience a translation. Even though thermal expansion is not taken into account, not having the translation fixed in the X direction allows the beam contract and expand freely in case it is needed.

On both sides of the beam the translation in the Z direction has to be restricted because it is the direction in which the load will be applied.

When it comes to restrictions in the Y direction, they are not needed at all because no load or reaction acts in that direction.

Considering the symmetry, both end inferior edges will be selected and the translations will be fixed in the directions mentioned before.

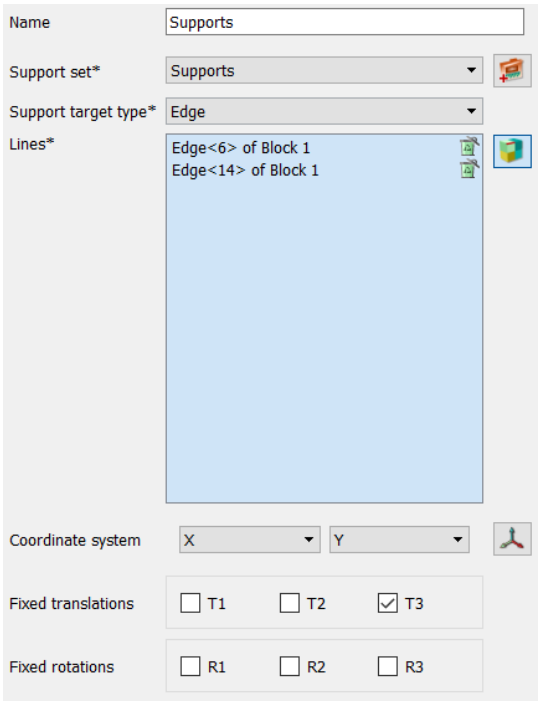

Figure 6: Definition of the supports

2.6. Mesh

As mentioned in *Finite Element Mesh* Refinement (COMSOL, 2016) [(4)], meshing is a crucial part in the Finite Element Software[2].

The smaller the mesh elements are, the more accurate are the obtained results of the analysis but also the more time it takes to run it.

For this case it was decided that the shape of the elements would be cubes and the dimension of them 0.1x0.1x0.1m^3. This way the results obtained would be really accurate and the time to execute the analysis, taking into account that the span of the beam is only 6m, would not be very long.

[2] The accuracy that can be obtained from any FEA model is directly related to the finite element mesh that is used. The finite element mesh is used to subdivide the CAD model into smaller domains called *elements*, over which a set of equations are solved. This process of mesh refinement is a key step in validating any finite element model and gaining confidence in the software, the model, and the results.

Once the mesh properties have been set, as it can be observed in *Figure 7*, then the meshing process can be executed.

The only elements that need to be selected when creating it are the two only blocks (concrete beam and steel plate) because the other elements, such as the rebar and the stirrups are inside and embedded, which means that they are a whole and, therefore, selecting the beam means also selecting the reinforcement.

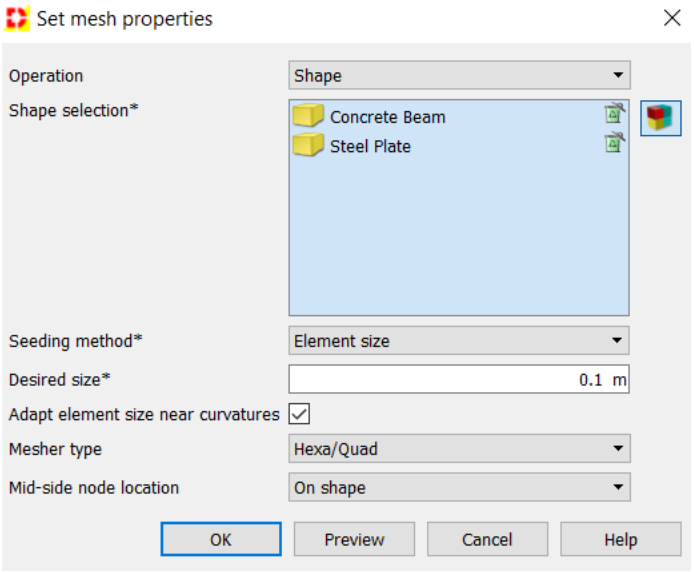

Figure 7: Setting the mesh properties

The result when the mesh has been executed is the following. It can be noted that the number of cubes is integer because there are no decimals in the measures set in *Figure 7*.

The steel plate has been painted in a different colour to ease the differentiation between the two blocks.

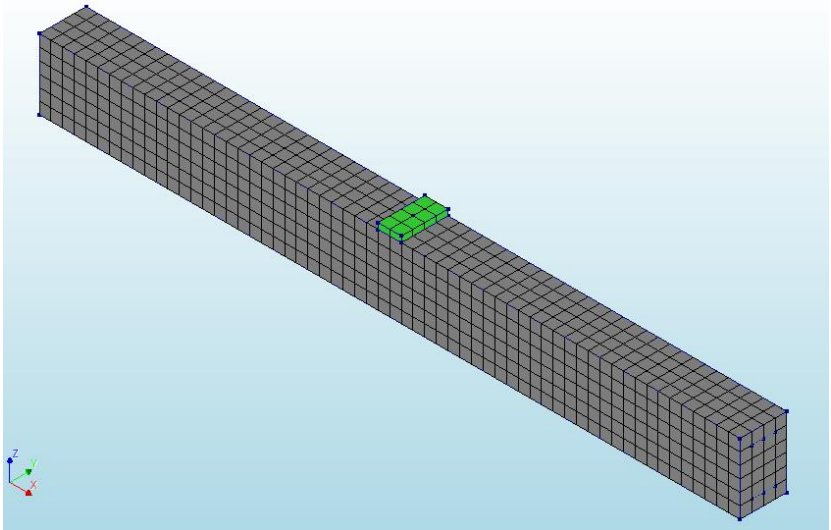

Figure 8: Mesh of the beam

2.7. Application of the loads

Two different type of loads will take place in this case. In the first place the self-weight of the beam and in second place the point load that will keep increasing until failure.

As explained in *Creep Response of a Prestressed Concrete Beam under Sustained Load* (DIANA FEA, 2010)[(2)] the self-weight is automatically applied when it is selected on the load cases.

Also, the document mentioned before explains that, first of all, the point in which the load will be applied has to be defined and included in the steel plate geometry. Once this procedure has been completed, the load can be introduced.

As said before, the aim of the thesis is to apply a load and keep increasing it until breakage. To do so, an initial value has to be determined and with the "load steps" it will be increased gradually.

The initial value will be -1000N in the Z direction. The value is negative because the force has to cause a positive bending moment in the beam, so that the cracks start to appear in the inferior part of it. Also, it would not make any sense to apply it in the positive Z direction, because no load in real life will act this way.

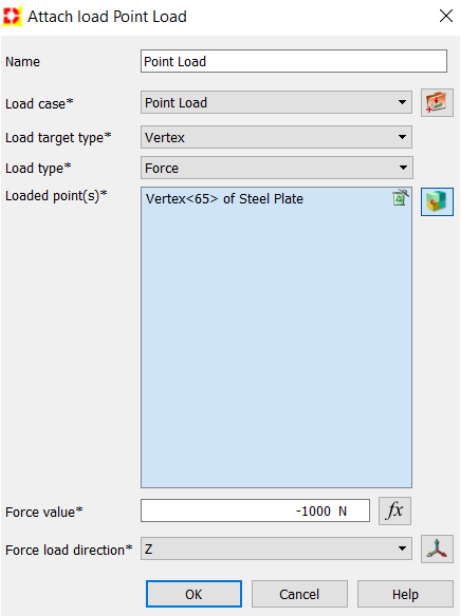

Figure 9: Definition and application of the point load

It has been decided that in every step the force will be increased by a factor of four. That means the first time the force will be -4000N, the other one -8000N, the other -12000N and so on, until a number of 60 iterations is reached, but that does not mean that the last one is when the breakage will take place. To observe and evaluate which is the iteration when the failure happens, the analysis has to be studied and then conclusions can be drawn.

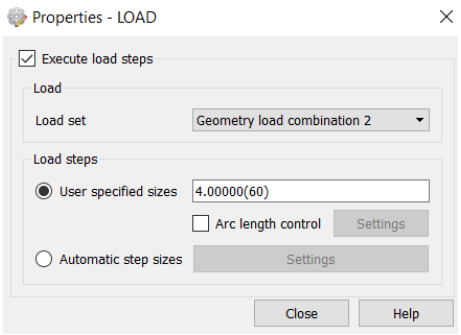

Figure 10: Load steps augmented by a factor of four (sixty iterations)

2.8. Nonlinear analysis

As Wasim Younis says in *An Overview of Part and Assembly Stress Analysis* (Younis, 2009)[(5)] the nonlinear analysis generally falls into three categories[3].
In the website ("Linear and nonlinear structural analysis", 2017)[(6)] it is said that <<a nonlinear analysis is an analysis where a nonlinear relation holds between applied forces and displacements. Nonlinear effects can originate from geometrical nonlinearity's (i.e. large deformations), material nonlinearity's (i.e. elastoplastic material), and contact. These effects result in a stiffness matrix which is not constant during the load application>>.

To start with, in the first case that will be studied there will be two different kinds of load (self-weight and point-load).
The order of the loads when applying them will be:

1. Self-weight
2. Point-load

Once these loads have been applied, the nonlinear analysis will be run and conclusions can be drawn. Despite the analysis being nonlinear, before cracks start to appear, all the materials will experience a phase of linearity in their behaviour. Consequently, both displacements and stresses can be estimated as long as this situation lasts.
Before running the analysis, the outputs have to be selected in order to obtain the results that are of interest and will allow studying and commenting the analysis afterwards. To give an example, in this document the most important outputs are: displacements, stresses and cracks. Depending on the goal of the analysis the outputs may change partially or even completely.
In section *8.3.9 Selection of the outputs* the explanation is more detailed and the outputs used in this study are exposed.

2.8.1. Linear behaviour

In all the steps where the load is not high enough to cause cracking in the concrete there will be a linear behaviour. This means the load will increase by the exact factor of four and the deformations will increase the same amount between two steps during some iterations.
The loads applied in all the steps are the self-weight and the load that will keep increasing in every single step. In very load step the initial force (1000N) will be multiplied by a factor. This factor keeps increasing by four in every iteration.
Considering that the first step will just have the self-weight, it will not be until the second step where the point-load will be applied. This means that in the second load-step the beam will be under the self-weight and the initial point-load which will be multiplied by a factor of 4. The third load-step will be exactly the same but changing the factor from 4 to 8, and the fourth load-step from 8 to 12. This will go on until the beam experiences failure.
In this section only the linear cases will be studied.

[3] **Geometric nonlinearity:** this is where a component experiences large deformations and as a result can cause the component to experience nonlinear behaviour. A typical example is a fishing rod.
Material nonlinearity: when the component goes beyond the yield limit the stress/strain relationship becomes nonlinear as the material starts to deform permanently.
Contact: includes the effect when two components come into contact where they can experience an abrupt in stiffness resulting in localized material deformation at the region of contact.

2.8.1.1. 1st load-step

The first load applied, before the point-load in the centre of the beam, is the self-weight. Once the beam is built and placed, it immediately has to withstand its own weight. That makes it bend because of the positive moment it is experiencing, but the displacements are really small.

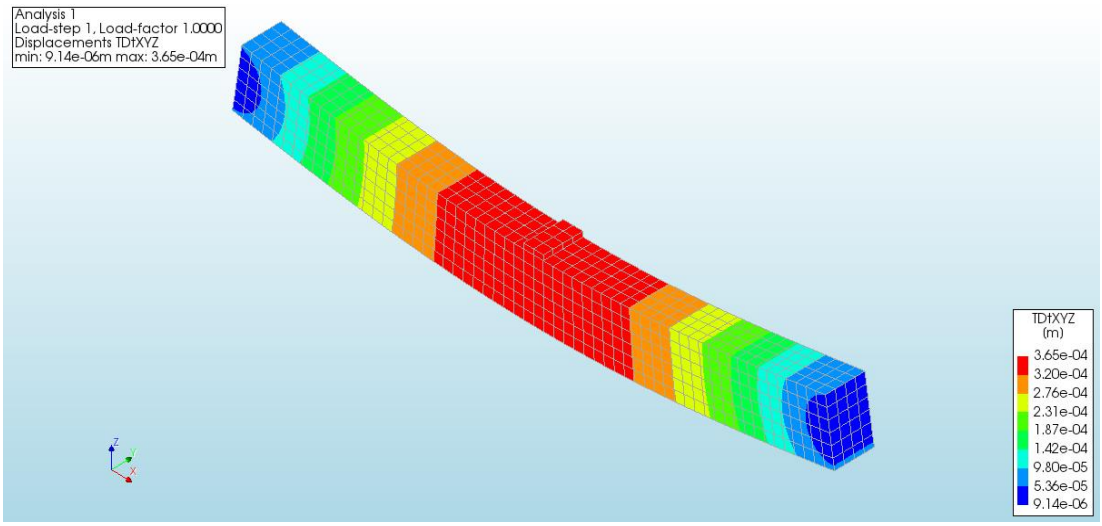

Figure 11: Total displacements in concrete on the 1st load-step

The results in *Figure 11* are expressed in metres. That means that the maximum displacement, which can be found in the very centre of the beam, is 0.365mm. The representation of the beam in the software exaggerates the deformation but in reality, it is nearly impossible to spot the displacements with a naked eye.

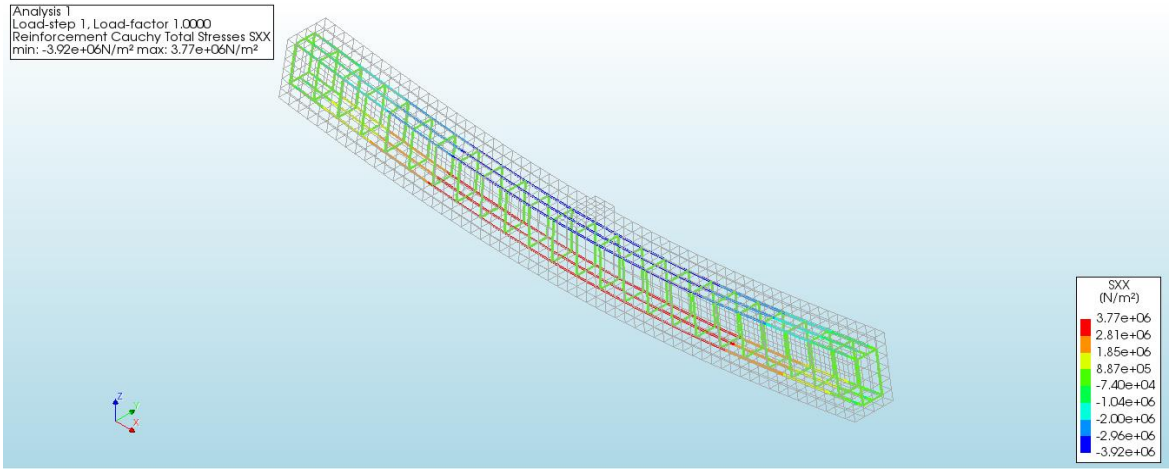

Figure 12: Total stress of the reinforcement in the X direction on the 1st load-step

In both *Figure 11* and *Figure 12* it should be noted that the self-weight makes the beam experience a positive bending moment, which means that the superior part of the beam will be under compression and the inferior under tension. Obviously, the rebar is bonded to the concrete, and that means that they will both experience the same, even though they will react in a completely different way.

The advantage of representing the stress in the X direction is that it directly indicates if the reinforcement is working in tension or compression. If the stress is positive it means the rebar is experiencing tension. On the contrary, it will be under compression (negative stress).

2.8.1.2. 2nd load-step

In the second iteration the point-load is introduced and therefore, the force has a value of 4000N, which is the initial force (1000N) multiplied by a factor of 4. Here, obviously, the displacements will start to get bigger, but for the moment the behaviour will be completely linear.

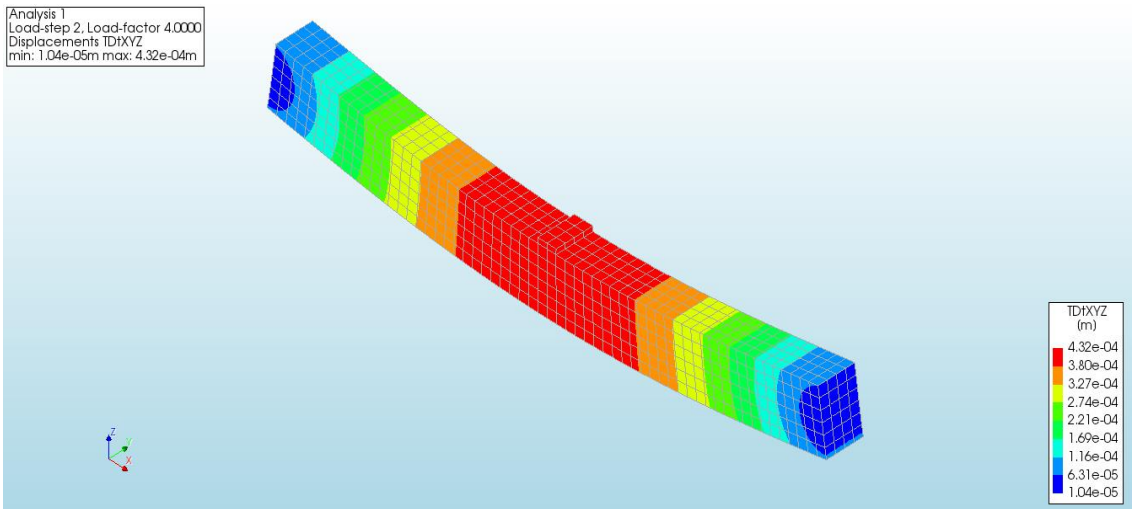

Figure 13: Total displacements in concrete on the 2nd load-step

In *Figure 13* it can be seen that the biggest displacement is 0.432mm, meaning that the load of 4000N has increased the previous displacement in 0.067mm. This means that every time the load is increased by a factor of four, as long as the behaviour is still linear (no cracks), the displacements will increase something around 0.067mm.

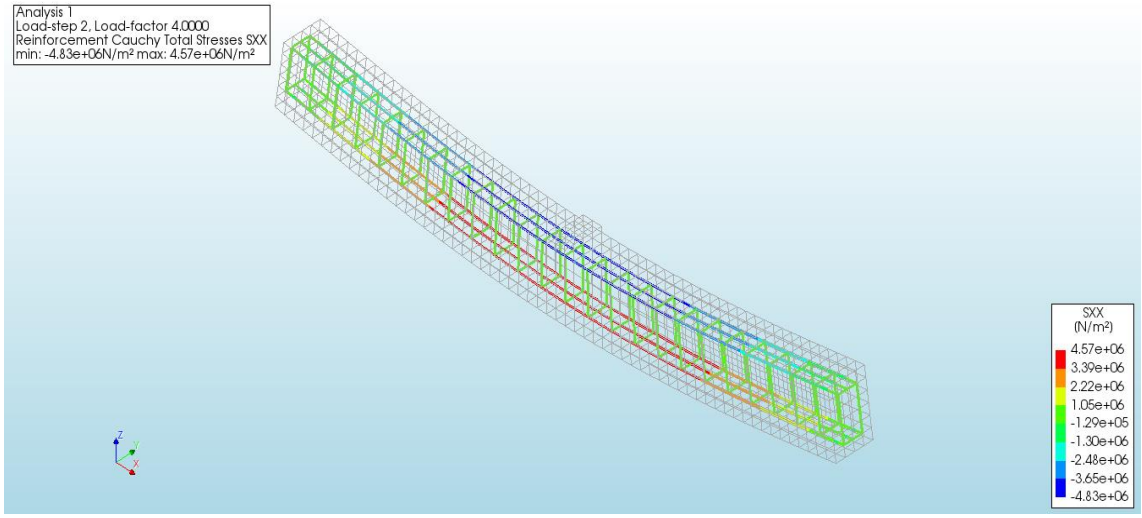

Figure 14: Total stress of the reinforcement in the X direction on the 2nd load-step

The exact same thing will happen with rebar. It has increased 0.8MPa with the application of the load, meaning that in the next load-step it will increase practically the same amount.
In *Figure 14* it can be seen that centre part of the inferior rebar is the most requested and will be the one that will eventually yield when the concrete cracks.

2.8.1.3. 3rd load-step

In this iteration it can be verified that the behaviour is linear. This can be done by comparing the displacements of this load-step to the ones in the load-step before, and checking that the increase is the one mentioned before (0.067mm).

If the highest displacement value from the previous load-step is subtracted from the highest in this one, it can be noted that the obtained value is really close to 0.067mm. It is exactly 0.069mm, meaning that the linear behaviour is indeed taking place for the moment.

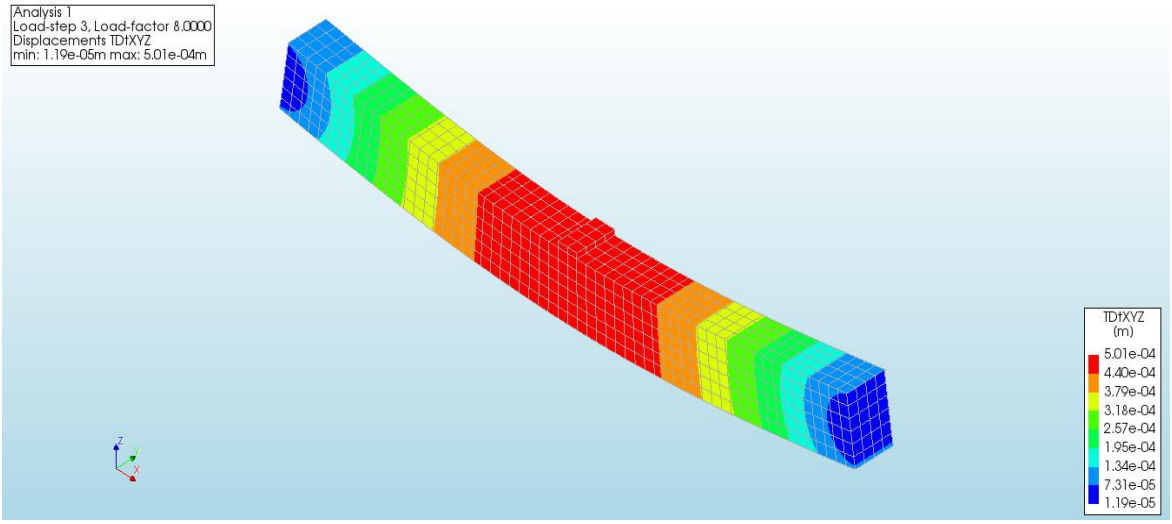

Figure 15: Total displacements in concrete on the 3rd load-step

Now, the load factor will be 8, meaning that the load applied in the centre of the beam is 8000N.

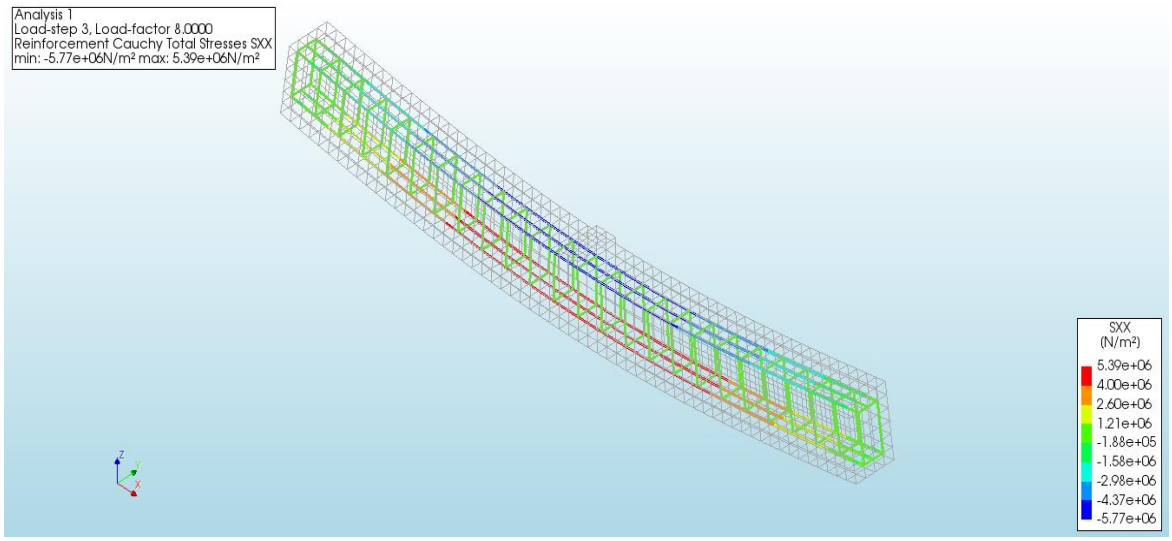

Figure 16: Total stress of the reinforcement in the X direction on the 3rd load-step

The same can be done with the stresses of the reinforcement. If the highest value in *Figure 14* is taken and the increase (0.8MPa) is added to it, the obtained value is 5.37MPa, which is really close to the 5.39MPa that appear in the legend of *Figure 16*.

2.8.1.4. Demonstration of the linear behaviour (14th load-step)

This is the very step before the beam starts cracking. As seen in the two anterior cases, the behaviour of the structural elements is linear, which means the displacements have gotten bigger thirteen times the first increase observed.
We could estimate the highest displacement value with the following formula:

$$Displ_{max\ n\ loadstep} = Displ_{FirstStep} + \Delta_{displ} \cdot \Delta_{loadsteps}$$

Equation 1: General equation to estimate the maximum displacement of the following load-step on linear regime

The maximum displacement of this step will be estimated knowing that the increase of load-steps is 13. From the 1st to the 14th.

$$\boldsymbol{Displ_{max\ 14th\ loadstep}} = 0.365mm + 0.067mm \cdot 13 = \mathbf{1.24mm}$$

Equation 2: Equation to estimate the maximum displacement in the 14th load-step on linear regime

It can be observed that the result is really similar to the one obtained in the following figure, meaning that the behaviour is, as said, completely linear.

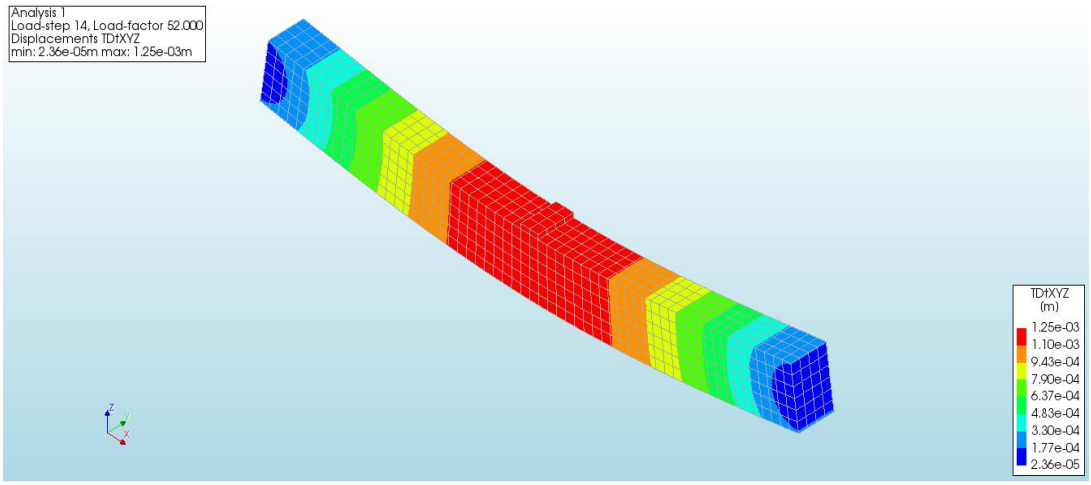

Figure 17: Total displacements in concrete on the 14th load-step

Unlike the first three cases, this displacement could be spotted with a naked eye, although all beams in the serviceability state experience the same or even bigger displacements.

The exact same thing could be done with the stress of the post-tensioning.

$$Stress_{max} = Stress_{FirstStep} + \Delta_{stress} \cdot \Delta_{\text{loadsteps}}$$

Equation 3: General equation to estimate the maximum stress of the following load-step on linear regime

The maximum stress of this step will be estimated.

$$\boldsymbol{Stress_{max}} = 4.57MPa + 0.8MPa \cdot 13 = \mathbf{15MPa}$$

Equation 4: Equation to estimate the maximum stress in the 14th load-step on linear regime

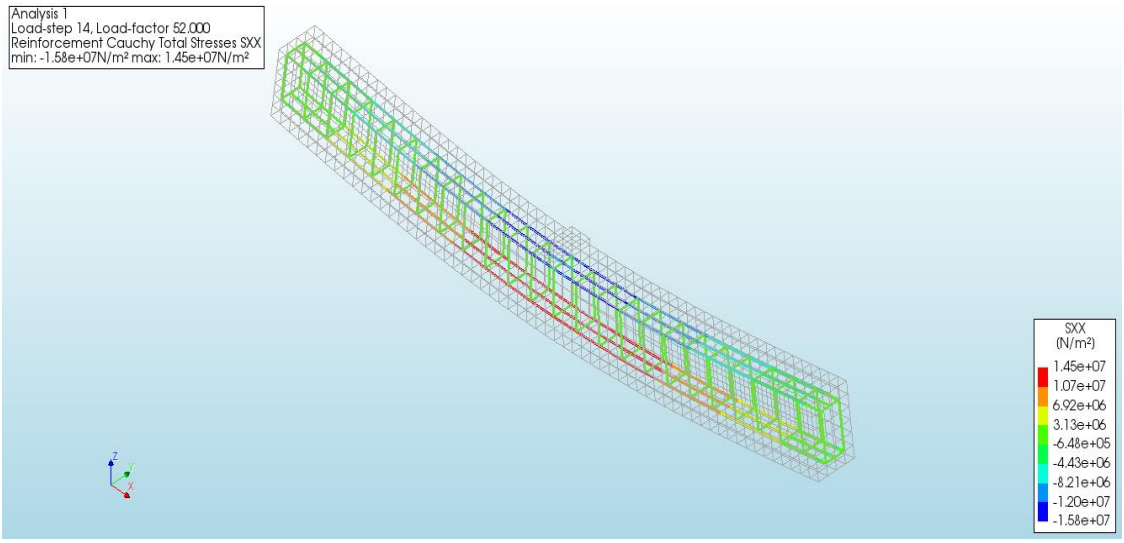

Figure 18: Total stress of the reinforcement in the X direction on the 14th load-step

The result obtained in *Equation 4* deviates a bit from the one appearing in *Figure 18*, but the approximation is pretty accurate taking into account the order of magnitude.

2.8.2. Nonlinear behaviour

It is not until the 15th iteration, where the load factor is 56 and so the force is 56000N, that the beam starts experiencing cracking. Until this point the deformation could have been guessed if the factor in which the force was increased was known, but from now on, this linear behaviour will no longer exist. That means the deformation will not increase in the measure the force does and so the performance now will become unpredictable.

2.8.2.1. 15th load-step

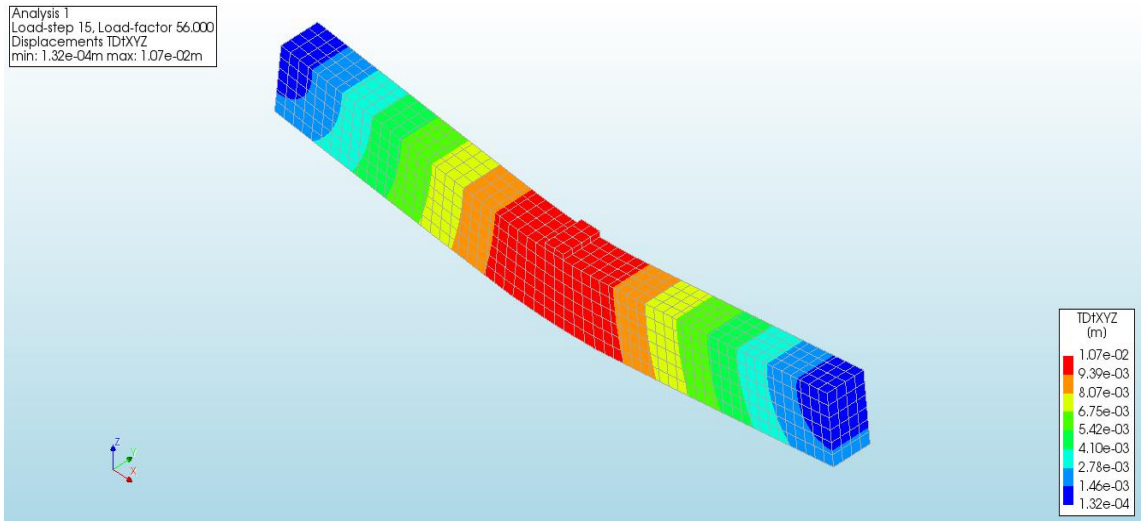

Figure 19: Total displacements in concrete on the 15th load-step

In *Figure 19* it can be noted that the beam is not bending as it did in the cases before. The crack now separates the two parts and the rebar is the element that bonds them together. With every step now the displacements will increase in a bigger measure compared to the anterior cases.

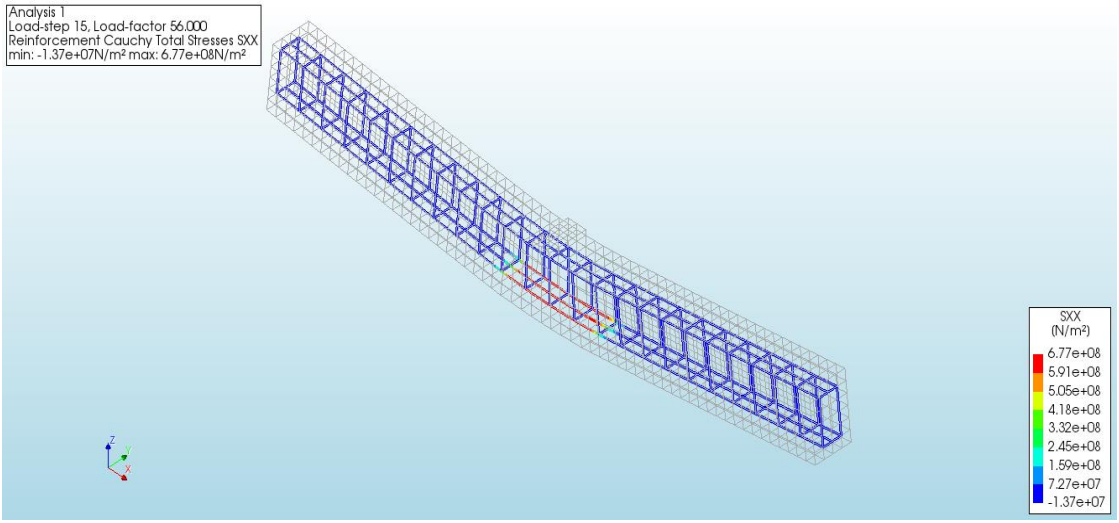

Figure 20: Total stress of the reinforcement in the X direction on the 15th load-step

The cracking can be spotted in *Figure 20* because the reinforcement now is only working in tension in the centre of the beam, exactly where the crack has started. All the other parts of the reinforcement are experiencing compression (negative stresses).

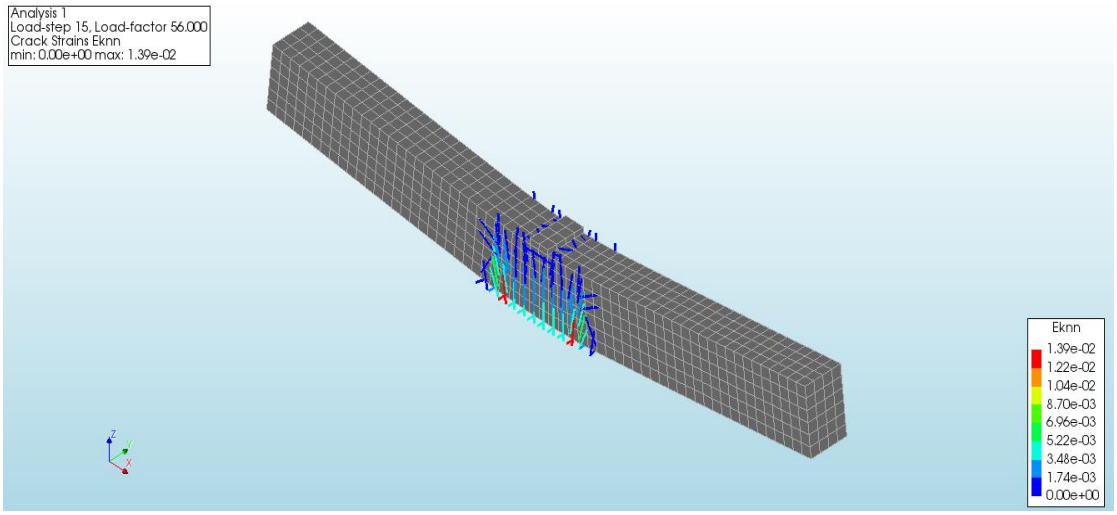

Figure 21: Cracks on the 15th load-step. First time the beam experiences cracks

Figure 21 shows the cracks and exactly where they are. It can be noted that the inferior part is where they are wider (red lines), meaning that is where they started propagating, and they keep spreading to the superior part and sides (blue lines).

2.8.2.2. 37th load-step

The following load-step is the last before the beam reaches failure. At this point the displacements can be spotted easily with a naked eye.

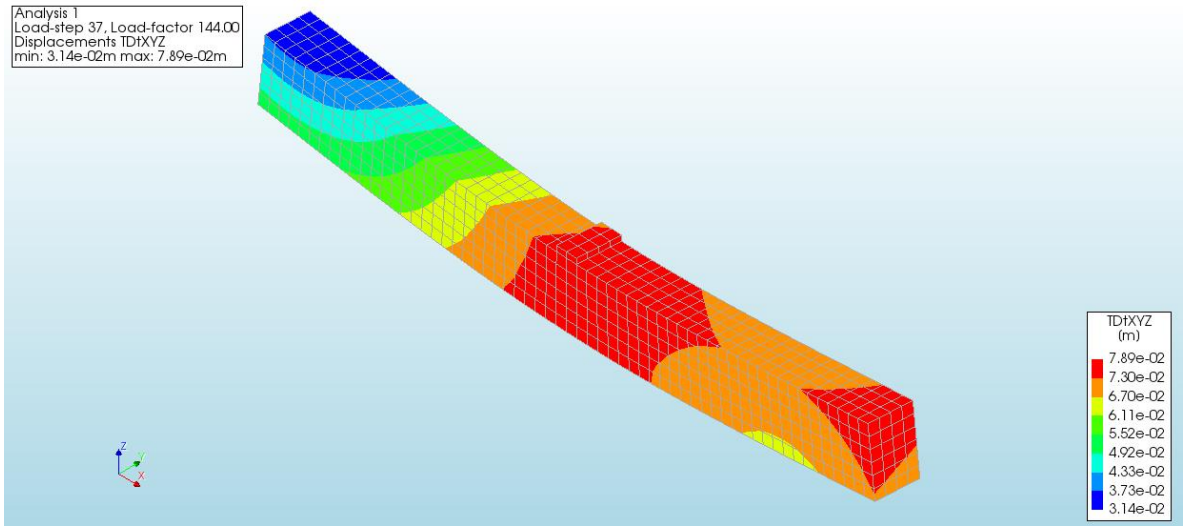

Figure 22: Total displacements in concrete on the 37th load-step

In *Figure 22* it can be appreciated that the failure is about to happen because the distribution of the displacements is no longer symmetrical, which means that one half of the beam is under more stress than the other, being the right part in this case.

The biggest displacement, taking place in most of the right part of the beam is 7.89cm, which is really high and tells that the failure is close.

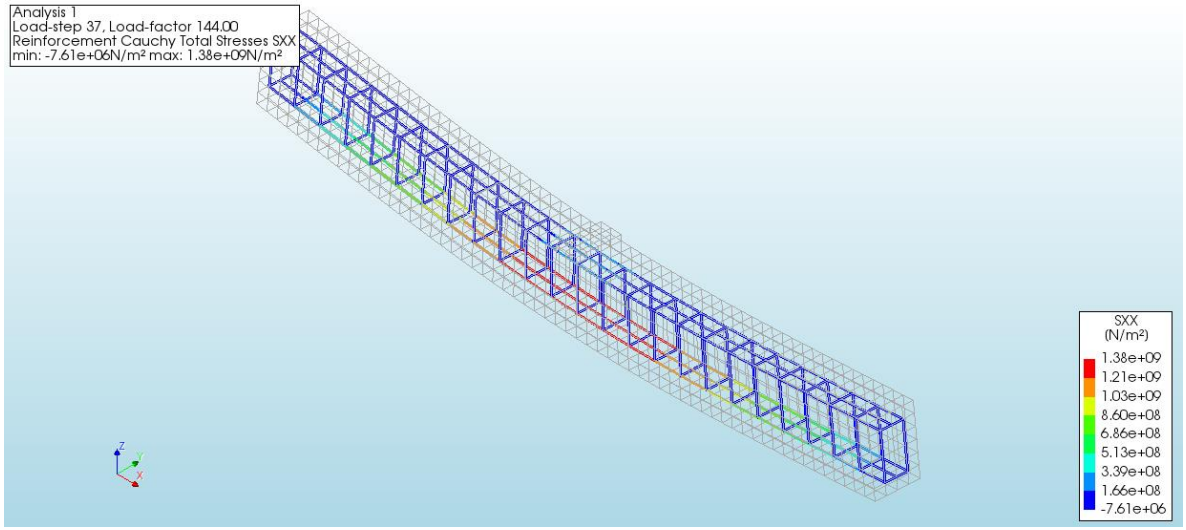

Figure 23: Total stress of the reinforcement in the X direction on the 37th load-step

Logically, the more the load increases, the more it does the stress of the reinforcement. The rebar has already yielded a while ago, but it can still withstand the load of 144000N.

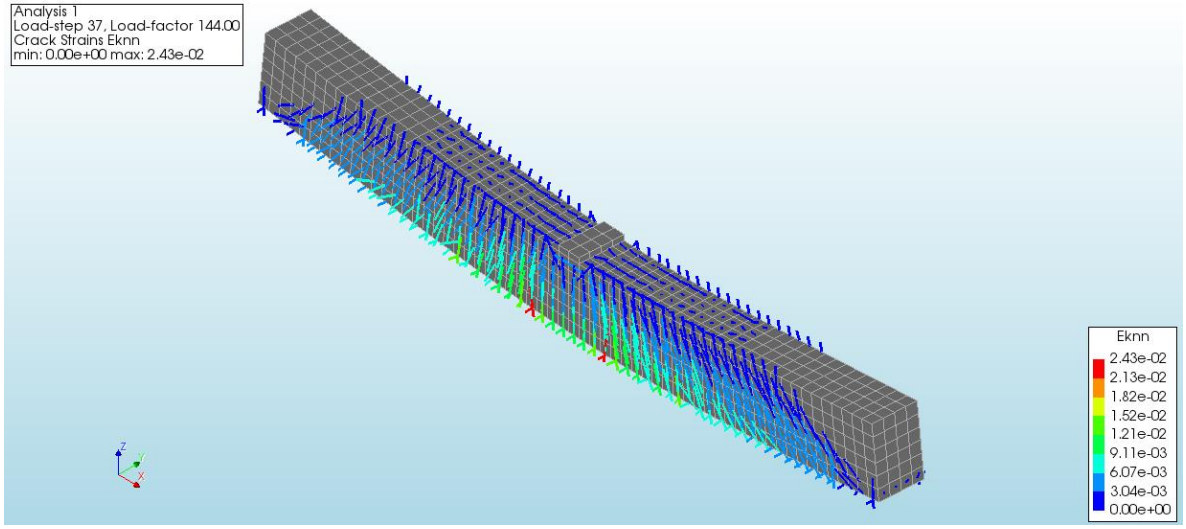

Figure 24: Cracks on the 37th load-step

At this point the cracks have spread all along the beam. Even though they cover most of the beam, that does not stop the reinforcement from withstanding the load and keeping the concrete together.
Not only more cracks appear, also the existing ones become wider every single load-step.

2.8.3. Failure

When the rebar can yield no more and the load keeps increasing, there is a moment where everything collapses.

Even though in the following pictures the beam does not appear bended or totally broken, it can be guessed because of the enormous displacements all over it and the distribution of them. It can be observed that the symmetry now is totally non-existent.

2.8.3.1. 38th load-step

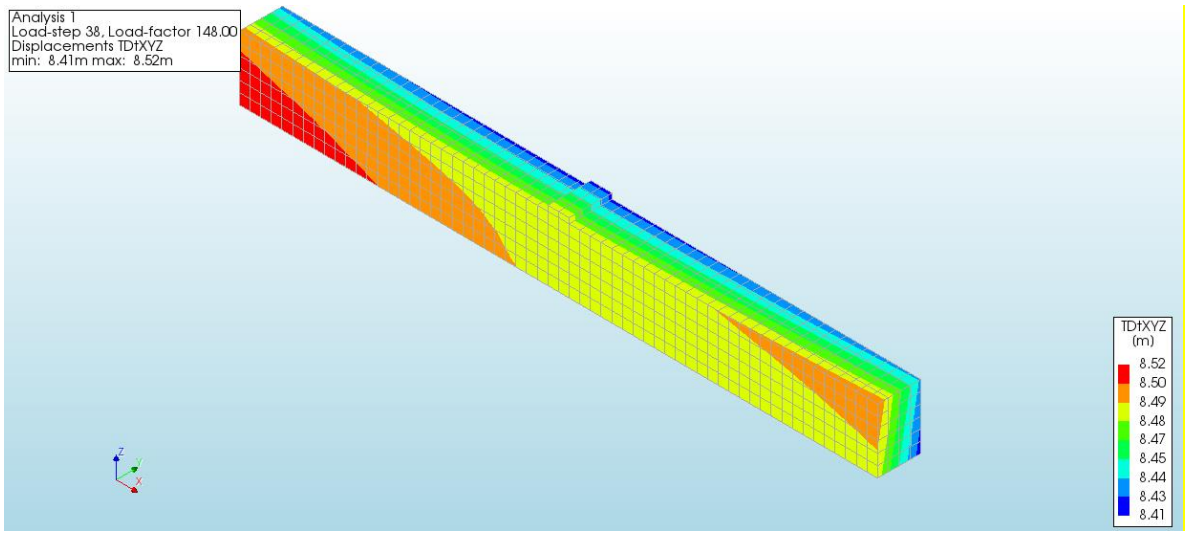

Figure 25: Total displacements in concrete on the 38th load-step

As said before, in *Figure 25* the distribution of the displacements looks completely different from all the previous ones. Also, if the values of these displacements are compared to the other ones it can be noted that they are extremely big.

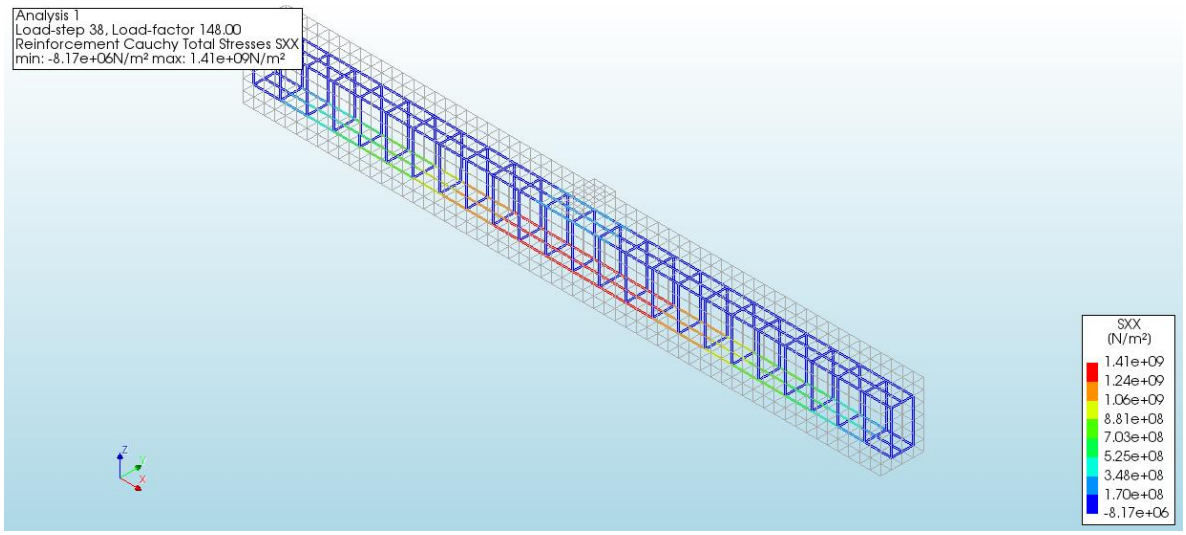

Figure 26: Total stress of the reinforcement in the X direction on the 38th load-step

Once the failure has taken place, the rebar has yielded to the point of breakage. In *Figure 26* the values in red are really high, meaning that the stresses are bigger than the ones the rebar can withstand.

If the beam appeared with the deformation, it would adopt a V shape. That means that the rest of the rebar (superior part and both ends) are under compression, and that the stresses have negative values.

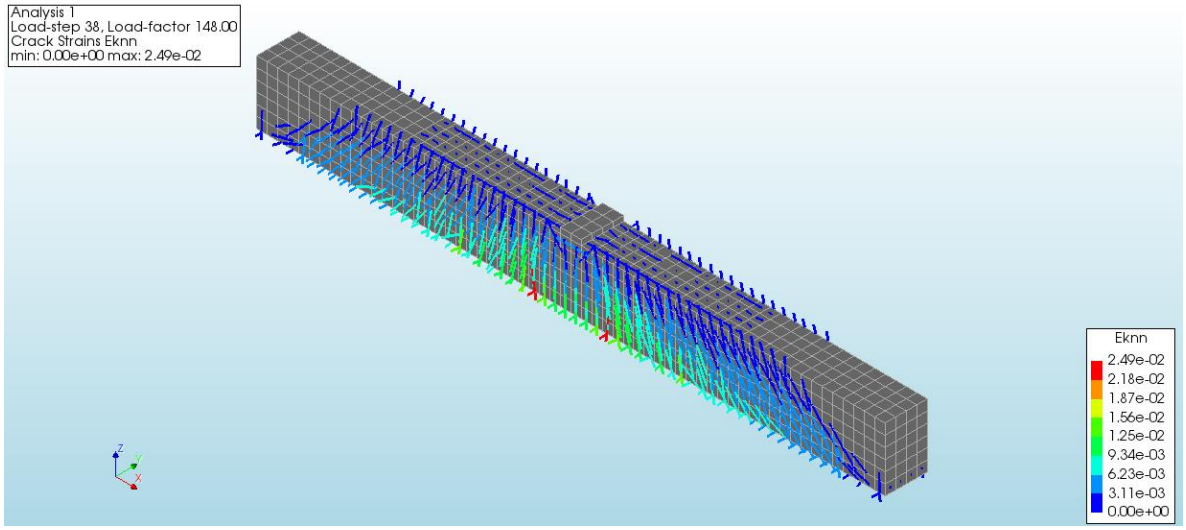

Figure 27: Cracks on the 38th load-step

The cracks have extended throughout the whole beam, making the reinforcement work much more and forcing it to its failure. Due to the positive bending moment, the cracks appear in the inferior part of the beam (in the centre) and keep spreading. The ones that appear first, now in colour read, become wider and wider every time the load increases.

3. Modelling of a beam with post-tensioning

Now that the beam without post-tensioning has been analysed and studied, the post-tensioning system can be introduced to see how it affects the behaviour when the load is applied.
Once the material and its properties have been defined, and therefore introduced in the material table, the active rebar will be introduced in DIANA and the analysis can be run again.
There are two different options when it comes to the post-tensioning: applying the post-tensioning load in one or both ends of the beam, and bonding the active reinforcement before or after applying that load. All the different options will be analysed and commented in section *5. DIANA options for the post-tensioning* so that the best option can be selected.
All the geometrical properties will be exactly the same as the case without post-tensioning, so the only thing left to define is the active rebar and its distribution.

3.1. Properties of the used materials

Concrete C45/55	**Value**	**Units**
Young's modulus *E*	3.62832e+10	N/m^2
Compressive strength f_{cm}	5.3e+07	N/m^2
Tensile strength	3.79545e+06	N/m^2
Mass density	2400	kg/m^3
Reinforcement steel (BST 500 S)		
Young's modulus	2e+11	N/m^2
Yield stress	5e+08	N/m^2
Mass density	7850	kg/m^3
Steel plates (S275)		
Young's Modulus	2.1e+11	N/m^2
Yield stress	2.75e+08	N/m^2
Mass density	7850	kg/m^3
Post-tensioning steel (ASTM A416M-2010)		
Young's modulus	1.9e+11	N/m^2
Yield stress	1.6e+09	N/m^2
Mass density	7880	kg/m^3

Table 2: Properties of the materials. Simple beam with post-tensioning

NOTE: When defining the properties of the reinforcement concrete, there is the option to bond it to the mother element. As it will be explained in section *5.1 Bonding to the mother element* this option will not be chosen because the tendons should be free when the post-tensioning load is applied. That will allow them to elongate and then transmit the load to the concrete, which will compress it.

3.2. Definition and distribution of the active reinforcement

Five horizontal bars will be used, distributed in the following way: two in the inferior layer, two in the superior and one just in the middle of them. Taking into account that the post-tensioning load will be big, it has been considered necessary using a diameter of 18mm.

As happened with the passive reinforcement, some concrete cover is needed so that the bars do not get corroded. This time the vertical cover will be 25cm and the horizontal 10cm.

In the following picture the distribution is represented, but it is just valid for both ends, because along the beam the reinforcement takes a parabolic shape.

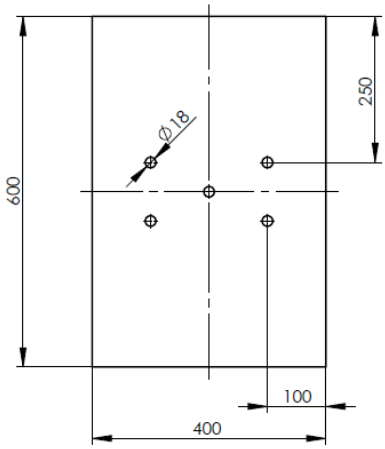

Figure 28: Geometry of the cross-section and distribution of the active reinforcement

Even though there are three layers, just the inferior one will be enough to see how they are all distributed.

When the load is applied to the active reinforcement it creates a negative bending moment because of the eccentricity of the tendons. This moment is able to counteract the one produced by the point-load and the self-weight, making the beam stay in a compression state so that the cracks start appearing and spreading way later than without post-tensioning.

The parabolic form of the tendons takes place because the positive bending moment created by the external loads is non-existent in both ends, and with this shape the eccentricity in the ends is 0 and, therefore, so is the negative moment. If the eccentricity had a value in the ends, the beam would experience some tension in the superior part and some cracks could appear.

So, it can be seen that where the positive bending moment (created by self-weight and point-load) is maximum (the centre), so is the eccentricity of the tendon. Therefore, the maximum bending moment will be in the centre of the beam, where the eccentricity is also maximum.

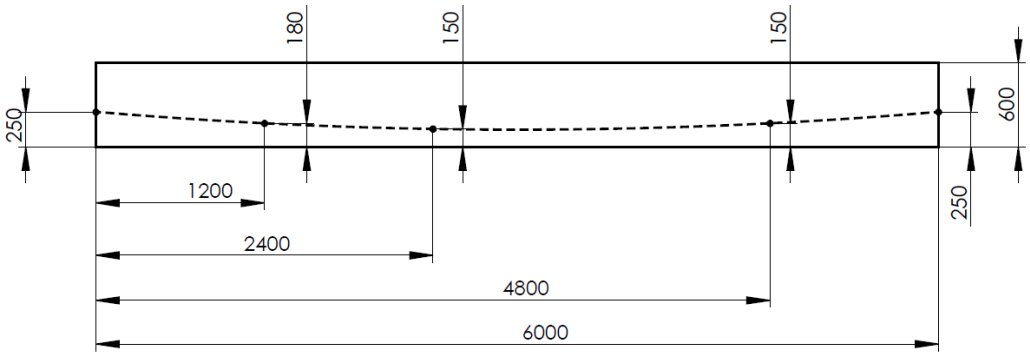

Figure 29: Distribution inferior layer of the active reinforcement along the beam

3.3. Application of the post-tensioning load

Once the active reinforcement has been placed there is one last thing to do concerning it, and that is applying the load.

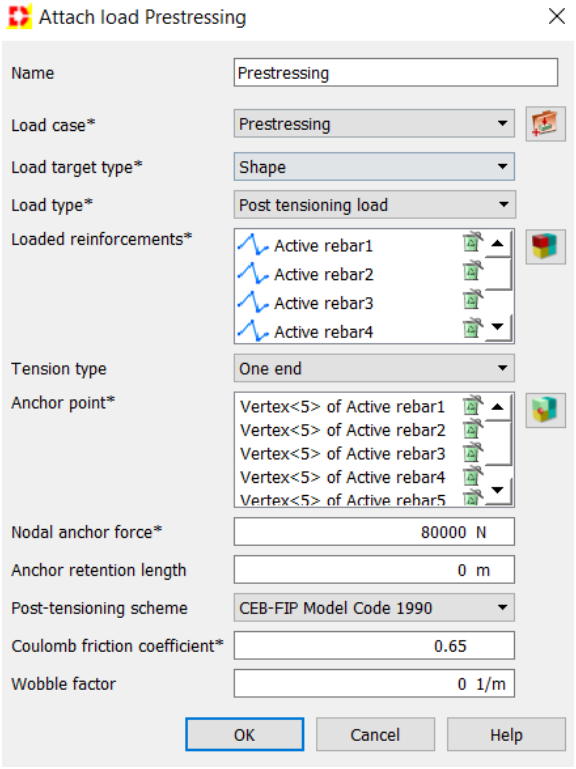

Figure 30: Attachment of the post-tensioning load

As can be observed in *Figure 30*, the first thing to declare is how many active bars there are and selecting them. Secondly, the points where the load will be applied have to be chosen and there are two options. The first one is applying the tension in both ends and the second is doing so in just one. Considering that one end of the beam is where the anchorage takes place, the load will be attached to the other end and its value will be that of 80000N per tendon.

Applying it in both ends could have some advantages and that is explained in section *5.2 Post-tensioning load applied in one or both ends.*

To end with, the other aspect that has to be taken into account is the Coulomb friction coefficient, which will be 0.65 according to the *Table 5.1* of the section *5.10.5.2 in the document Design of concrete structures in Eurocode 2 (Freeman, 2013),* considering that the tendons are internal deformed bars.

3.4. Nonlinear analysis

Whereas in the first studied case there were only two different kinds of load (self-weight and point-load), in this one post-tensioning load will be added, making the beam start from scratch when the point-load is applied. In other words, the post-tensioning counteracts the self-weight of the beam generating a bending moment on the opposite direction, consequently making it be in a neutral load state or even experience counter-deflection.

When the procedure of post-tensioning is done and the tendons have been stressed, the ducts are filled with grouting so that the corrosion does not affect them. This action will make them bond to the plastic ducts and, as a consequence, to the concrete.

The order of the loads when applying them will be:

1. Self-weight
2. Post-tensioning
3. Change of the properties of the active reinforcement from not bonded to bonded (grout pouring in reality)
4. Point-load

Undoubtedly, if the post-tensioning load is very large, the moment generated due to the eccentricity of the active tendons will be greater than that generated by the self-weight, so the beam will be counter-deflected. That is, the beam will be deflected in the opposite direction to the usual one.

Since the goal is exactly the same as in the anterior case (applying the load until failure), so will be the followed procedure concerning the application of the point-load.

Due to the nonlinear behaviour that the materials composing the beam take on when the load applied reaches a large value and cracks start to appear, there is no other option but to run a nonlinear analysis. This will allow studying all the stages that the beam goes through and to understand the behaviour of the materials at any moment.

3.4.1. Linear behaviour

Like the case without post-tensioning, during the first load-steps the materials will be experiencing a linear conduct. It will not be until the first cracks start to appear that this behaviour will change and, consequently, it will be impossible to predict the displacements.

3.4.1.1. 1st load-step

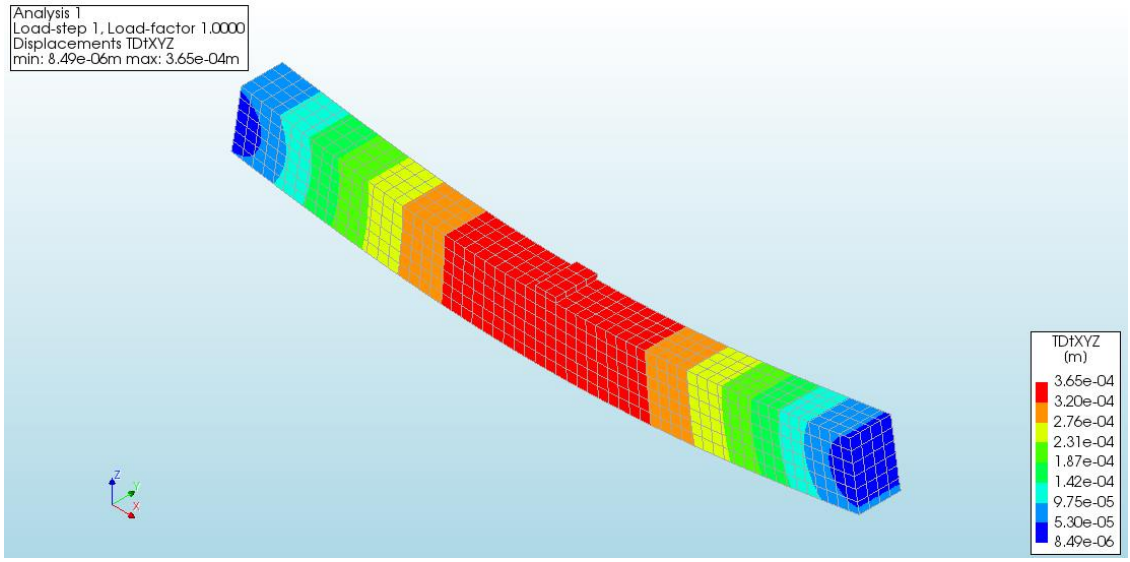

Figure 31: Total displacements in concrete on the 1st load-step

As commented before, the self-weight generates a positive bending moment (counter-clockwise) that deforms the beam downwards. Since the beam is exactly the same as the case studied before, so are the displacements once the self-weight is applied. That is if both *Figure 11* and *Figure 31* are compared it is noted that they are identical.

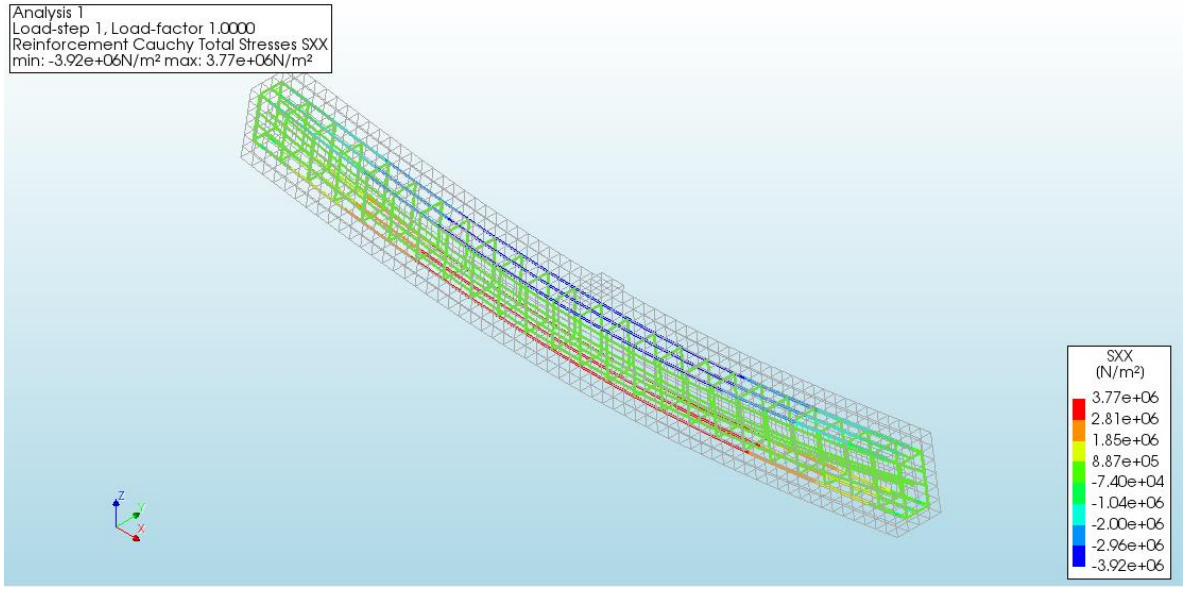

Figure 32: Total stress of the reinforcement in the X direction on the 1st load-step

Logically, the exact same thing happens if *Figure 12* and *Figure 32* are compared. Even though in the latest there are more tendons (active reinforcement) these ones are experiencing the same tension as the other ones because the post-tensioning load has yet to be applied.
Thus, when the said tension will be applied, the distribution of the stresses along the reinforcement will be a completely different matter.

3.4.1.2. 2nd load-step

Unlike the first studied case, this one has active reinforcement steel and, therefore, a post-tensioning load applied to them. As commented before, this load generates a clockwise bending moment due to the eccentricity of the tendons with respect to the neutral axis that counteracts the self-weight and allows the beam to withstand larger loads.

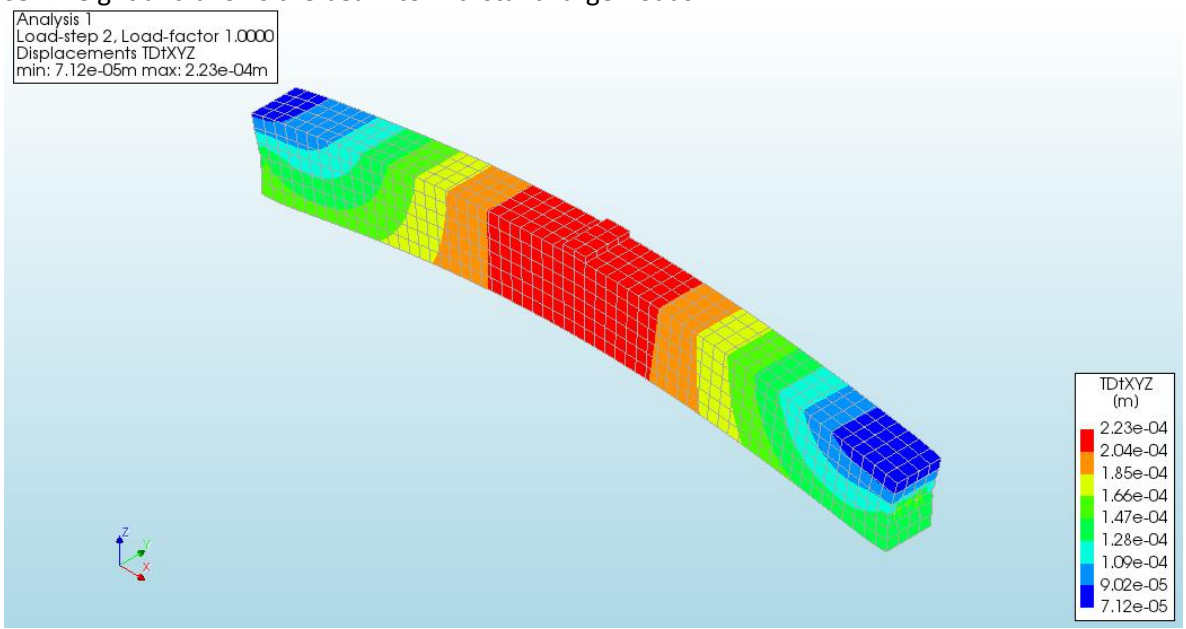

Figure 33: Total displacements in concrete on the 2nd load-step

In *Figure 33* it can be noted that not only the inferior part is no longer under tension, but it is now experiencing compression. This means that the moment generated by the loads applied to the tendons is greater than the one generated by the self-weight and, as a result, the beam is now subjected to a counter-deflection.

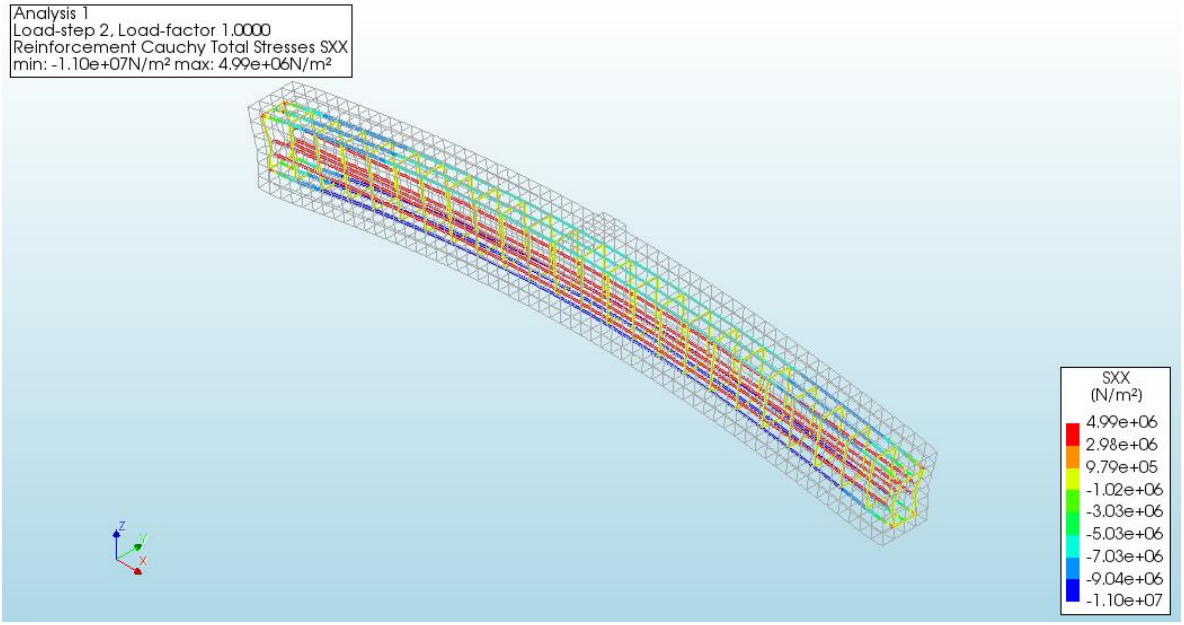

Figure 34: Total stress of the reinforcement in the X direction on the 2nd load-step

For the first time so far, the inferior part of the reinforcement is under compression. It can be observed in the legend, where the negative stresses mean the bars are under compression and the positive ones under tension.
Logically the tendons under the greater stress are the active reinforcement because of the post-tensioning load applied.

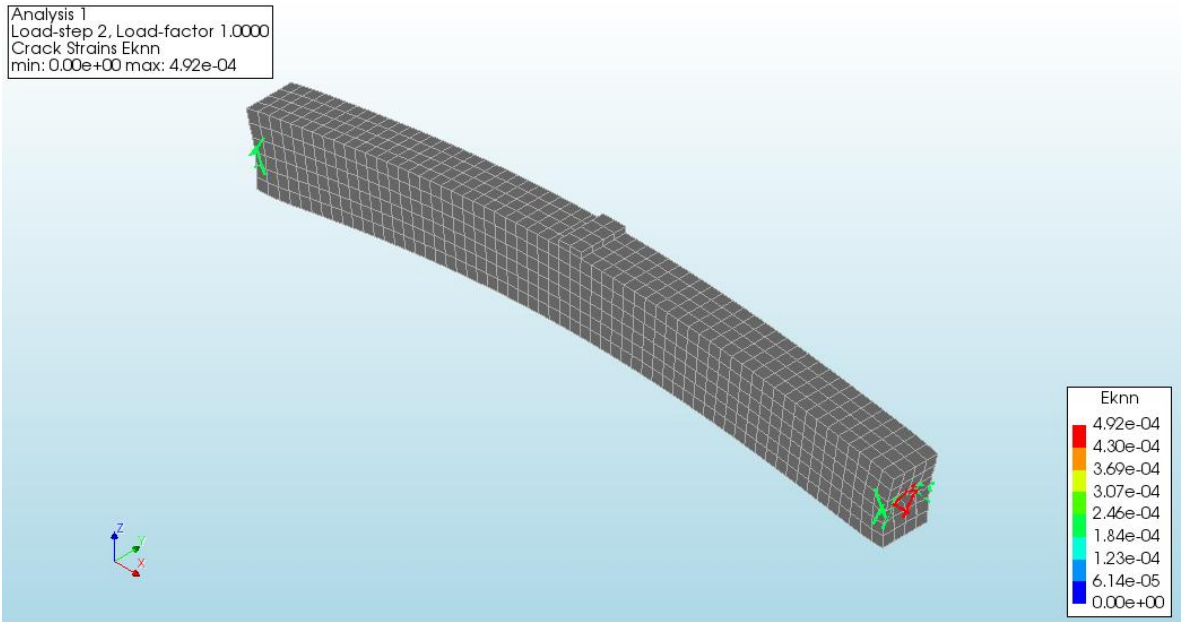

Figure 35: Cracks on the 2nd load-step. Cracks generated by the post-tensioning load

Taking into account that the load applied to every tendon is very large (80000N) this creates some cracks at both ends of the beam. That does not stop the beam and the materials composing it from behaving in a linear way. When the cracks start appearing in the inferior part of the beam, that is when the behaviour will change radically.

3.4.1.3. 4th load-step

The 3rd load-step does not appear because is the one where the properties of the reinforcement are changed and no load that would change the shape of the beam is applied.
This is the first step where the point-load is applied and, therefore, with this one starts the procedure to discover how many times the initial load can be increased until the beam collapses completely.

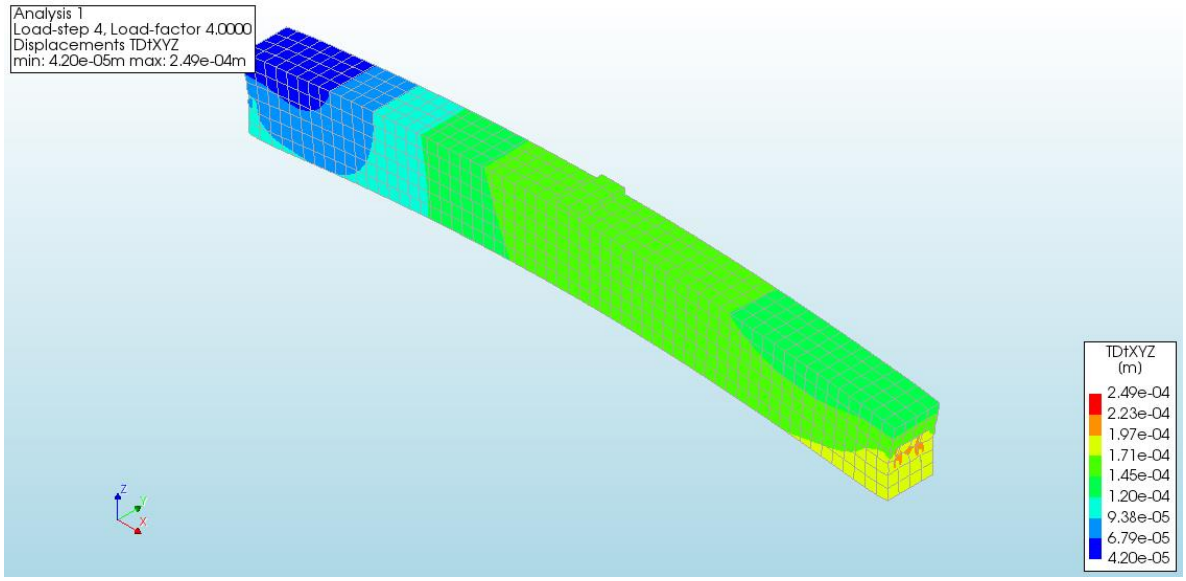

Figure 36: Total displacements in concrete on the 4th load-step

Now that the first load has been applied in the centre of the beam (4000N) the displacements are very small because the beam is really close to a neutral state, which would mean that there are no tensions in both inferior and superior passive reinforcements.

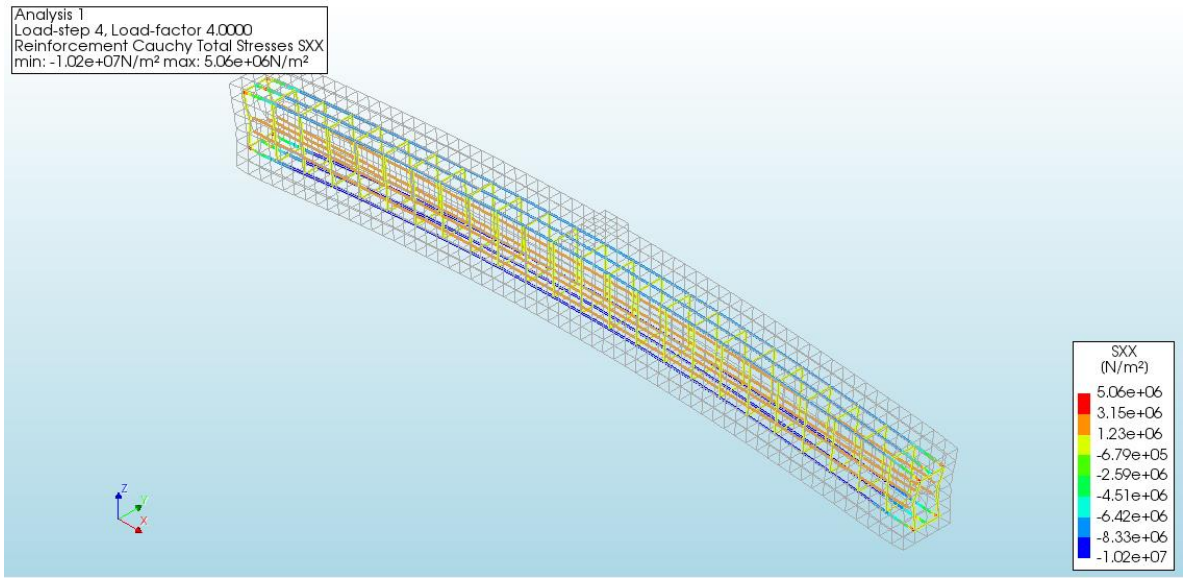

Figure 37: Total stress of the reinforcement in the X direction on the 4th load-step

In *Figure 37* it can still be noted that the beam is under compression because of the shape it has acquired. Even though this will change, for the moment, and like before, the bars under the most stress are the active tendons.

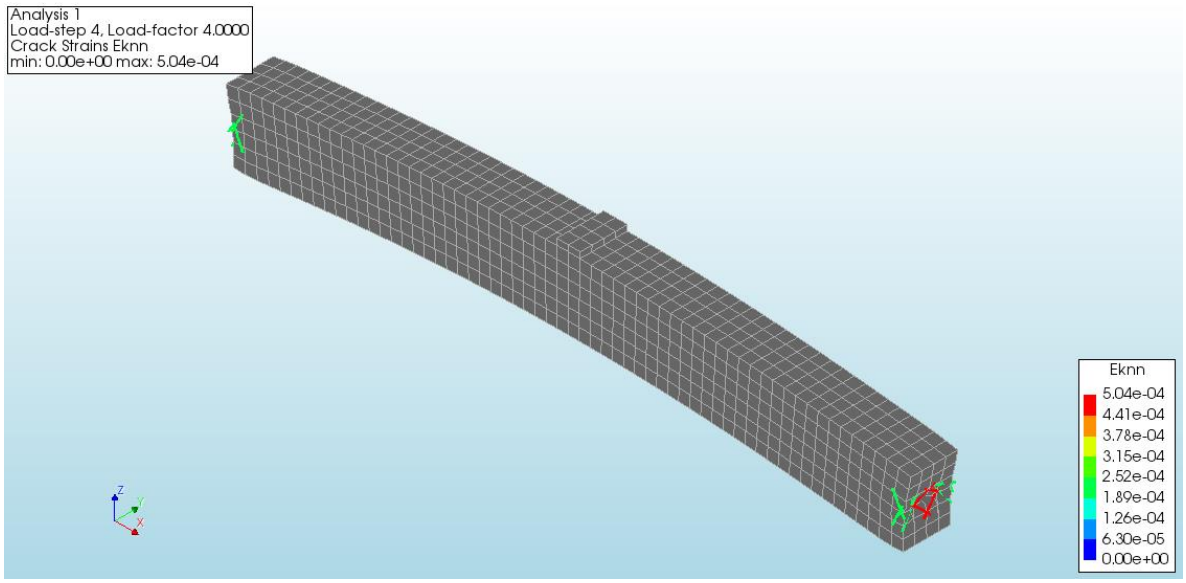

Figure 38: Cracks on the 4th load-step. Cracks generated by the post-tensioning load

Just like the anterior load-step, in this one the cracks due to the tension in the inferior part of the beam are yet to appear, which still means that the elastic regime is ruling.
It will take some more iterations to reach the nonlinear regime than the case studied before because of the post-tensioning load and the generated counter-deflection by it.

3.4.1.4. Demonstration of the linear behaviour (22nd-25th load-steps)

To prove that the behaviour of the beam is linear before the cracks on the inferior part start to appear, the displacements of load-steps 22, 23, 24 and 25 will be compared so to find the increase of the displacements in each iteration.

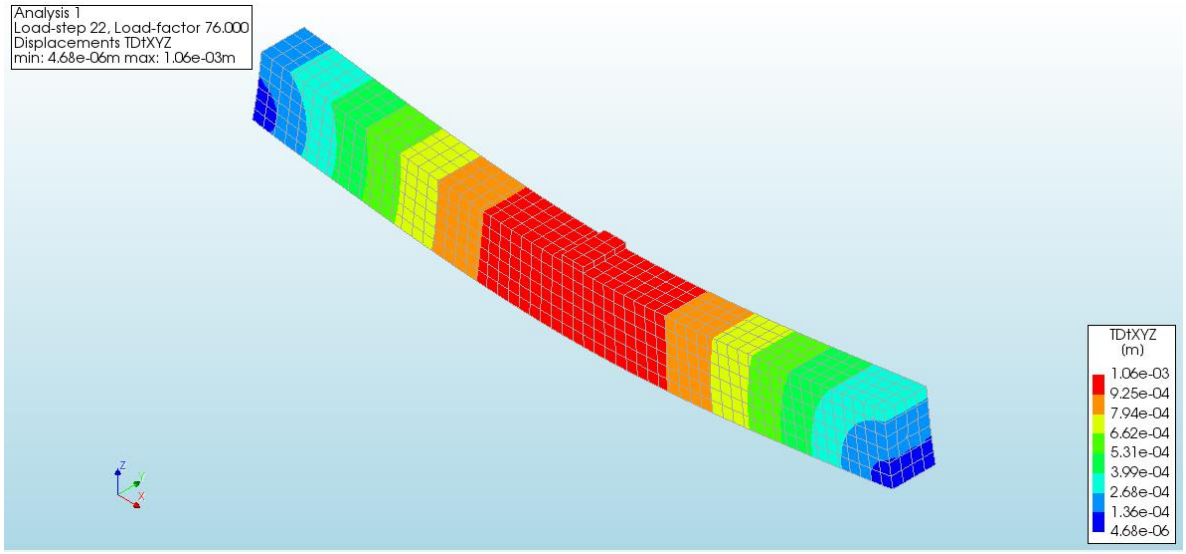

Figure 39: Total displacements in concrete on the 22nd load-step

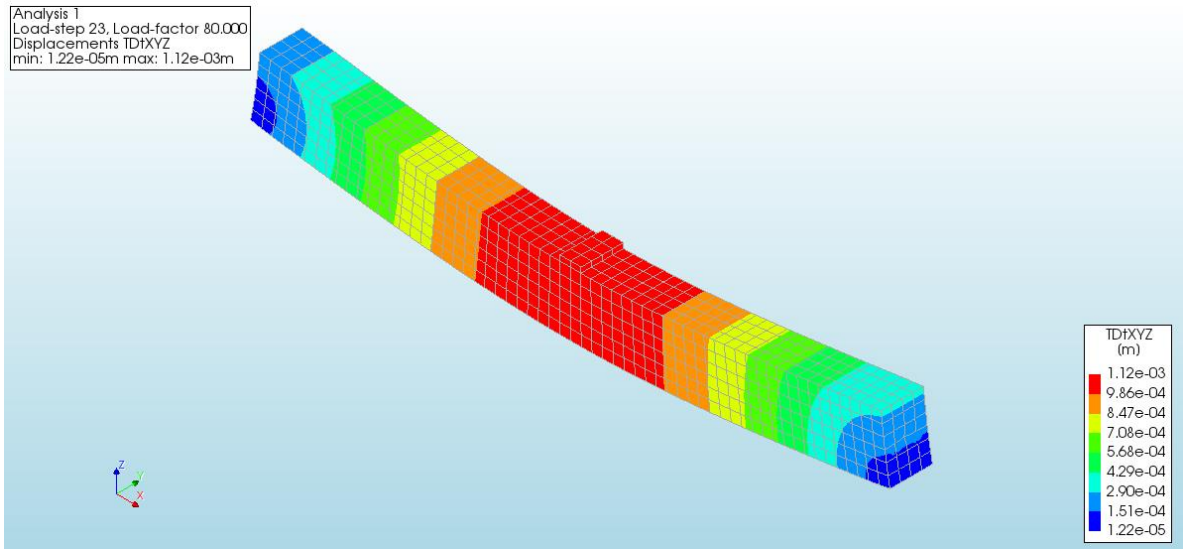

Figure 40: Total displacements in concrete on the 23rd load-step

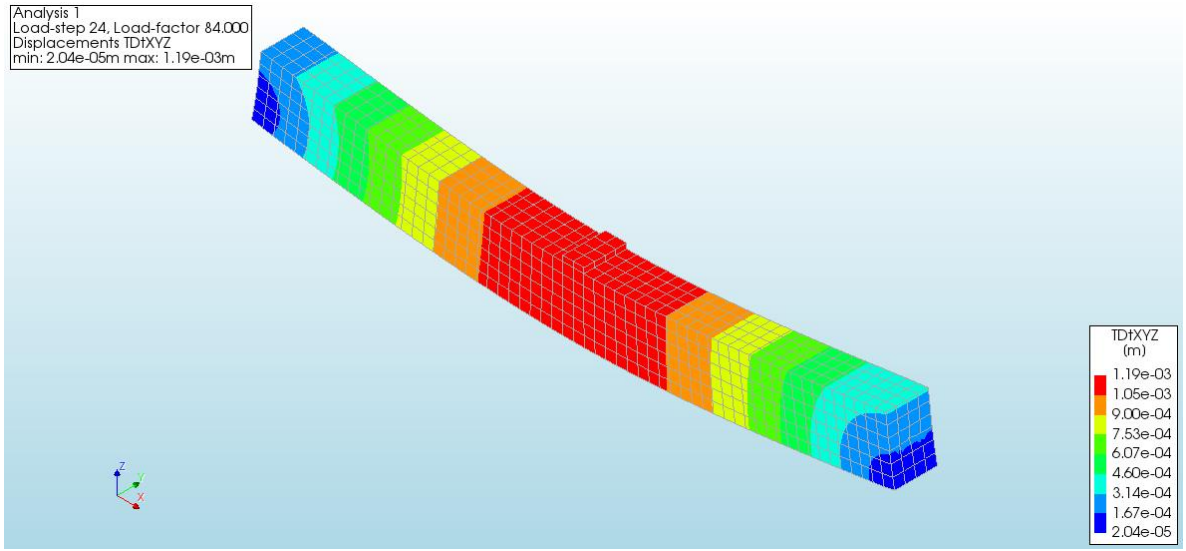

Figure 41: Total displacements in concrete on the 24thload-step

The increase in the displacements between *Figure 39* and *Figure 40 is* 0,06mm and between *Figure 40* and *Figure 41* is 0.07mm, which is nearly the same amount. Thus, there can be a way to approximate the value of the maximum displacement of the 25th load-step, just like in section *2.8.1.4 Demonstration of the linear behaviour (14th load-step)*, but with some differences.
To do the calculations, the biggest displacement increase (0.07mm will be taken).

$$Displ_{\max n\ loadstep} = Displ_{n-1\ loadstep} + \Delta_{displ}$$

Equation 5: General equation to estimate the maximum displacement of the following load-step on linear regime

$$Displ_{\max 25th\ \ loadstep} = Displ_{24th\ \ loadstep} + \Delta_{displ}$$

Equation 6: General equation to estimate the maximum displacement of the 25th load-step on linear regime

$$\boldsymbol{Displ_{max\ 25th\ \ loadstep}} = 1.19mm + 0.07mm = \mathbf{1.26mm}$$

Equation 7: Equation to estimate the maximum displacement of the 25th load-step on linear regime

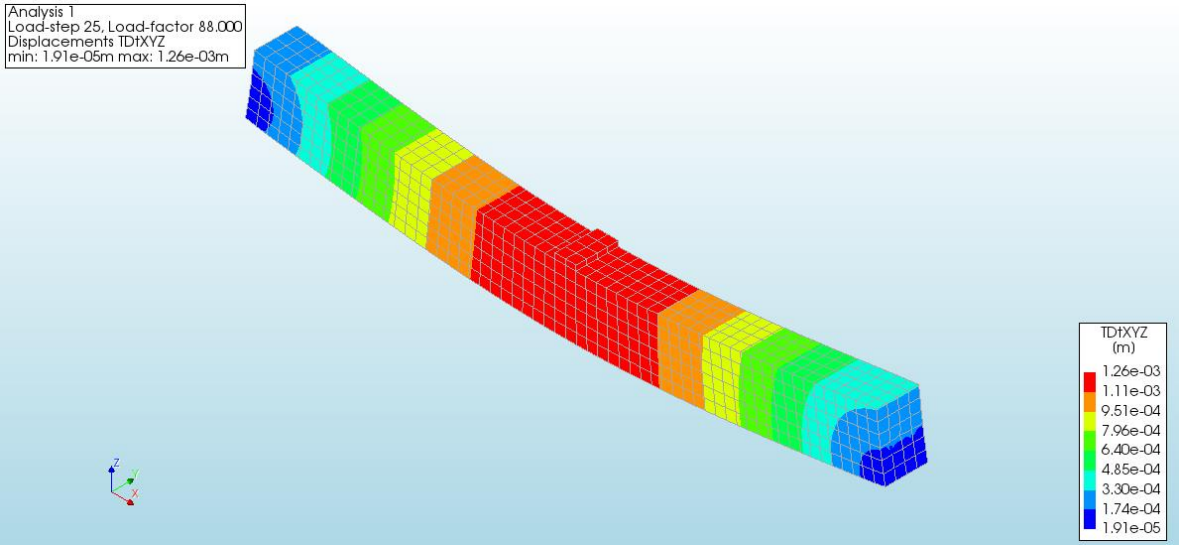

Figure 42: Total displacements in concrete on the 25th load-step

The result in *Equation 7* is exactly the same as the maximum displacement value in the legend of *Figure 42*, which means that the approximation is reliable and the linear regime is still present.
If the same thing was done with the other load-steps there are some possibilities that there was some error when comparing the equation result with the simulation one, but all the results would still be close to the calculated ones, as long as the linear regime is there.
As commented before, the exact same thing can be done with the stresses of the reinforcement.

3.4.1.5. 31st load-step

This is the very step before the beam starts cracking and, therefore, the last one with linear behaviour. Same as done in the anterior cases, the greatest displacement value can be approximated but another equation is needed, since the displacement of the 30th load-step is unknown in this document.
Another variable will be introduced to find out how many times the increase has to be multiplied by. A step in which we know the maximum displacement and the increase will be needed to carry on with the approximation.

$$Loadstep_{difference} = Present_{loadstep} - Model_{loadstep}$$

Equation 8: General equation to determine the load-steps difference

Where the model is the one in which the displacements are known and the present the one being studied now. The model that will be used is the 22nd load-step.

$$Loadstep_{difference} = 31 - 22 = 9\ steps$$

Equation 9: Equation to determine the load-steps difference

Once the difference has been calculated, the last thing to do is multiply this number by the displacement increase between two consecutive load-steps.

$$Displ_{\max present\ loadstep} = Displ_{max\ \ model\ loadstep} + \ Loadstep_{difference} \cdot \Delta_{displ}$$

Equation 10: General equation to estimate the maximum displacement of any load-step in the linear regime

$$Displ_{\max 31st\ loadstep} = Displ_{max\ \ 22nd} + \ Loadstep_{difference} \cdot \Delta_{displ}$$

Equation 11: General equation to estimate the maximum displacement of the 31st load-step on linear regime

$$\boldsymbol{Displ_{\max 31st\ loadstep}} = 1.06mm + \ 9 \cdot 0.07 = \boldsymbol{1.69mm}$$

Equation 12: Equation to estimate the maximum displacement of the 31st load-step on linear regime

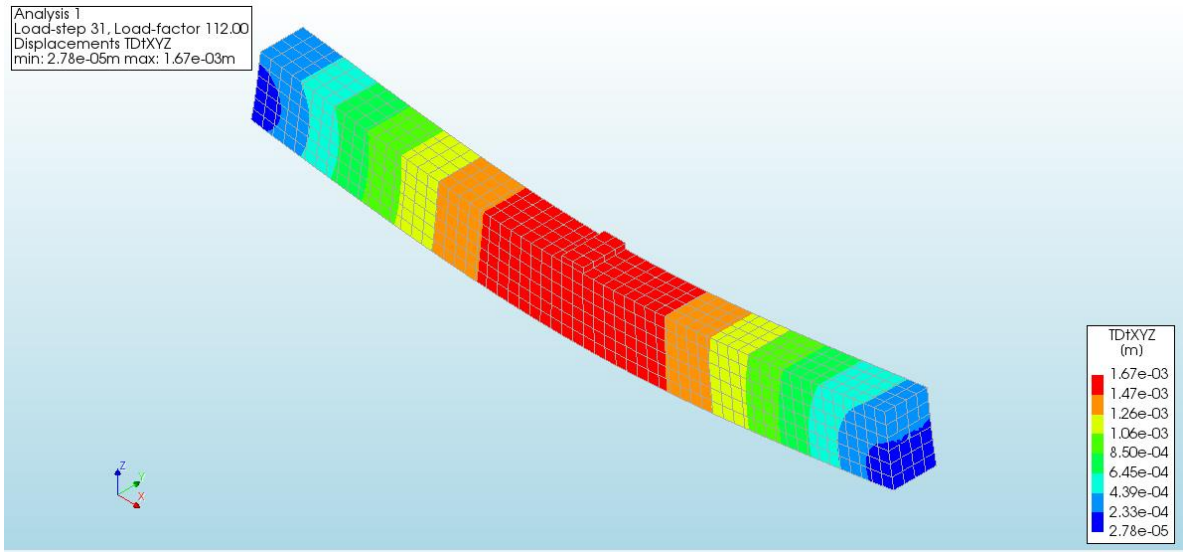

Figure 43: Total displacements in concrete on the 31st load-step

The result obtained in the simulation is 1.69mm instead of 1.67mm but, taking into account that the results are expressed in millimetres, the difference is so small that it does not imply any change and, therefore, the approximation can be considered accurate.

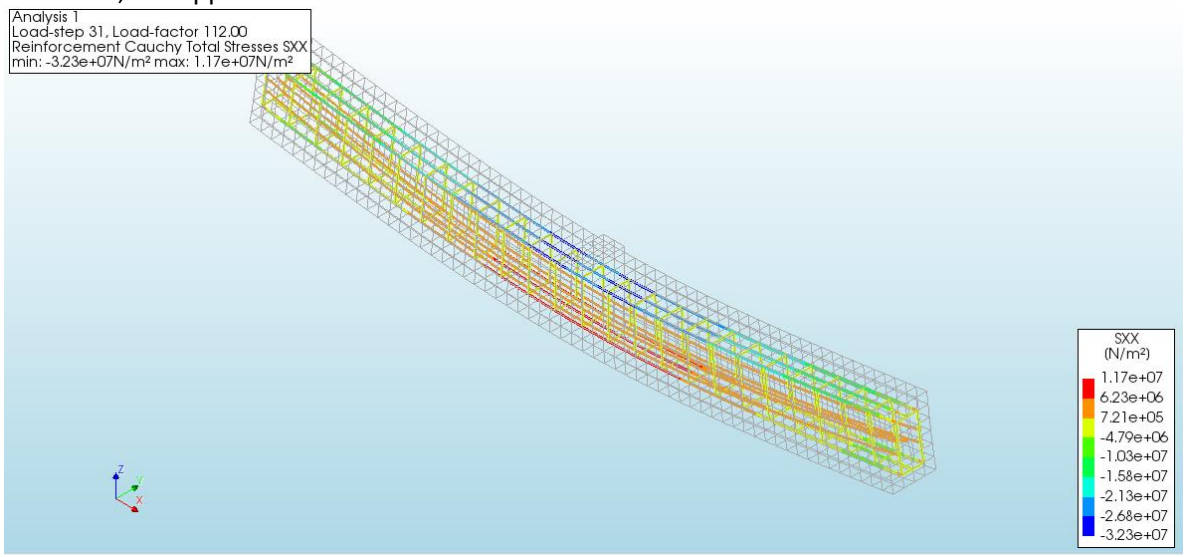

Figure 44: Total stress of the reinforcement in the X direction on the 31st load-step

Some approximations could be also done to the stresses in the reinforcement, but it would be exactly the same equations studied before, with the unique difference of changing displacements for stresses.

In *Figure 44* it can be noted that the post-tensioning tendons are no longer the ones under the greatest tension and the ones substituting them now are the inferior passive rebars. That is because the beam is under a point-load of 112000N that causes a positive bending moment that subdues the inferior part of the beam to a great tension, making the rebar work really hard.

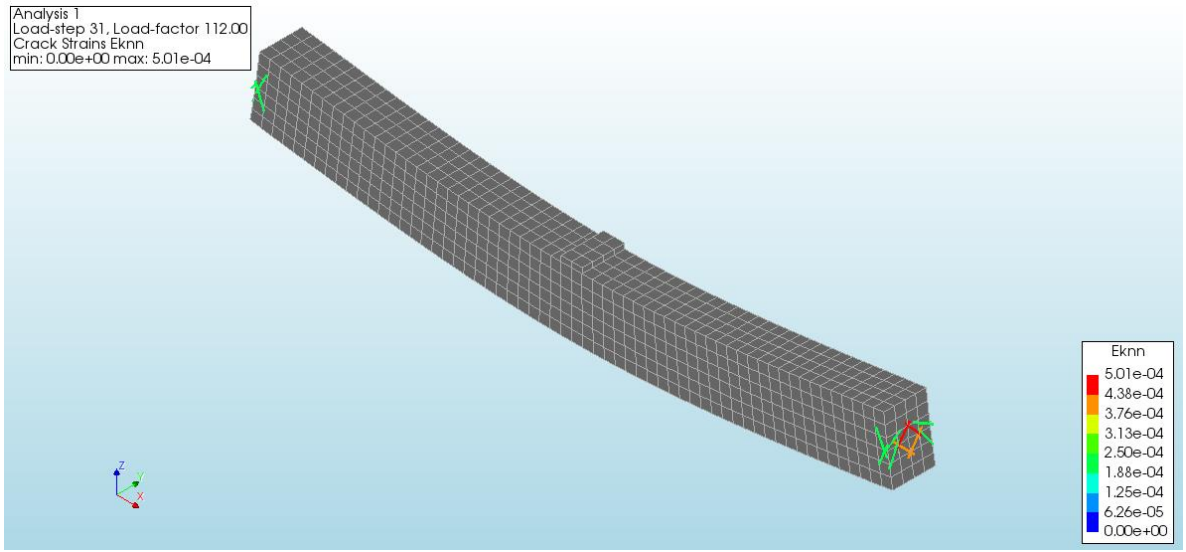

Figure 45: Cracks on the 31st load-step. Cracks generated by the post-tensioning load

Since the behaviour is still linear, the cracks on both ends of the beam do not have an impact on the linear regime itself. Once the cracks start appearing on the inferior part of the concrete, the only ones currently present will become negligible.

3.4.2. Nonlinear behaviour

3.4.2.1. 32nd load-step

As the title says, from now on the studied cases will no longer have a linear behaviour, which means that the displacements and stresses cannot be estimated.
It has taken 116000N to finally break the concrete, which obviously is a great load.

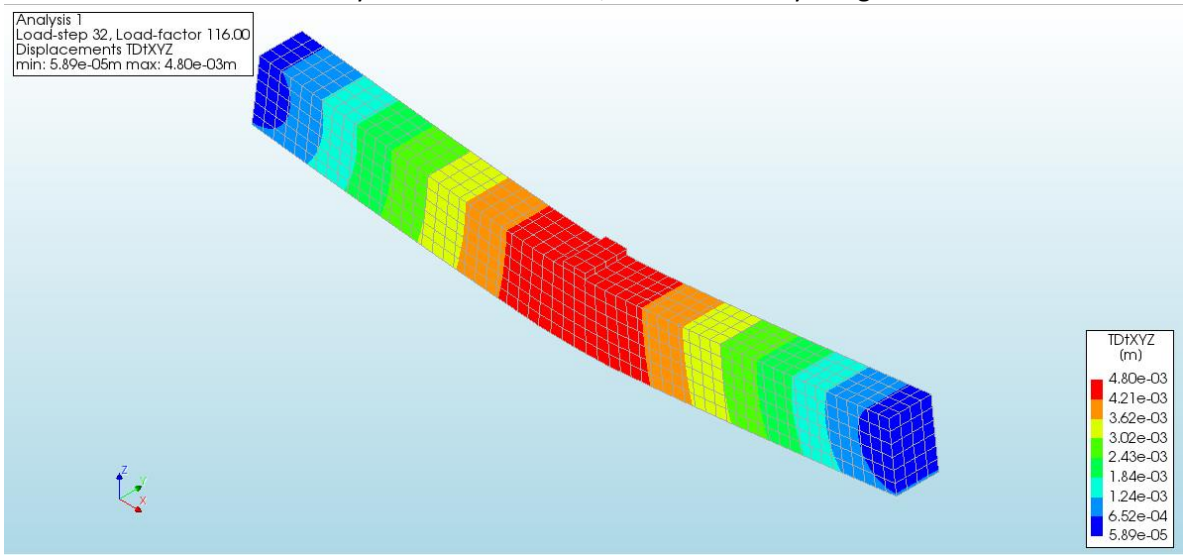

Figure 46: Total displacements in concrete on the 32nd load-step

If the greatest displacement in *Figure 43* is subtracted from the greatest in *Figure 46*, it can be observed that the increase now is way bigger than the one found in section *3.4.1.4 Demonstration of the linear behaviour (22nd-25th load-steps)*. This is clear proof that the behaviour of the structural element has changed because with the same load increasement, the displacement has become nearly 45 times greater.

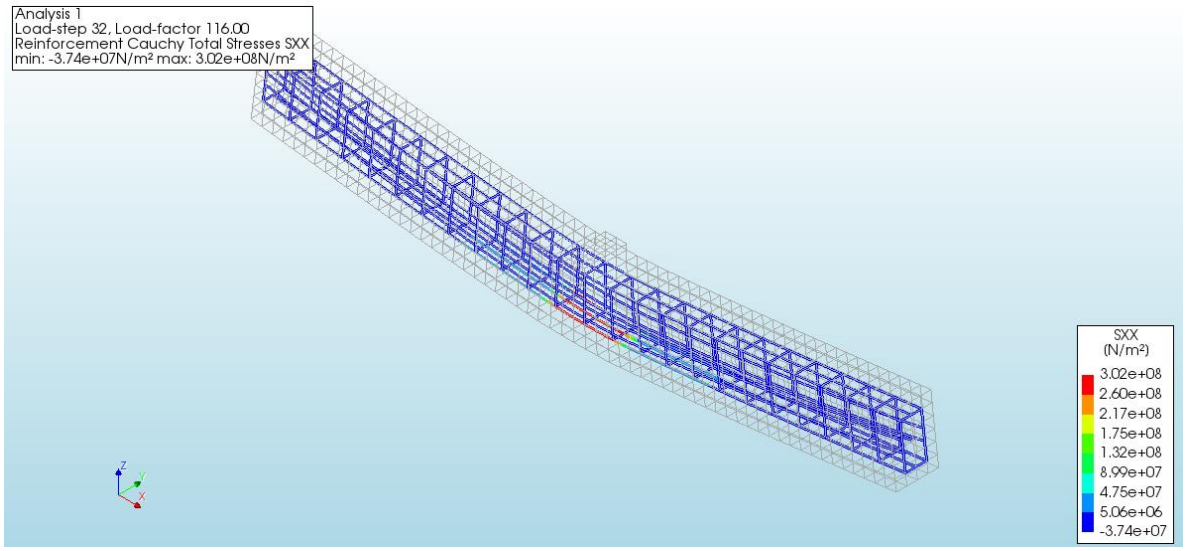

Figure 47: Total stress of the reinforcement in the X direction on the 32nd load-step

Once the concrete has cracked, the stresses in the reinforcement have a completely different distribution. The major part of it is under compression now, meaning that their stress value is negative. In contrast to the major part, the central inferior part of the rebar is in a huge tension due to the separation of the concrete in that zone. The function of the steel now is to keep the beam together and absorb all the tension the concrete is not able to.

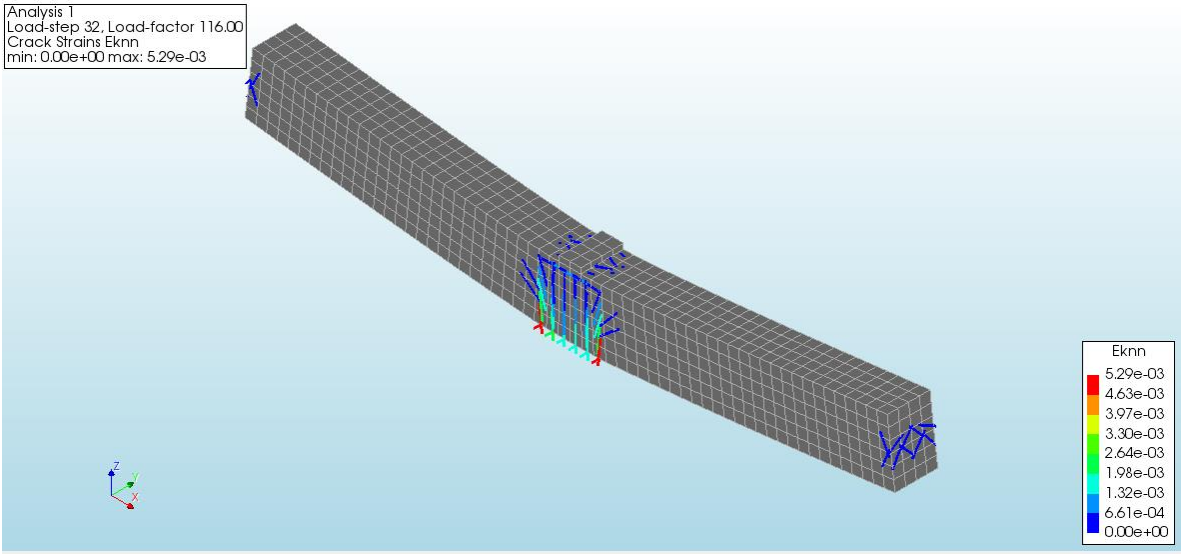

Figure 48: Cracks on the 32nd load-step. First time with cracks in the inferior part

Despite still being cracks in the ends of the beam, for the first time so far there are also cracks in the centre of it. Taking a look at the legend and the colours in the beam representation, it can be seen that the biggest and widest cracks are the ones in the middle, which allows foreseeing where the failure will take place in the end.

3.4.3. Failure

Even though it takes way more load-steps to reach failure with post-tensioning than without, in the end, it is reached too.
This time, the load-step that breaks the beam completely is the 49th, but there is an error in the simulation that cannot be fixed and, therefore, the result is different than in the real life.

3.4.3.1. 49th load-step

In the simulation with post-tensioning there is a problem when the failure is approaching. Both ends of the beam start deforming plastically and the displacements there are bigger than in the centre. The analysis has been run multiple times to try and solve the error and many things have been changed, but the beam still behaves the same way when the 49th load-step is reached.
Despite all the results obtained so far are trustworthy and make sense in every moment, the result that will be presented now is not.

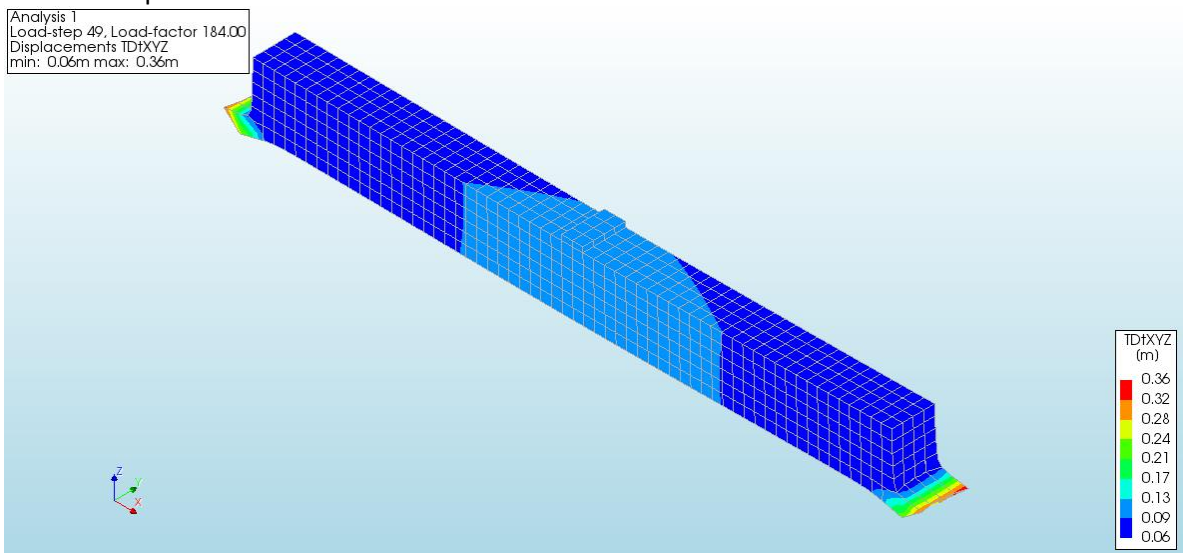

Figure 49: Total displacements in concrete on the 49th load-step

Even though the greatest displacements take place in the ends of the beam, the ones that are in the centre are also pretty big, which means that the beam, despite not appearing in *Figure 49*, is really bended.
The figures of the stresses of the reinforcement and the cracks will not be included in this section, since the error keeps the results from being reliable and would not make any sense to put and comment on them.

4. Load-displacement diagrams

Having already analysed the results and drawn the necessary conclusions, in this section two graphs will be presented that will help to see the behaviour of the beam at any moment of the process.
On the Y axis of the graph will be the load applied, which increases by 4000N every step, and on the X axis will be the displacement on the Z direction of the beam.
Knowing that the central load is applied on the negative direction of Z, the displacements will also be negative, but in these graphs, the used displacements are absolute values (always positive).

4.1. Without post-tensioning

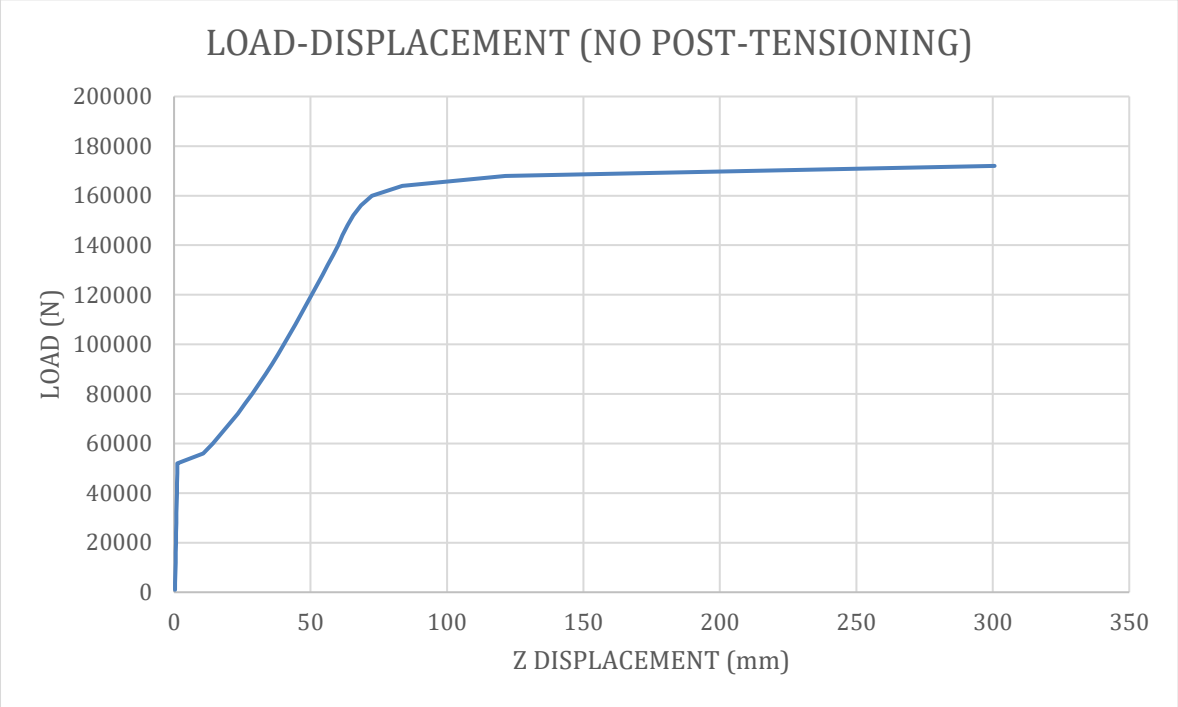

Figure 50: Load-displacement graph without post-tensioning

This graph could be divided into three sections:

- The first section is the linear part of the behaviour. It can be observed that the line has a linear tendency, even though it seems pretty vertical in the chart. This means that with every increase of the load the displacement increases exactly the same amount as before. If the slope of the line was calculated it would give the elastic modulus, also known as Young's Modulus.
  The displacements during this section are so small that cannot be seen with a naked eye.

- During the second one the behaviour is no longer linear and, for that reason, it is impossible to know the displacement that will take place with every step. It is important to remind that the load increases the same amount every time (4000N).

- The third section is also nonlinear but the difference between it and the one before is that now, with a little increase on the load the displacements grow extremely bigger.

The graph should also have a part where the load decreases and the displacements keep going up, but the analysis was designed so that the load could increase in factors of four but never decrease.

4.2. With post-tensioning

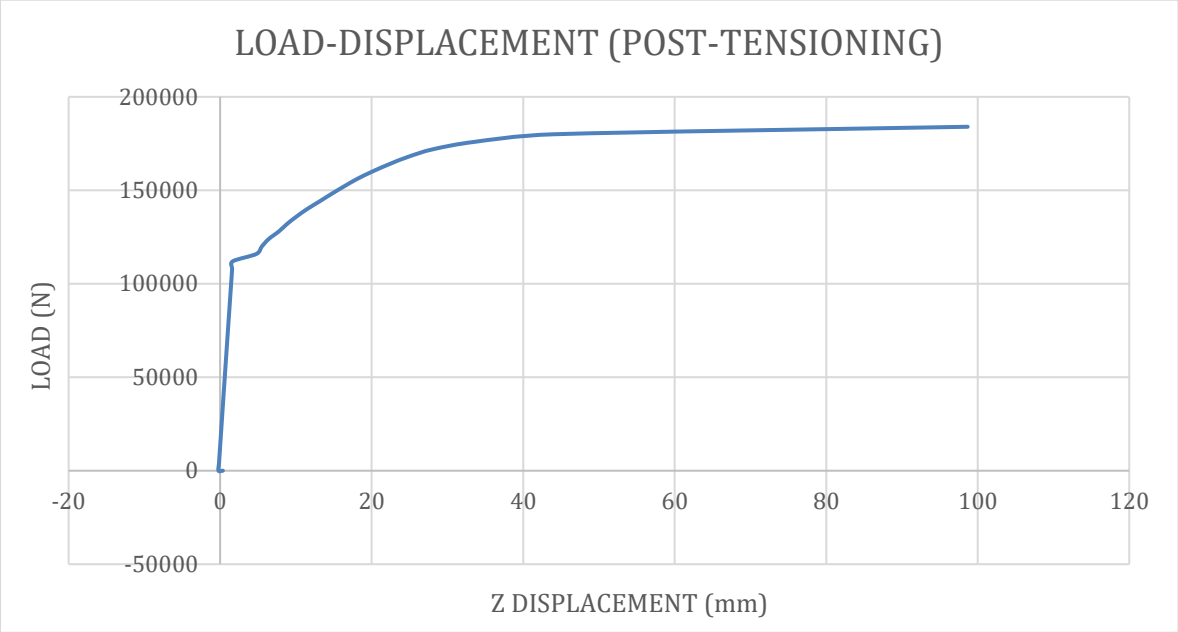

Figure 51: Load-displacement graph with post-tensioning

This graph could be divided into two sections:

- The first section, just like in *Figure 51*, is the linear part of the behaviour. Now the line does not look that vertical because the post-tensioning process allows the beam withstanding more load before the cracks start to appear and, therefore, the linear part is longer than the case before. As said before, the Young's Modulus value corresponds to the slope.

- Again, during the second section the behaviour is no longer linear and also the displacements become way bigger with every load increase.
  The displacements in this section can be spotted with a naked eye and could make people see that the beam is withstanding more load than it should and, therefore, warn them that something is going wrong. All that before the beam finally collapses.

This chart, just like the one before, should have a part where the load decreases and the displacements keep going up, but it just stops when the beam can no longer withstand the applied load.

5. DIANA options for the post-tensioning

Like any feature, the software used offers multiple options and in this section the ones concerning the post-tensioning will be studied and commented.

5.1. Bonding to the mother element

As commented in section *1. Introduction*, the tendons that work as post-tensioning cables are not in direct contact with the concrete because they find themselves encapsulated in a plastic duct.
Once the concrete has been cast and set, the tendons are stressed by pulling their ends through the anchorages and that will make them press against the concrete, compressing it and counteracting the external loads.
The next step is filling the ducts with grout so that the tendons are bonded to the ducts and, therefore, to the concrete. This process can or cannot be done, and that will determine the kind of post-tensioning method that is being used. The two methods are:

- Bonded post-tensioning
- Not bonded post-tensioning

The two of them have advantages over the other, but they also have some disadvantages. The most commonly used is the "bonded post-tensioning" mainly because it avoids the corrosion of the tendons, which can be a pretty worrying issue in the long run.
The biggest difference when it comes to the performance of the structural element is that when the tendons are bonded, the product E·I (which are the Young's Modulus and the moment of inertia of the cross section of the beam respectively), also called stiffness of the beam, increases. That would be the same as if the surface cross section had become a bit bigger and it improves certain structural behaviours.
When defining a material in DIANA, there is the option to bond it automatically to the mother element, which is the concrete beam. This option seems to be really handy but does not work at all for the active reinforcement. If the material is defined to be bonded to the concrete before the post-tensioning load is applied, it is like saying that the grout has been poured before the tendons are stretched. If the process is done in that order in the real world, the load cannot be applied because the tendons are completely adhered to the concrete and cannot elongate.
In DIANA, the load can still be applied, but when the analysis is run, it can be observed that the result is exactly the same as when there is no active reinforcement whatsoever.

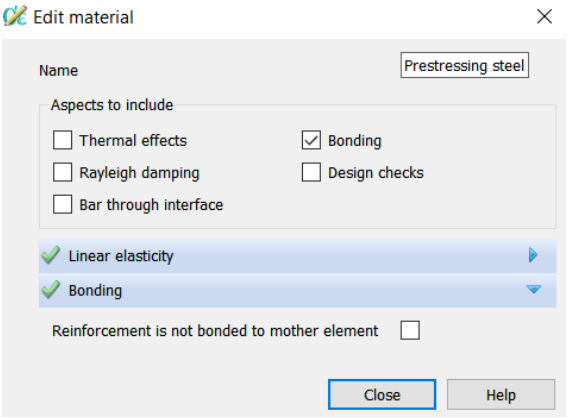

Figure 52: Active reinforcement bonded to mother element

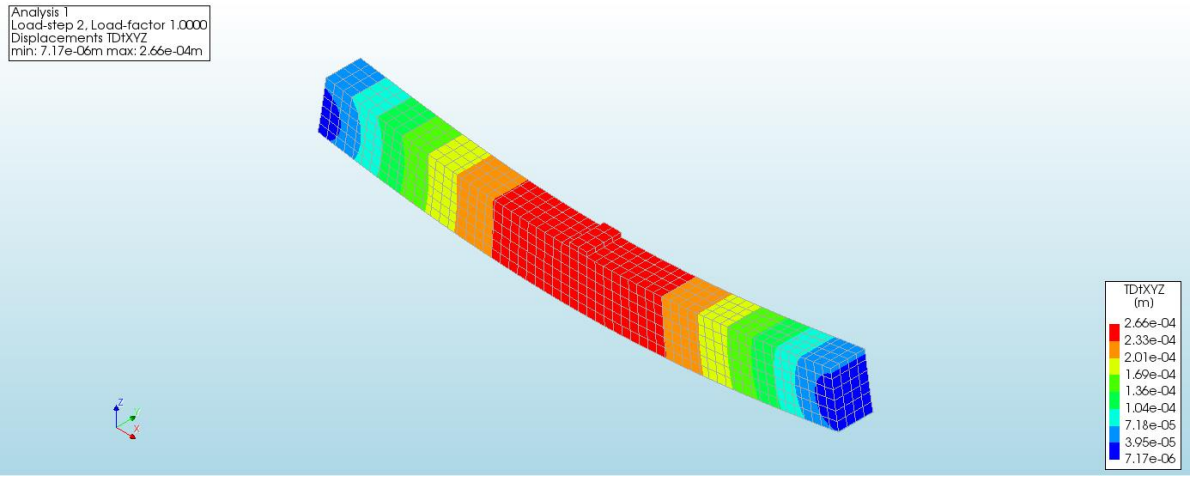

Figure 53: Total displacements in concrete on the 2nd load-step with bonding before the load is applied

In *Figure 53*, even though the second load-step is when the post-tensioning load is applied, it can be seen that there is no counter-deflection whatsoever, meaning that the explanation done before is correct and that there is no point on bonding the active reinforcement to the beam before applying the load.

5.2. Post-tensioning load applied in one or both ends

When it comes to applying the load, there are also two different options that can be studied and compared and these are reflected in the name of the section.
It is of much importance to know that there is friction between the tendons and the plastic ducts and, therefore there are losses, meaning that the load applied to the tendons is greater than the active force. These losses depend on the Coulomb friction coefficient, that depends on the same time on the steel used for the tendons and the plastic used for the ducts. This is thoroughly explained on Part 2 of the 5th chapter in *Betong-Konstruksjoner; Beregning og dimensjonering etter Eurocode 2* (Sørensen, 2013)[(7)].
Also, in this section appear different solutions to reduce the friction loss[4].

In short, applying the load in both ends can result helpful in reality to reduce the friction loss and, therefore, obtain better results with the same load that would have been applied in one end.
Both simulations have been run and the results will be compared.

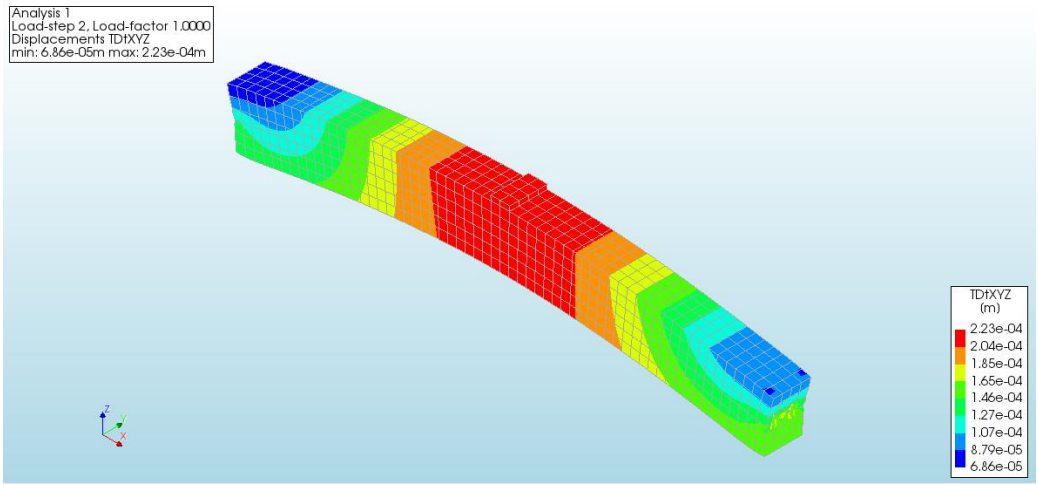

Figure 54: Total displacements with 80000N per tendon applied on one end

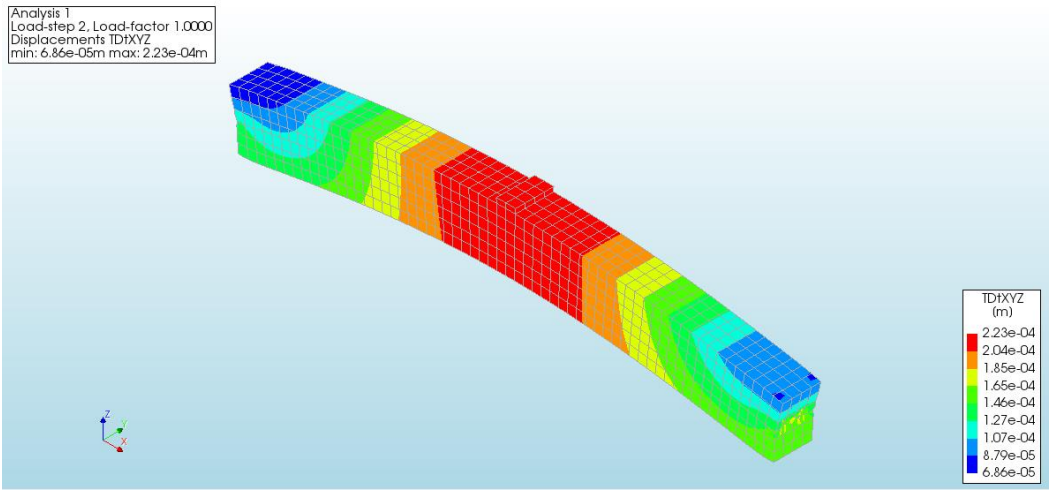

Figure 55: Total displacements with 80000N per tendon applied on both ends

[4] Besides the efforts to reduce the friction coefficient, there are other means to reduce the friction loss:
a) Reduce the curvature of the tendon
b) Apply the prestressing from both ends
c) Increased prestressing with subsequent slacking

When the load applied in both ends is exactly the same as the one applied in one end, and so is the friction coefficient, the obtained results in DIANA are exactly the same, as can be observed if *Figure 54* is compared to *Figure 55*.
This is not true if it takes place in reality because like Sørensen says <<*the losses from the middle to the passive end can be avoided if we are jacking from both ends*>>.
In conclusion, DIANA does not take into account the difference in friction loss if the load is applied on one end or both, meaning that it is exactly the same as long as the friction coefficient is the equal in both cases. For this exact reason, it is irrelevant whether in the simulation with post-tensioning the load is applied in on end or on both.

6. Comparison between non post-tensioned and post-tensioned beams

Post-tensioning is known to have a huge impact on the structural element's behaviour, not only when the post-tensioning load is applied to the tendons, usually giving place to a counter-deflection and especially if the element is a beam, but also when the external loads are applied to the beam itself, postponing the failure.

The main reason why the post-tensioning is used nowadays is to make the structural element work under compression as long as possible, knowing that that is the concrete's comfort zone.

In order to see the advantages of such technique and prove that it really has an impact on the beam, that is to say in the case studied in this document, a comparison table is going to be written. This one will allow knowing in a more visual way the differences between the two cases and will also help in drawing conclusions referring to their performances.

	No post-tensioning		Post-tensioning	
	Load	Maximum displacement(mm)	Load	Maximum displacement (mm)
1st load-step	Self-weight	0.365	Self-weight	0.365
2nd load-step	Point-Load (4000N)	0.432	Prestressing (80000N per tendon)	-0.223
3rd load-step	Point-Load (8000N)	0.501	Change of reinforcement properties	-0.223
4th load-step	Point-Load (12000N)	0.568	Point-Load (4000N)	-0.249
15th load-step	Point-Load (56000N)	10.7	Point-Load (48000N)	0.583
32nd load-step	Point-Load (124000N)	50.7	Point-Load (116000N)	4.8
38th load-step	Point-Load (148000N)	8520	Point-Load (140000N)	11.5
48th load-step	-	-	Point-Load (180000N)	360

STEPS WITH REGULAR DEFLECTION
STEPS WITH COUNTER-DEFLECTION
STEPS WITH CRAKS ON THE INFERIOR PART
FAILURE

Table 3: Comparison table along with its legend

If the anterior table is analysed, the conclusions that can be withdrawn are the following.

While in the case without active reinforcement the cracks on the inferior part of the beam start appearing at the 15th load-step, it is not until the 32nd that they start appearing on the one with post-tensioning. That is mostly because the bending moment created by the post-tensioning force, due to the eccentricity of the tendons, counteracts the self-weight and the point-load to an extent that it makes the beam experience a counter-deflection. That means that now the load applied on the centre of it has to be much greater in order to produce the same deformations than in the first case.

This counter-deflection starts appearing when the post-tensioning load is applied (2nd load-step in *Table 3*) and lasts until the 7th load-step, when the beam starts bending just like it should when a load is applied to the centre.

Post-tensioning allows the beam withstanding double the force that it can without this procedure, which proves that it is really working and fulfilling its purpose. To put it in numbers, while the 1st case only withstands 56000N before the cracks start appearing, it is not until 116000N that they appear in the 2nd case. It is more than two times the load.

It can be observed that the displacements when failure happens in both cases are excessive, but that simply means that the beams have completely collapsed and fallen apart.

The load-step difference of failure between both cases is 10, which is worthy of consideration and really puts in evidence the huge improvement in performance of the post-tensioning.

7. Modelling in DIANA FEA with Python

Taking into account that Python is a programming language, it is of vital importance to use a proper environment so that some help is given when the syntax is incorrect or when some commands are completely unknown, which is really likely in the beginning. Since the programming language, as mentioned before is Python, the environment that will be used is PyCharm[5]. It should be downloaded from the Internet, together with Python. When both the environment and Python have been downloaded, the procedure can get started.

After some research it has been observed that there are no libraries for the DIANA FEA, which means that no help will be received from the environment when using the commands that are responsible for the actions in the modelling software. Taking into account that the syntax is really complex and is absolutely impossible to know by heart, it is strongly recommended to first model a beam with the reinforcement (both active and passive), the loads, the supports, the mesh, etc. Knowing that every document created in the DIANA saves a script in the computer with the code of the commands used, it is highly recommendable to use it and write some improvements to automate the process. The commands of DIANA, such as creating a block or a polyline, will be copied and pasted so that the syntax is correct and then the script will be changed, including classes[6] to organize all the variables, loops[7] to automate some processes and other functions that will be explained later. This two definitions are found in the Python support web ("9. Classes — Python 3.8.2 documentation," n.d.)[(8)].

As a consequence of having three really different cases, one without post-tensioning, one with and one with but grouted poorly, there will be three different command scripts. Even though all the variables can be easily changed in the scripts, the default versions of the beams will be exactly the same as the ones studied in this document. That includes beam measures, material properties, applied loads, mesh properties, restricted movements, etc.
A decision has been made when it comes to organizing the script in order to make it clearer to read and understand. There is a command that allows linking two or more scripts, taking all the variables declared in one and passing them over to the script where they are called, in which all the DIANA commands are found. This makes the script way tidier and organized.
All in all, there will be five scripts, one where the variables are declared for the case without post-tensioning, one exactly the same but for the post-tensioning and the three others mentioned before.

In order to see how the classes work and help keeping the script tidy and organized some examples will be given, not only on how to create them but also how to use them once they have been declared.

[5] PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. *https://www.jetbrains.com/es-es/pycharm/* (link to download PyCharm)
[6] Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. *https://www.python.org/downloads/* (link to download Python)
[7] A loop in a computer program is an instruction that repeats until a specified condition is reached.

7.1. Definitions script without post-tensioning

In this document all the variables needed to create a beam without post-tensioning will be declared. In this script the values of the variables will be introduced. It is a super user-friendly way to change them to model all kinds of beams with different material properties, loads, etc.

This is where the classes explained before come into play. It has been considered necessary to declare a big number of classes, a total of 14, because there are a lot of features to be taken into account and not distinguishing between them could become a mess when writing the command script that will be executed by DIANA.

In order to see what kind of classes have been declared and their roles inside the script, a table will be included naming them all and explaining what kind of variables they contain, allowing to see the bigger picture and getting a better idea of all the properties needed in the 3D modelling.

Classes	Variables
Beam	Span, width, depth, name…
SteelPlate	Span, width, depth, name…
Point	Coordinates (x, y, z), name
Covers	Vertical, horizontal
Rebar	Distance between them, coordinates
Stirrups	Distance between them, coordinates
Loads	Self-weight, point-load
Categories	Beams, plates, rebar, stirrups
Materials	Concrete, steel, reinforcement steel
Supports	(x, y, z): Translation, rotation
PrincipalAxis	x, y, z
FactorCombination	c1, c2, c3
MeshProperties	Size
Analysis	Name, type, load combination…

Table 4: Classes and their respective variables on the script without post-tensioning

All the necessary classes can be seen in *Table 4*, but there are lots of variables that still do not appear. The ones that are present are considered to be the most important ones and help getting a general idea of the ones that are not there.

Having everything organized in this classes will really help when the command script is being written because all the variables are included in bigger categories. This means that, even though there are two types of spans or widths, since they belong to different classes, the way to call they will also be completely different.

To illustrate that with an example, the span of the beam and then the span of the steel plate will be declared, and that will demonstrate how to call them afterwards.

Beam.span = 6

SteelPlate.span = 0.2

It can be observed that the first word used to declare the variables is the class in which they will be included and the word that comes after is the name of the variable itself.

7.2. Definitions script with post-tensioning

This case is practically the same as the one before but with some modifications because of the presence of the post-tensioning, meaning that most of the script can be exactly copied, but some additional classes and variables will be needed.

If a look is taken at the script without post-tensioning, it will be seen that the classes declared deal with all the properties and features about the beam and its components and the only thing missing, obviously, is the active reinforcement and the application of the post-tensioning load.

Considering that there is already a class for the loads applied, this means that only the variable of the post-tensioning load is needed, and so the number of classes declared will only increase by one.

As did in section *7.1 Definitions script without post-tensioning*, a table with all the classes used in the script will be included, to be able to make a general idea of its content and the differences between both of them.

Classes	Variables
Beam	Span, width, depth, name…
SteelPlate	Span, width, depth, name…
Point	Coordinates (x, y, z), name
Covers	Vertical, horizontal
Rebar	Distance between them, coordinates
Stirrups	Distance between them, coordinates
ActiveReinfo	Vertical cover, horizontal cover
Loads	Self-weight, point-load
Categories	Beams, plates, rebar, stirrups
Materials	Concrete, steel, reinforcement steel
Supports	(x, y, z): Translation, rotation
PrincipalAxis	x, y, z
FactorCombination	c1, c2, c3
MeshProperties	Size
Analysis	Name, type, load combination…

Table 5: Classes and their respective variables on the script with post-tensioning

Undoubtedly, for the “ActiveReinfo” class, since the distribution of the active reinforcement is not straight all along the beam and therefore, the software needs to have different points at diverse coordinates to create the lines that will become the reinforcement. Hence, this is the class were more variables are needed.

Just as it was done with the two types of reinforcement before (passive reinforcement and stirrups), the geometry of the active will have to be declared too, alongside with its material properties such as the Young’s Modulus.

It is known that this type of reinforcement needs to withstand a greater load than the rest because the latter is applied directly to it, making it obvious that the diameter of the bars will be the greatest too.

7.3. Command script without post-tensioning, with post-tensioning and with poor grouting

There will be no difference between the case without post-tensioning and the one with in this section, because the explanation is about the general obtention of writing of the script, leaving aside which commands includes every one of them.

It has been said before that the crucial part of the command script is directly obtained from the document saved on the computer when using the DIANA interface. That is, it gets automatically saved with all the commands that have been used, including view changes and whatnot.
Since the syntax is pretty complex and there is no environment that will correct the typos, a case of a simple beam, first without active reinforcement and then with, will be modelled in DIANA and all the commands will be copied from the document directly saved in the computer to the Python environment, so that no mistakes are done with the commands.
All the classes and variables have been already declared in the documents (scripts) explained in sections *7.1 Definitions script without post-tensioning* and *7.2 Definitions script with post-tensioning*, so the only thing remaining now is using them in the command script. Furthermore, some changes will be included in the latter so that it becomes simple and clear to read, and allows the user working comfortably despite the number of variables.

Logically, the bigger the beam, the greater the amount of reinforcement. This aspect has to be taken into account when modelling the beam with DIANA interface but also has to be automatically dealt with when the script is used. Since the measures of the beam can be changed by the user, some automatic algorithms have to be introduced in order to change the amount of reinforcement according to the new measures. That will need the use of loops, as explained in section *7. Modelling in DIANA FEA with Python,* to take the measures declared by the user and adapt the amount of reinforcement according to the new measures. This will just be done with the passive reinforcement because the active will always be the same.

Also, there are some folders in the software's interface that include the blocks created (beam and steel plate) and the lines and polylines (active and passive reinforcement). In the beginning there is only one folder that includes it all, so the loops will also be used to, once the other folders needed are created, put all the blocks, lines and polylines at their place. this will allow selecting them if they have to be hidden or shown if, for example, just the reinforcement wants to be seen.

If the script automatically created by DIANA is observed, it can be seen that when this process is done but it takes a line of code every time that an element has to be moved from one folder to another. These loops will help to brief the code and making the script more visual and easier to understand, adapting this process also to the measures and the properties of the materials inputted of the user.
In these loops, a counter will be used to control the number of times the iteration has to repeat itself and it will be redefined every time that a new loop takes place so that no more variables than needed are used.

7.4. Running the Python script in DIANA

Once both the variable and the command scripts have been written, they can be run from the software DIANA so to do all the modelling.

There are two ways of modelling the structural element with the script:

- Copy and paste the script to the command console on the inferior part of the interface.
- Open the script with the option “run saved script”.

The one that will be used is obviously the second one because it is way faster and more comfortable. To run the script, the only thing that has to be done is click on the option “File”, then click on “Run saved script” and finally select the file that has to be run. In this project, the files that will be selected are *no_Prestress* and *Prestress*.

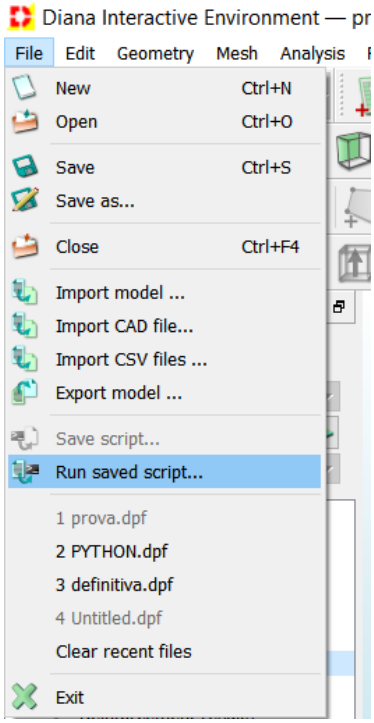

Figure 56: How to run a saved script

All the scripts are included in the *Attachments*.

8. Explanation of the most difficult parts

This section is designed to gather all the necessary information to enable everyone to go through all the steps to model the beam and run the analysis. It has been considered necessary to add this section because there are some processes that are not really clear in the DIANA manuals or in their video tutorials, and this will keep the people that need to do a similar model from struggling.

8.1. Load-cases and combinations

Even though the most logical thing would be adding the three loads in one combination, there is a problem when the load-steps are applied.

If they are all on the same combination, when the point-load is increased, so are the other loads, and this does not make any sense in reality. For that reason, the main goal will be making sure that there is a load-combination for every one of the applied loads, always with a factor of 1.

Although this section will be focused on the post-tensioning case, the exact same thing can be done with the case without. The only difference will be that instead of three loads and combinations there will only be two.

On the first place, and knowing that post-tensioning is present in this section, the three different load-cases have to be applied. When introducing the load-cases and their respective loads it is not important at all the order in which is done, as long as they are all included.

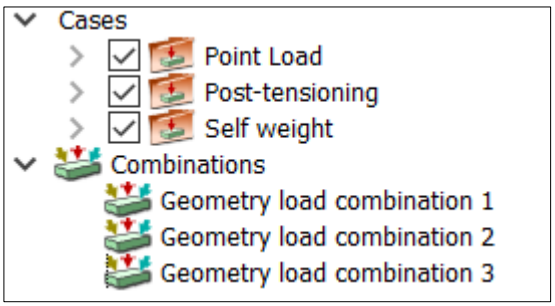

Figure 57: Box with the different cases and load combinations

If the number of load cases is three, so will be the number of load combinations. The thing that has to be done now, once the process reflected in *Figure 57* has been completed, is giving a factor to each one of the load cases. It is of the most importance that there is just one load applied in every combination because the wanted result is having three load combinations, each one of them with a different load case.

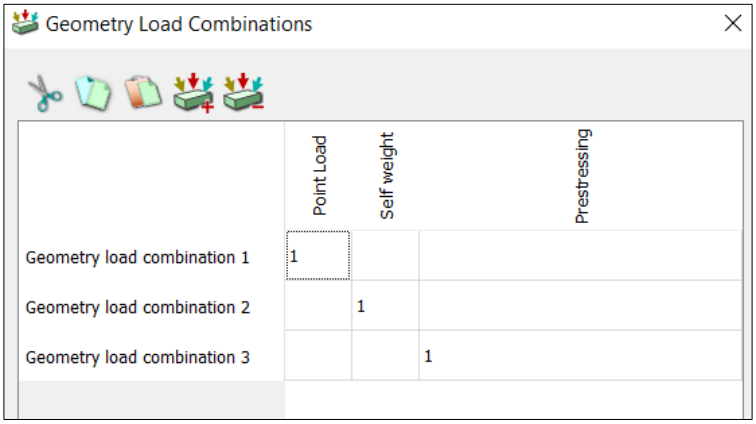

Figure 58: Box with the load combinations and its factors

8.2. Mesh

One of the most important parts in a Finite Element Software is creating a mesh with the appropriate measures so that the obtained results are accurate and adjust to reality.

Even though it may seem that to mesh the structural element all its components have to be selected one by one (beam, steel plate, rebar, stirrups, etc.), this can just be done by selecting the main blocks that form it. For example, in this case just the concrete beam and the steel plate have to be selected and all the other elements will also be included, because they are inside of the beam and embedded to it, making it all a whole.

The meshing process is explained in different tutorials or other documents but there can also be the doubt of whether all the elements have to be selected or just the exterior ones. The answer is: if the elements that are not being selected are inside and bonded to the selected one, the mesh will also include them and, therefore, the analysis will be completely valid.

This fact will ease the ease the process of writing the script, because no loops will be needed to include all the reinforcement bars and just two elements will have to be selected.

8.3. Nonlinear analysis

Undoubtedly, this is the most important part of the whole process and needs special attention because some things may be not clear in DIANA documents or video tutorials.

In this section the procedure to run a nonlinear analysis will be thoroughly explained step by step in order to obtain the expected results, starting from creating the analysis, going through the application of all the loads, and finally running it.

8.3.1. Command box

To start with, in the box command on the inferior part of the screen there are different options: “Geometry”, “Mesh”, “Analysis”, etc. Since the analysis is the part that will be done now, this is the option that will be selected.

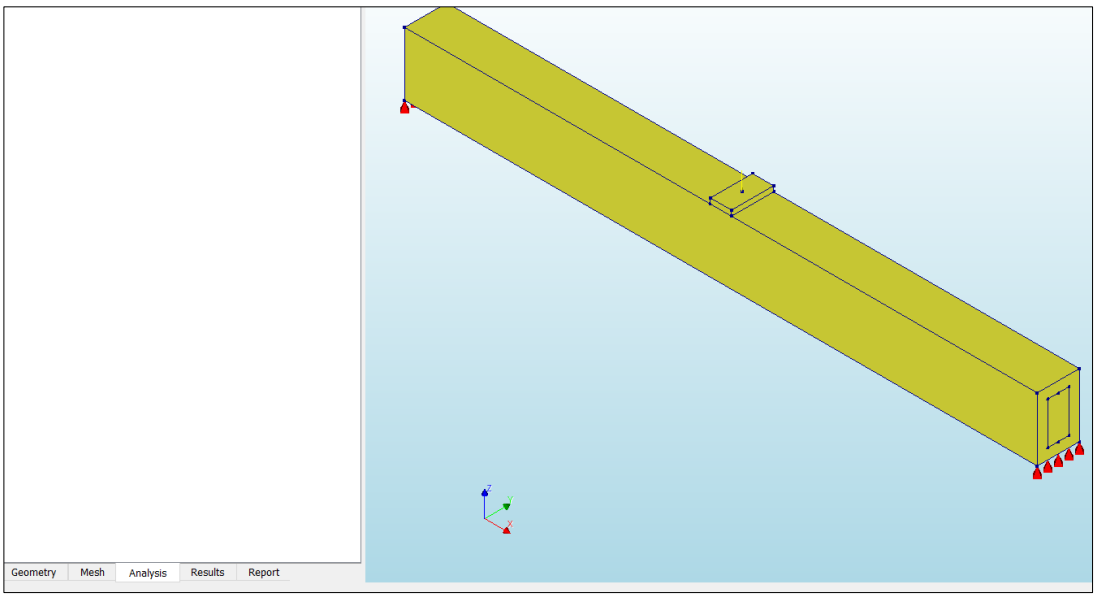

Figure 59: Command box with all the different options

In *Figure 59* it can be observed where the command box is and which is the option that has to be selected.

8.3.2. Creation of Analysis 1

To continue with this procedure, the next thing is creating an analysis. To do so, the icon on the left has to be selected and the name of it when created will automatically be "Analysis 1".

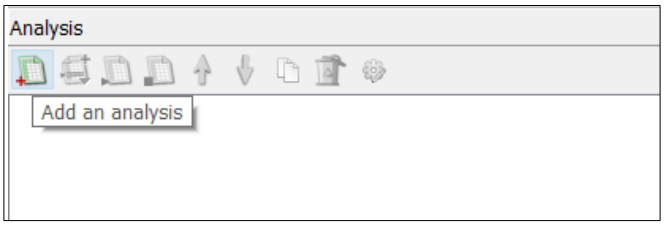

Figure 60: Creation of "Analysis 1"

8.3.3. Selection of the type of analysis

Once the "Analysis 1" has been created, the type of it has to be chosen and this will obligatorily be nonlinear. The part with a plus will be selected (see *Figure 61*) and the "Structural nonlinear" is the command that will be chosen.

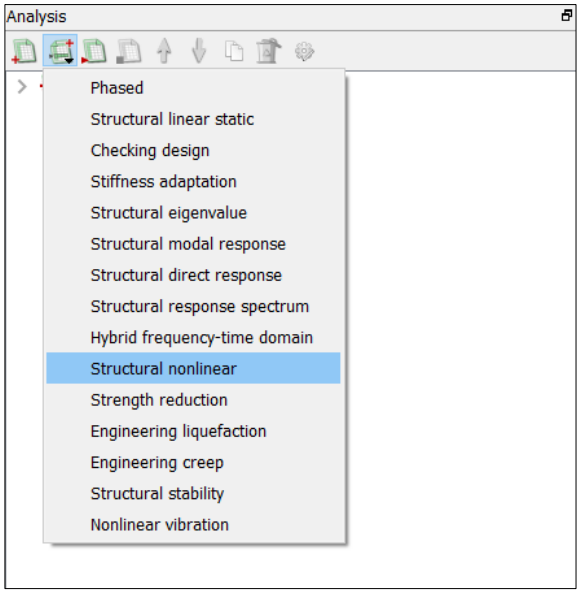

Figure 61: Creation of the nonlinear analysis

The next step is declaring the different load combinations that exist in order to apply one after the other, that is, the self-weight, the post-tensioning load, the change of properties and the point-load. The result that has to be achieved is:

- First, application of the self-weight of the beam (Load combination 1).
- Then, application of the post-tensioning load, in case there is active reinforcement (Load combination 2).
- After, change of material properties of the reinforcement. From not bonded to bonded, in case there is active reinforcement.
- Finally, application of the point-load and increase of it until failure (Load combination 3)

The difficult part in this section is that the displacements that the self-weight creates to the beam have to be the initial position when the post-tensioning is applied, and the same has to happen with the displacements of the post-tensioning when the point-load is applied. To summarise, the displacements of the anterior application of the load have to be the initial position for the next load case.

8.3.4. Creation of the first "Execute steps". Load combination 1

First, the self-weight case will be introduced. This will be done by changing the name of "new execute block" to "Self-weight". Then, the following step is to right-click to the option "Load steps" and select the load set, which in this case will be the "Load combination 1".

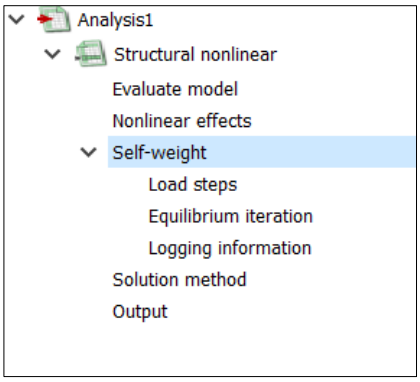

Figure 62: Menu where all the options concerning the load case can be modified

Since the load of the self-weight does not have to be modified or increased, no load-steps will be needed and, therefore, the "user specified sizes" blank will have to be filled with a number one. See *Figure 63*.

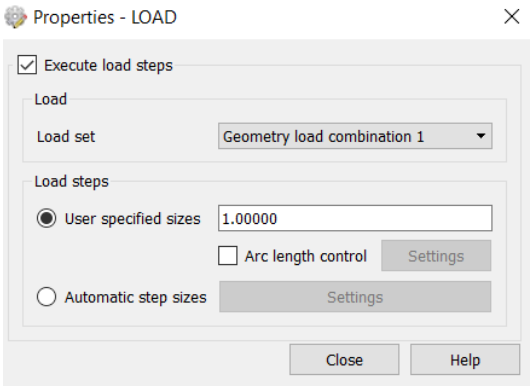

Figure 63: Application of the first load combination (self-weight)

8.3.5. Creation of the second "Execute steps". Load combination 2

The most important part of the process is now. In order to save the displacements of the anterior load-step all the load combinations have to take place in the same analysis and in the same order that they will be applied in reality.

To do so, a "Execute steps – Load steps" has to be added and this time it will be named "Post-tensioning".

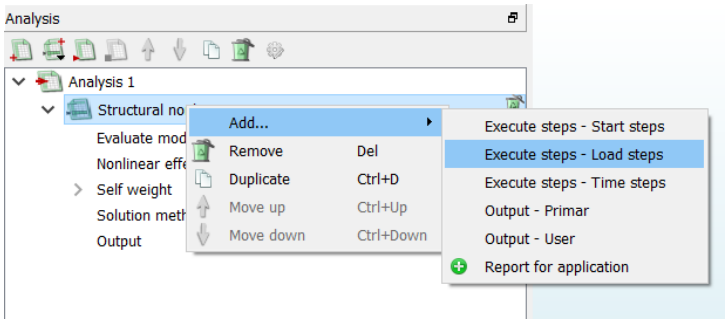

Figure 64: Addition of the second load combination

Now the load-steps have to be configured and, just like in *Figure 63*, even though now the load combination is number two, the blank space in the "user specified sizes" will be filled with a number one. This means that the post-tensioning load defined before (80000N) will not be increased during the application and that just one step will take place.

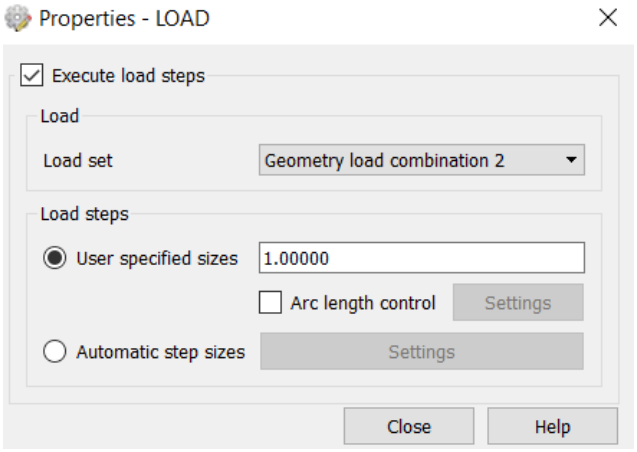

Figure 65: Application of the second load combination (post-tensioning)

8.3.6. Creation of the third "Execute steps". Step to change the reinforcement properties

As explained in section *5.1 Bonding to the mother element*, after the tendons have been stressed there are two possibilities: pouring grout so that the tendons bond to the concrete or not doing it. This will have an impact on the structural behaviour, also explained in the same section.

To do exactly the same process with DIANA, another execute step has to be added, and it will be between the application of the post-tensioning load and the point-load. It of great importance to realize that no load will be introduced in this step, because the only thing that has to be done is change the bonding property of the active reinforcement.

Firstly, the procedure explained in *Figure 64* has to be followed, and then the name changed to "Bonding to mother element". Now, the actions that appear in *Figure 65* will be done but this time the geometry load combination will be the first one (it does not change anything to use the second one or the third). Now the "user specified sizes" will be 0 because no load has to be applied during this change of properties.

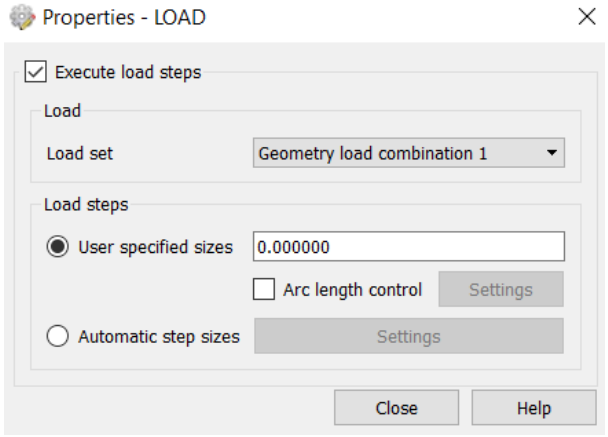

Figure 66: Application of the change of properties. Load = 0

Secondly, the option to change the properties has to be created. This will be done by right-clicking on "Bonding to the mother element", selecting the option "Add…" and finally clicking on "Physic nonlinear options".

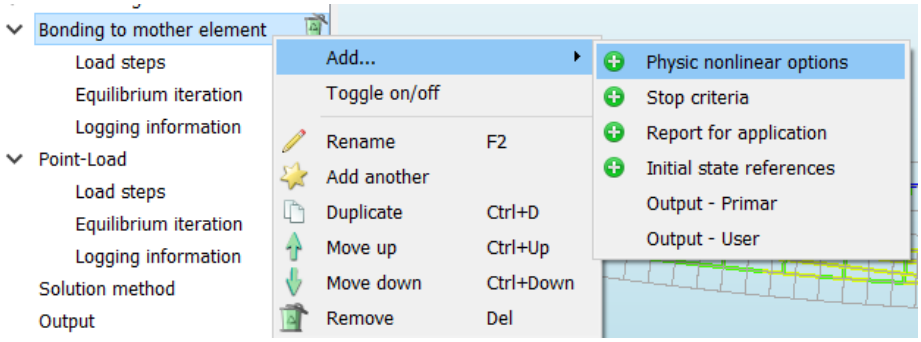

Figure 67: How to add the option "Physic nonlinear options"

Once the process in *Figure 67* has been done, the reinforcement that has to be bonded to the concrete has to be selected. Taking into account that all of it should be bonding because the post-tensioning load has already been applied, the option selected will be: ALL.

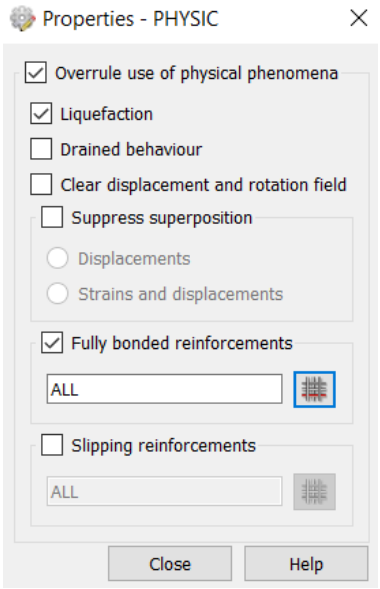

Figure 68: Selecting ALL the reinforcement to bond it to the concrete

This will also affect the active reinforcement because the passive was already bonded to the mother element.

8.3.7.3 Creation of the fourth "Execute steps". Load combination

The remaining step is applying the point-load and keep increasing it during some steps, that will also be defined in this section. The process when creating the "Execute steps" is exactly the same as the one that appears in *Figure 64* but now the name, when changed, will be "Point-Load".

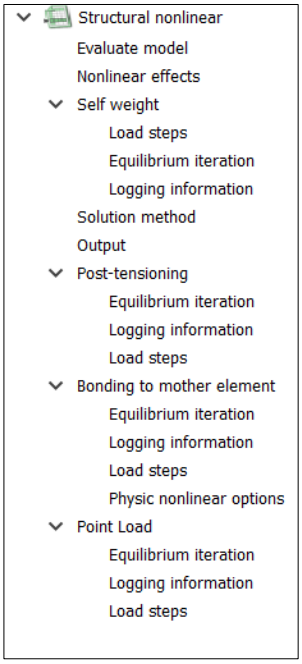

Figure 69: The four different "Execute steps"

The difference between this case and the other two is that now different load-steps are needed in order to increase the load until the beam collapses. Although the name of iterations that will be necessary to make this happen is unknown, a high number will be set to make sure that the failure is reached, and when the beam collapses the simulation will stop.

In "used specified sizes" (see *Figure 70*) and as explained in section *2.7 Application of the loads*, the number four means the factor by which the load will increase every iteration and the number sixty is the iterations that will take place.

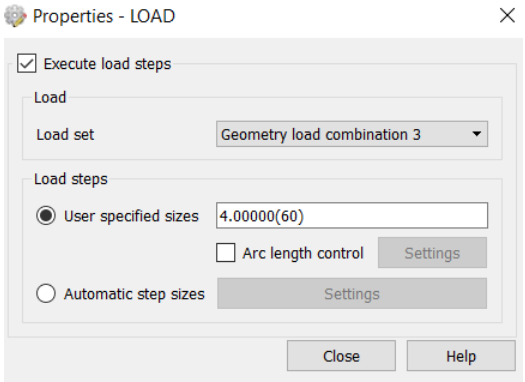

Figure 70: Application of the third load combination (point-load)

8.3.8. Equilibrium iteration

On the third load combination, two modifications will have to take place so that the simulation continues after the number of iterations for the calculation has been completed and no convergence has taken place. First, right-click to "Equilibrium iteration" on the menu that can be seen in *Figure 72*, then click on "Edit properties" and the menu on *Figure 71* will appear.

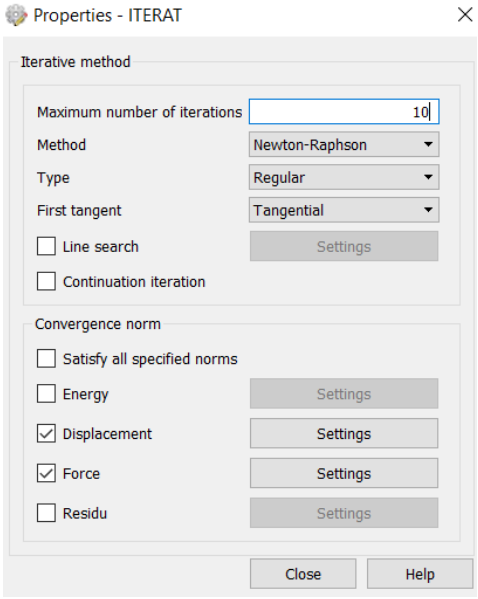

Figure 71: Maximum number of iterations when solving the equations

Once the menu has appeared, click on the "Settings" beside "Displacement" and then change the option "Terminate" for "Continue". The same thing should be done with the "Settings" beside "Force". This process is reflected in *Figure 72*.

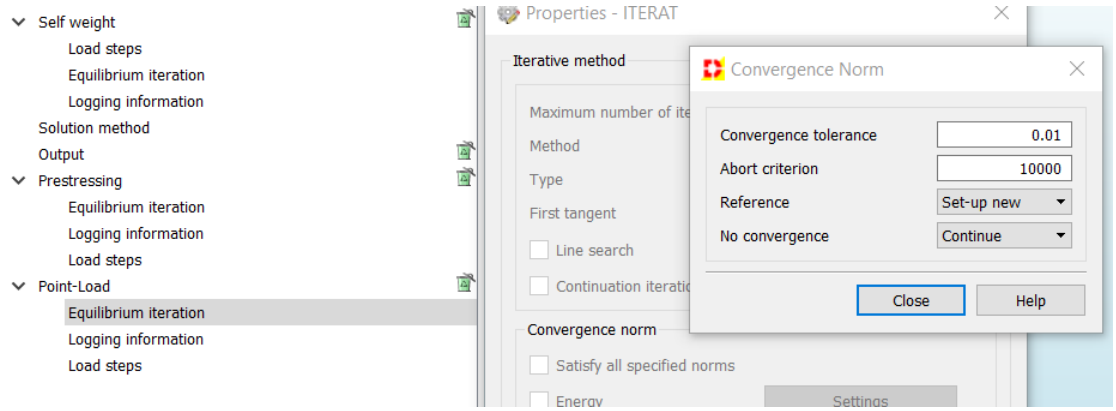

Figure 72: Convergence norm

8.3.9. Selection of the outputs

The last step is choosing the outputs that the user considers necessary to have a clear idea on how the beam behaves in every moment of the process and to be able to draw conclusions about, for example, the displacements, the stresses and the cracks.
Even though there are lots of outputs to be chosen, the following are the ones considered necessary to carry out the analysis and the respective interpretation:

- Displacements total translation global
- Stress total Cauchy global
- Strain crack green
- Strain crkwdt green global
- Status crack

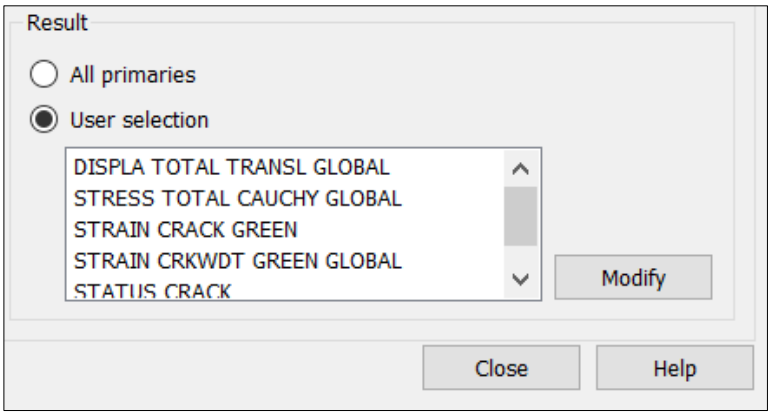

Figure 73: Selected outputs for the analysis

8.4. Working with Python

Modelling a structural element such as this beam can take some time, and especially if you are not familiar with the software. If, for some reason, the study that has to be carried out needs a number of simulations where the measures of the elements or the material properties change, and this would mean having some models that differ from one another, that is when Python comes in handy. It allows changing the inputs easily and quickly and, therefore, making a great number of models in a short time.
Once the base script has been written, changing the inputs is a matter of minutes, whereas modelling another beam from scratch can take way more time. It is strongly recommended to do two different scripts for the same model.

1. Script with all variables and inputs that the user has to define.
2. Script with all the commands that generate the model in DIANA.

It has been commented before that there is no environment that corrects the typos that can be made when writing the DIANA commands, which is why a model should be done with the software's interface and later use the saved script in your PC to make sure that the commands are well written.

8.4.1. How to define classes and variables

It has to be said that there is plenty of information on Internet on how to program with Python, including all the functions that will be used in this project.
Despite this fact, in the following section a little introduction on how to program the basics on Python will be done.

To start with, a class will be declared. As explained in sections *7.1 Definitions script without post-tensioning* and *7.2 Definitions script with post-tensioning*, there will be classes for all the features of the model: beam, steel plate, reinforcement, etc.

```
class BeamClass:
    x0 = 0
    y0 = 0
    z0 = 0
    span = 6
    width = 0.4
    depth = 0.6
    name = "Concrete Beam"
    typeOfConcrete = "C45/55"
    pass
Beam = BeamClass()
```

In this part of the script there is nearly everything that will be needed to do the "variables script". On the first line the class "BeamClass" is declared and inside of it will be the necessary variables to model a beam. In the first place there are the three origin coordinates (x, y, z), in the second place there are the dimensions (span, width, depth), then the name that the beam will receive, and finally the type of concrete that will be used.
And the last line creates a new instance of the class and assigns this object to the local variable Beam, meaning that every time that the variables declared inside have to be used, they will be called in the following way:
Beam.span
If the value of the variable written on the anterior line is printed, the result will be 6, because that is the value that has been introduced before. It can be changed by just giving it another number. For example:
Beam.span = 8; And now the span of the beam will be 8 metres.

The same is done with the covers of concrete, where the class is called CoversClass and the local variable to which this object is assigned is Covers.
The *pass* is used when no more variables are to be declared in that class.

```
class CoversClass:
    vertical = 0.1
    horizontal = 0.1
    pass
Covers = CoversClass()
```

8.4.2. Linking scripts

Having a tidy and organized script is crucial, especially if the structural element that wants to be modelled is geometrically complex. Even though the beam modelled in this project is simple, there are many things to be defined such as the geometry, the materials and their properties, the supports, the loads and whatnot. Therefore, it will strongly help having one script with all the variables and another with the commands, not only for the people who want to read and understand it but also for the person who is writing it.
That is when the command of linking two scripts should be used. This will allow having a script with all the declared variables and these ones can be used in the other document just as if they had been declared in the same.
The command that is used is the following:
exec(open('C:/Users/marti/Desktop/Python_Scripts/Definitions_prestressing.py').read())

It can be observed that the command opens the document that is saved in the direction introduced afterwards and then it reads it. In this case, the document that is being read is the "Definitions_prestressing.py" that is the document where all the classes and variables of the case with post-tensioning are defined. This command is written just in the beginning of the command script and it is a way to import all the information that is contained in the direction script to the one where the command is written.
The same command is used for the case without post-tensioning, but this time the definitions are different and, therefore, so is the script.
exec(open('C:/Users/marti/Desktop/Python_Scripts/Definitions_no_prestressing.py').read())

8.4.3. Creating loops

In programming, when a command has to be done more than once under a condition, the loops are used.
A good example would be when the properties of the steel rebar have to be assigned to all the bars that the beam contains. If the automatically saved script from DIANA when a model is done is observed, it will be seen that it takes a line of code for every rebar that has to have its material properties assigned. Whereas if a loop is introduced to the code, it can take three lines to set the material properties of a hundred rebars.
The loop that will be used in this project is the *for.* That means the process inside the loop will be done as many times as the condition imposed in this structure. To put it in an example:

```
counter = 1
for counter in range (1, 5):
DIANA COMMAND
```

In the first line a variable named *counter* is declared and a number is assigned to it.
In the second line it can be observed that the variable declared before is used and the *loop* will take place as long as the value of the variable is within the range inside the brackets. It has to be born in mind that the limits are exclusive with the last number, which means that number five is not included and the loop will only take place from *counter = 1* to *counter = 4.*
Every time that the loops takes place the variable *counter* is increased by one.

9. Structural damages on post-tensioned systems

The post-tensioning process can be difficult to execute because it is made up of different steps that must be followed carefully and no mistakes can be made if the results are to be effective and long-lasting.
Even though the people who execute this process try to be really careful, sometimes some errors can appear and that will alter the expected behaviour and durability of the structural element.
One of the most common errors is the lack of grouting when the post-tension load has already been applied, making it possible for the air to establish contact with the active reinforcement and, therefore, giving place to early corrosion that will end up damaging the rebar.
Modelling with a lack of grouting can be easy to do and it will be studied in this thesis, but the corrosion will not be taken into account.

9.1. Lack of grouting

There is a way to simulate this kind of errors in DIANA, and that consists in dividing the active reinforcement in different sections and modifying their properties, so that some of them are bonded to the mother element (well-done grouting) and the other are not (poor grouting).
To proceed with this section, and using exactly the same amount of active reinforcement as in section *3.Modelling of a beam with post-tensioning,* each bar will be divided into three different sections of the same length. To do so, three different curves will be created for each bar, meaning that the total number of curves created will be fifteen.
It has been explained before that the active reinforcement has no bonding with the concrete, meaning that it can extend and contract freely, just having some friction with the plastic ducts. During the analysis, this property will be changed and the reinforcement will be bonded to the mother element, meaning that the grout has been poured.
In this specific case, just the sections in the middle of the tendons will be bonded. This means that the grout has not arrived at the sections in both ends because it has stayed in the bottom and, consequently, the extreme sections are not bonded and are exposed to the corrosion.

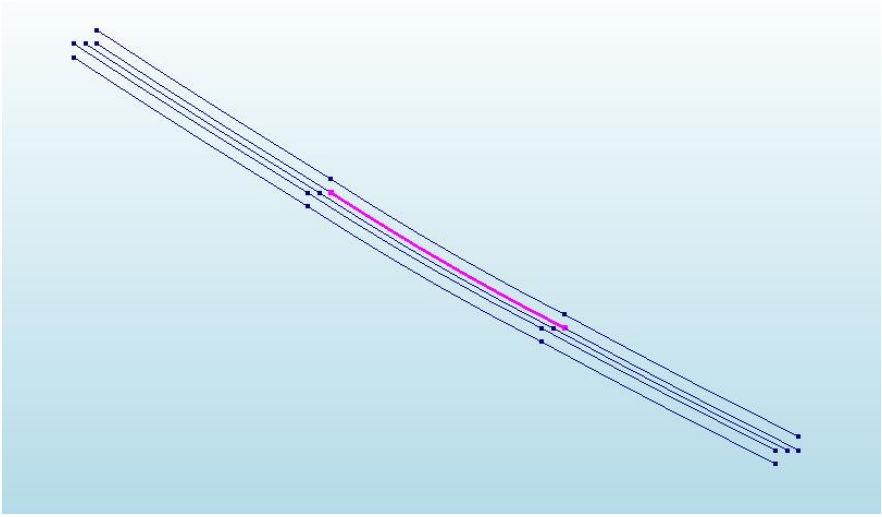

Figure 74: Reinforcement divided into three sections

9.1.1. Change of properties

The process followed to change the properties of the middle sections of the tendons has to be exactly the same as the one in section *8.3.6 Creation of the third "Execute steps". Step to change the reinforcement properties*. Specifically, the procedures reflected in *Figure 67* and *Figure 68*.
This change will mean that the only part where the grouting has settled is on the middle sections of the active reinforcement

9.1.2. DIANA's limitations

During this thesis three DIANA limitations have been found and here they will be explained. That will keep the results from being the hoped ones.

1) There is no continuity when a tendon is split in different sections.

The used software considers every one of the sections as a different bar even though they share a coordinate. This means that the last point of the first section is, obviously, the first of the second section.
In a Finite Element Software this should mean that the information about displacements and stresses that the last coordinate of the first section contains is exactly the same as the first coordinate of the second section, but that does not happen in DIANA when different tendon sections are used. If it worked properly, despite being different sections, they would all be part of the same tendon.
Consequently, the sections will work as separate tendons and each one of them will move on its own accord. This will be a problem because the results that are going to be obtained when the analysis is run are not reliable and stray from reality.

2) It is not possible to apply post-tensioning to a section without anchoring points

Because of the reason explained in section point 1), the middle part of the tendon will not be under the post-tensioning load applied on the extreme parts of the tendon and the post-tensioning load will have to be directly applied to it. Here is where the other problem takes place. It has been tried to apply the post-tensioning load to the middle part of the tendon, but some anchorage needs to take place. The ideal thing would be stressing it with the same load that the other parts have been stressed but not anchoring it anywhere, considering that they are all part of the same tendon.
In conclusion, the obtained results will stray from reality because the middle section will not be receiving any of the post-tensioning force and, as a consequence, the beam will not be experiencing the loads that take place in reality.

3) The properties of the embedded reinforcements are difficult to change

As seen in *8.3.6 Creation of the third "Execute steps". Step to change the reinforcement properties,* the only property of the reinforcement that can be changed is the bonding with the concrete.
It would have been ideal to be able to change the friction coefficients, the post-tensioning load, and other features, but it has not been possible. This has resulted in a lack of reliability on the obtained analysis and proves that some work has to still be done.
The solution would be using the bond-slip modelling, as explained in sections 4 and 5 in (DIANA FEA, 2015)[(9)].

9.1.3. Application of the load

For the reasons explained before, the post-tensioning load will be divided into two different load-cases. They will have the same value but will be applied in different sections of the tendons.

1. It will affect the sections on the left side of the beam. The load will just be applied in one end and this will be the anchorage point (x=0m). Its value will be 80000N.
2. It will affect the sections on the right side of the beam. The load will just be applied in one end and this will be the anchorage point (x=6m). Its value will be 80000N.

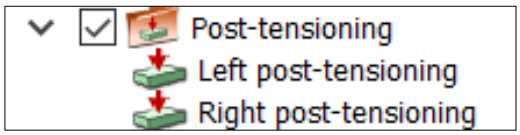

Figure 75: The two load-cases for the post-tensioning

9.1.4. Results

Because of DIANA's limitations and the lack of knowledge on the software, the obtained results in this section are far from precise and have nothing to do with reality, but they should be included so that the people that want to continue working on this topic do not make the same errors.

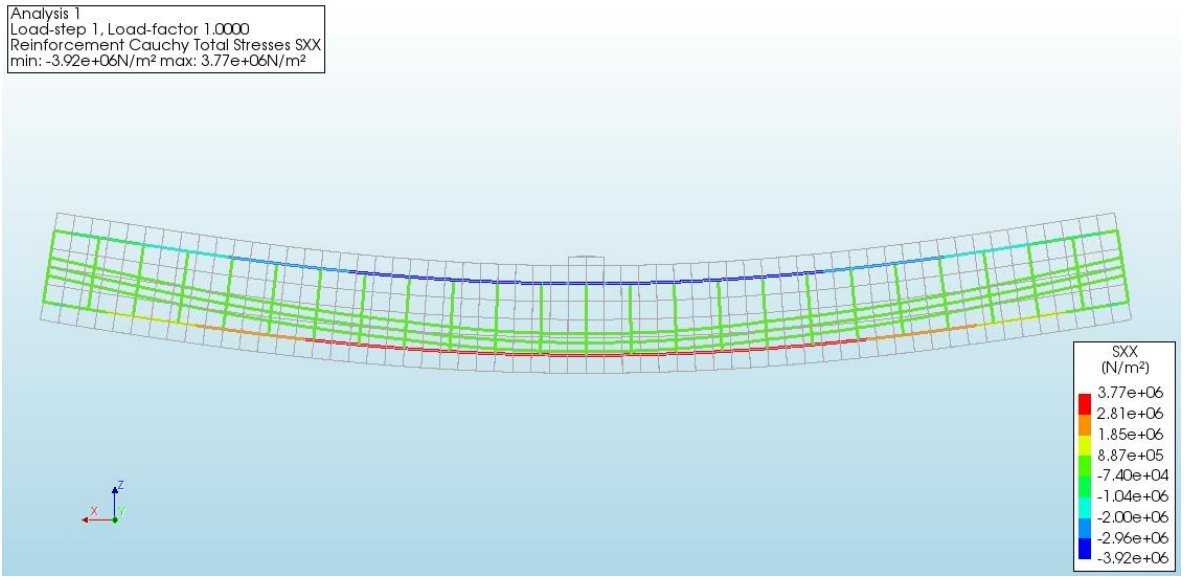

Figure 76: Reinforcement stresses on the 1st load-step with poor grouting

In this figure it can be observed that the active reinforcement has the same colour all along the beam and that is green, meaning that now there is no stress at all. In this first load-step only the self-weight of the beam has been applied.

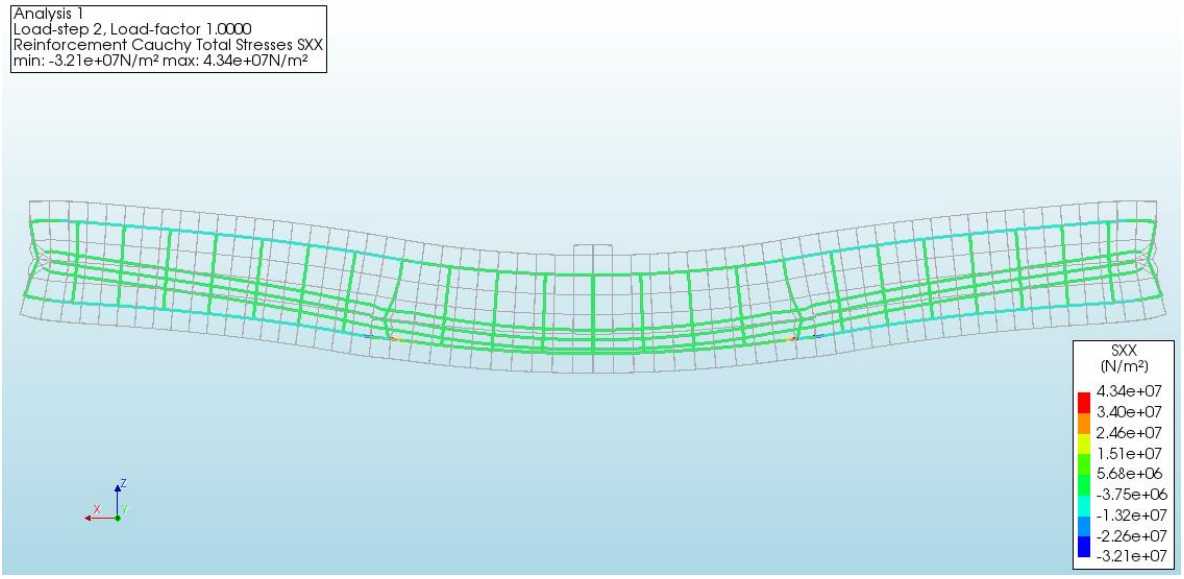

Figure 77: Reinforcement stresses on the 2nd load-step with poor grouting

Now the post-tensioning load has been applied on the sections in the ends of the beam, and it can be noted they are deforming the structural element.

The distribution is also green, but this time it is a colour convention, because if the legend is observed, the stresses are big in all the reinforcement (active and passive).

If the post-tensioning load had been applied, the beam would a shape really similar to the one in section *3.4.1.3 4th load-step.*

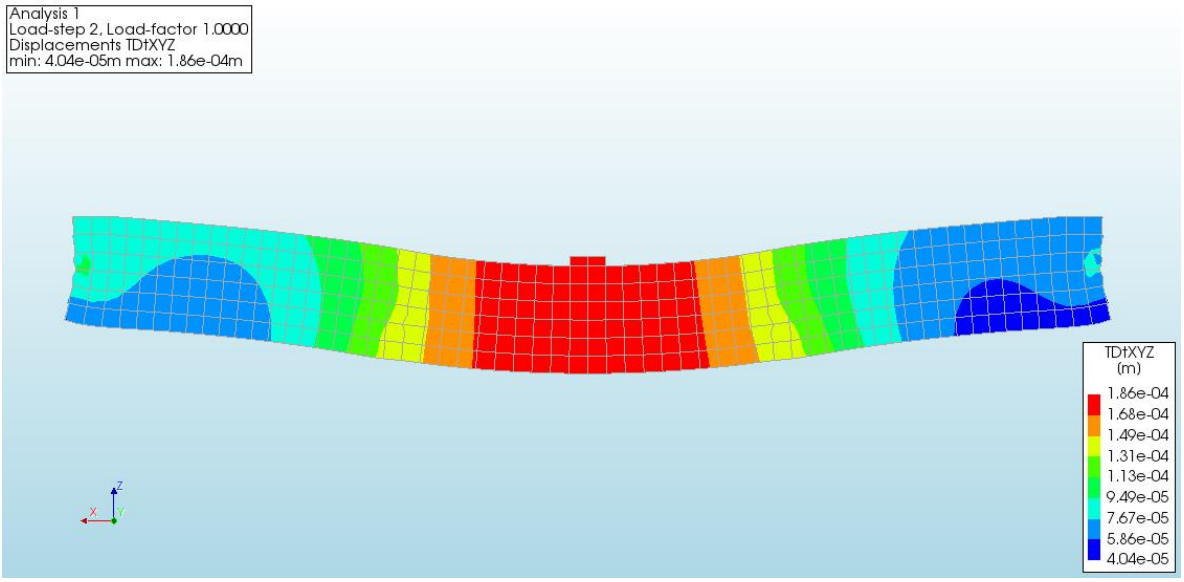

Figure 78: Total displacements on the 2nd load-step with poor grouting

The colour distribution allows observing which are the parts where the displacements are greater and where they are smaller. It is also easier to see the general shape, and *Figure 78* permits noting it is a weird distribution that would not take place in reality.

10. Conclusions

In this project, a 3D model of a reinforced concrete beam with and without post-tensioning has been made using the Finite Element Software DIANA, and later the exact same thing has been done but this time using the feature of the software to program with Python. Afterwards, the obtained results have been studied, commented and compared. Later, some Python introduction has been made to get familiarized with it.
To end with, a case with poor grouting has been studied and commented. Moreover, in order to help the people who want to do a similar model or use Python in DIANA, the most difficult things throughout the process have been thoroughly explained along with some pictures.
The conclusions that can be finally drawn are the following:

- Great importance in using a steel plate if the load is applied in a point. The fact of not using it could cause cracks on the superior part, that is where the load is applied and, therefore, change the beam's behaviour completely.

- It has been seen that the size of the mesh can have a great influence on the obtained results, making them less precise if the mesh is bigger. Meshing the structural element into very small parts can make the analysis process way longer and can also require a more powerful processing unit, but it will make the results extremely accurate and reliable.

- In reality, there are always temperature changes that could modify the dimensions of the structural element and, consequently, cause some cracks, even though they are not being considered in this project. To avoid that from happening, some translations are not restricted and that will give the element freedom to expand or contract in that direction. For instance, one end of the beam in the X direction has to be bond-free and that will allow size change if there are some.
  Considering that in this project there are no thermal effects, the only translation restriction will be on the Z-axis.

- After some simulations and studied cases, it has been observed that the linear behaviour of the reinforced beam lasts as long as there are no cracks on the inferior part. It is the first place they start appearing unless the post-tensioning load is really big and causes cracks, but these cracks do not stop the linear behaviour.

- When failure is reached, the load applied keeps increasing and makes the displacements huge. That is the reason why in the load-displacement charts there is not a section where the load starts decreasing while the displacements keep getting bigger. What happens instead is that the line stops abruptly when failure has been reached.

- One of the most important things of the post-tensioning process is making sure that the active reinforcement adopts a parabolic shape so that the bending moment created by the post-tensioning load due to the eccentricity of the tendons counteracts the one created by the loads applied.

This eccentricity has to be zero on the ends so that tension does not appear on the superior part of these zones because there the bending moment created by the external loads is also non-existent and, as a result, there is nothing to counteract.

- The most difficult part when modelling the beam is the active reinforcement. It has already been explained that it should adopt a parabolic shape and that means defining lots of coordinate points so that the lines take this shape. The job becomes even more tedious if the beam is bigger and especially when it has to be programmed with Python, since it is a difficult process to automate and consequently takes a lot of lines on the code to do it.

- Even though it is pretty obvious what to do in reality, DIANA offers the option, as seen in section *5.1 Bonding to the mother element*, to choose whether to bond the active reinforcement to the beam before the application of the post-tensioning load or after. In the same section, it has been observed that it makes no sense to bond it before applying the load because then the bending moment that should counteract the self-weight and point-load does not take place.
  Also, it is known that the grouting is added when the load is already acting on the structural element.

- Despite not making a difference in DIANA software, it has been read and observed in different documents mentioned in the *Bibliography,* that applying the post-tensioning load in both ends of the tendons can have a big impact on the load loss due to friction. As explained in section *5.2 Post-tensioning load applied in one or both ends,* reducing the curvature can also prevent the system from having big load losses but this has already been taken care of.

- Even though it has been proved countless times and post-tensioning is a very common process nowadays, in section *6. Comparison between non post-tensioned and post-tensioned beams,* there is a comparison table that allows seeing the effect that this process truly has, and how much the load that the structural element is able to withstand without post-tensioning increases.

- Python has been proved to be incredibly useful when a lot of simulations have to be done with slight changes between models. Having to model the beam every time that a simulation wants to be done would take from ten to fifteen minutes, while with the Python script it is completely instantaneous.

- The classes in Python have resulted to be the perfect way to organize all the variables and to remember what every one of them does, since they are all linked to the name of the element that they represent. As explained in sections *7.1 Definitions script without post-tensioning* and *7.2 Definitions script with post-tensioning* there is a class for every element of the model and this makes the user know and remember all the variables even if they have not touched the script for days.

- DIANA may seem a difficult software to use, but once the user has gotten familiar with it, it results to be very user-friendly, and an extremely powerful and precise tool. During this project there has also been a downside to it because there are some limitations that have stopped the model with lack of grouting from being realistic and reliable.

- Even though the lack of grouting scenario has not worked as expected, the progress on learning how to use DIANA and complementing it with Python has been more than satisfactory.

- For a future project where the goal is studying the structural effects of damaged post-tensioned systems, the document *Pre- and post-tensioning of a concrete beam* from DIANA can be extremely helpful, especially sections 4 and 5.

Bibliography

(1) Palmer, B. (2010). Post-Tensioning- Methods for Reinforcing Concrete - The Concrete Network. Retrieved May 11, 2020, from ConcreteNetwork.com website: https://www.concretenetwork.com/post-tension/

(2) Freeman. (2013). Eurocode 2, *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699. https://doi.org/10.1017/CBO9781107415324.004

(3) DIANA FEA. (2010). *Creep response of a prestressed concrete beam under sustained load*.

(4) COMSOL. (2016). Finite Element Mesh Refinement Definition and Techniques. Retrieved March 7, 2020, from https://www.comsol.com/multiphysics/mesh-refinement

(5) Younis, W. (2009). An Overview of Part and Assembly Stress Analysis. In *Up and Running with Autodesk Inventor Simulation 2010*. https://doi.org/10.1016/b978-1-85617-694-1.00009-3

(6) Linear and nonlinear structural analysis. (2017). Retrieved April 25, 2020, from https://www.femto.eu/stories/linear-non-linear-analysis-explained/

(7) Sørensen, S. I. (2013). *Betong-Konstruksjoner; Beregning og dimensjonering etter Eurocode 2*.

(8) 9. Classes — Python 3.8.2 documentation. (n.d.). Retrieved April 19, 2020, from https://docs.python.org/3/tutorial/classes.html

(9) DIANA FEA. (2015). *Pre- and post-tensioning of a concrete beam*.

Attachments

1. Definitions: without post-tensioning (*Python script*)

```
from math import * #math import that allows us using some mathematical functions
class BeamClass:
    x0 = 0 #origin x coordinate
    y0 = 0 #origin y coordinate
    z0 = 0 #origin z coordinate
    span = 6 #beam length
    width = 0.4 #beam width
    depth = 0.6 #beam depth
    name = "Concrete Beam" #name that we want to give to the block created in DIANA
    typeOfConcrete = "C45/55" #class of concrete that we want to use for the simulation
    pass #where the declaration of variables ends before giving name to the class
Beam = BeamClass() #we establish the name to call all the variables once they've been defined

class SteelPlateClass:
    y0 = Beam.y0 #origin y coordinate
    z0 = Beam.depth #origin z coordinate
    span = 0.2 #span of the plate
    width = Beam.width #width of the plate
    depth = 0.05 #depth of the plate
    name = "Steel Plate" #name that we want to give to the second block created
    typeOfSteel = "S275" #kind of steel that we want to use for the simulation
    pass #where the declaration of variables ends before giving name to the class
SteelPlate = SteelPlateClass() #we establish the name to call all the variables once they've been
defined
SteelPlate.x0 = Beam.span / 2 - SteelPlate.span / 2 #origin x coordinate. We declare it after the
class has been created because some variables created inside are needed.

class PointClass:
        x = Beam.span / 2 #x coordinate
        y = Beam.width / 2 #y coordinate
        z = Beam.depth + SteelPlate.depth #z coordinate
        name = "Point" #name that we want to give to the point created
        pass #where the declaration of variables ends before giving name to the class
Point = PointClass #we establish the name to call all the variables once they've been defined

class CoversClass:
    vertical = 0.1 #vertical cover
    horizontal = 0.1 #horizontal cover
    pass #where the declaration of variables ends before giving name to the class
Covers = CoversClass() #we establish the name to call all the variables once they've been defined

class RebarClass: #we declare the class REBAR.
    distance_between = 0.1 #distance between the horizontal bars.
    x = Beam.x0 #x coordinate where the rebar has to start (it is the same as where the beam starts)
    y = Covers.horizontal  # initial position in Y axis where the first rebar will be
    z_inferior = Covers.vertical  # position in Z axis where the inferior rebar will be
    z_superior = Beam.depth - Covers.vertical  # position in Z axis where the superior rebar will be
    name = "Rebar" #name that we want to give to all the bars created
    young_modulus = 2e+11 #value of the young's modulus of the steel
    yield_stress = 5e+8 #value of the yield stress of the steel
    diametre = 0.012 #value of the diametre of the bars
    name_diametre = "Diametre Rebar" #name that we want to give to the diametre defined before
    pass #where the declaration of variables ends before giving name to the class
Rebar = RebarClass() #we establish the name to call all the variables once they've been defined
Rebar.number_inferior = (ceil(Beam.width / Rebar.distance_between)) - 1  # number of bars in the
inferior part
Rebar.number_total = 2 * Rebar.number_inferior  # total number of bars (inferior+superior)
#they're both declared after the class has been named because some variables included inside were
needed.

class StirrupsClass:
    distance_between = 0.25 #distance between stirrups
    name = "Stirrup"
    #we define the positions of the points of the polyline
    x = Beam.x0  # this is where the first stirrup will be in the X axis
    y1 = Covers.horizontal #coordinate y of the first and fourth points
```

```
    y2 = Beam.width - Covers.horizontal #coordinate y of the second and third points
    z1 = Covers.vertical #coordinate z of the first and second points
    z2 = Beam.depth - Covers.vertical #coordinate z of the third and fourth points
    diametre = 0.008 #we declare the diametre of the stirrups
    name_diametre = "Diametre Stirrups" #we declare the name we want to give to all the created bars
    pass #where the declaration of variables ends before giving name to the class
Stirrups = StirrupsClass() #we establish the name to call all the variables once they've been
defined
Stirrups.number = (int(Beam.span / Stirrups.distance_between)) + 1 #we count all the stirrups needed
for the beam

class LoadsClass:
    pointload = 1000  #force in Newtons
    pointload_direction = 3 #1 stands for x, 2 for y and 3 for z
    pointload_name = "Point Load" #name of the point load applied
    selfweight_name = "Self weight"#name of the selfweight applied
    pass #where the declaration of variables ends before giving name to the class
Loads = LoadsClass() #we establish the name to call all the variables once they've been defined

class CategoriesClass:
    beams = "Beams" #we give name to the general category that includes the beam
    plates = "Plates" #we give name to the general category that includes the steel plate
    rebar = "Rebar" #we give name to the general category that includes all the rebar
    stirrups = "Stirrups" #we give name to the general category that includes all the stirrups
    pass #where the declaration of variables ends before giving name to the class
Categories = CategoriesClass() #we establish the name to call all the variables once they've been
defined

class MaterialsClass:
    concrete = "Concrete" #name of the material 1
    steel = "Steel" #name of the material 2
    reinfo_steel = "Reinforcement steel" #name of the material 3
    pass #where the declaration of variables ends before giving name to the class
Materials = MaterialsClass() #we establish the name to call all the variables once they've been
defined

class SupportsClass:
    name = "Supports" #name of the general category
    x_translation = 0 #a 0 will mean the translation can take place. A 1 will mean that it is
impeded
    y_translation = 0
    z_translation = 1
    x_rotation = 0
    y_rotation = 0
    z_rotation = 0
    pass #where the declaration of variables ends before giving name to the class
Supports = SupportsClass() #we eestablish the name to call all the variables once they've been
defined

class PrincipalAxisClass: #numbers that correspond to the principal axis. Cannot be changed
    x = 1 #we want the user to know what number corresponds to each axis
    y = 2
    z = 3
    pass #where the declaration of variables ends before giving name to the class
PrincipalAxis = PrincipalAxisClass() #we establish the name to call all the variables once they've
been defined

class FactorCombinationClass:
    c1 = 1 #just point-load applied
    c2 = 2 #just self-weight applied
    c3 = 3 #point-load+self-weight applied
    pass #where the declaration of variables ends before giving name to the class
FactorCombination = FactorCombinationClass() #we establish the name to call all the variables once
they've been defined

class MeshPropertiesClass:
    size = 0.1 #we define the size of cubes of the mesh
    pass #where the declaration of variables ends before giving name to the class
MeshProperties = MeshPropertiesClass() #we establish the name to call all the variables once they've
been defined

class AnalysisClass:
    name = "Analysis 1" #name of the analysis
    type = "Structural nonlinear" #type of analysis that we will run
```

```
    factor_and_numberIterations = "4(60)" #the first number is the factor and the one in the
parenthesis is the nº of iterations
    number_iterations_calculation = 10 #number of iterations DIANA will do for every load-step
    pass #where the declaration of variables ends before giving name to the class
Analysis = AnalysisClass() #we establish the name to call all the variables once they've been
defined
```

2. Definitions: with post-tensioning (*Python script*)

```
from math import *
#all the explanation of the classes and variables but the active reinforcement and the post-
tensioning load is done in Definitions_no_prestressing
class BeamClass:
    x0 = 0
    y0 = 0
    z0 = 0
    span = 6
    width = 0.4
    depth = 0.6
    name = "Concrete Beam"
    typeOfConcrete = "C45/55"
    pass
Beam = BeamClass()

class SteelPlateClass:
    y0 = Beam.y0
    z0 = Beam.depth
    span = 0.2
    width = Beam.width
    depth = 0.05
    name = "Steel Plate"
    typeOfSteel = "S275"
    pass
SteelPlate = SteelPlateClass()
SteelPlate.x0 = Beam.span / 2 - SteelPlate.span / 2

class PointClass:
        x = Beam.span / 2
        y = Beam.width / 2
        z = Beam.depth + SteelPlate.depth
        name = "Point"
        pass
Point = PointClass

class CoversClass:
    vertical = 0.1
    horizontal = 0.1
    pass
Covers = CoversClass()

class RebarClass:
    distance_between = 0.1
    x = Beam.x0
    y = Covers.horizontal  # initial position in Y axis where the first rebar will be
    z_inferior = Covers.vertical  # position in Z axis where the inferior rebar will be
    z_superior = Beam.depth - Covers.vertical  # position in Z axis where the superior rebar will be
    name = "Rebar"
    young_modulus = 2e+11
    yield_stress = 5e+8
    diametre = 0.012
    name_diametre = "Diametre Rebar"
    pass
Rebar = RebarClass()
Rebar.number_inferior = (ceil(Beam.width / Rebar.distance_between)) - 1  # number of bars in the
inferior part
Rebar.number_total = 2 * Rebar.number_inferior  # total number of bars (inferior+superior)

class StirrupsClass:
    distance_between = 0.25
    name = "Stirrup"
    # we define the positions of the points of the polylines
    x = Beam.x0  # this is where the first stirrup will be in the X axis
    y1 = Covers.horizontal
    y2 = Beam.width - Covers.horizontal
    z1 = Covers.vertical
    z2 = Beam.depth - Covers.vertical
    diametre = 0.008
    name_diametre = "Diametre Stirrups"
    pass
```

```
Stirrups = StirrupsClass()
Stirrups.number = (int(Beam.span / Stirrups.distance_between)) + 1

class ActiveReinfoClass:
    # we define the cover and the positions of the active rebar
    vertical_cover = 0.25 #we declare the vertical cover so that the vertical rebar doesn't get
corroded
    horizontal_cover = 0.1 #we declare the horizontal cover so that the vertical rebar doesn't get
corroded
    name = "Active rebar" #we declare the name that all the bars will receive
    young_modulus = 1.9e+11 #value of the young's modulus of the active rebar
    diametre = 0.18 #diametre of the active rebar
    name_diametre = "Diametre Active Reinforcement" #name that the diametre of the active rebar will
receive
    pass
ActiveReinfo = ActiveReinfoClass()
#we define all these variables after the class has been declared because some variables included
inside are nedeed
ActiveReinfo.y1 = ActiveReinfo.horizontal_cover
ActiveReinfo.y2 = Beam.width / 2
ActiveReinfo.y3 = Beam.width - ActiveReinfo.horizontal_cover
ActiveReinfo.z1_1 = ActiveReinfo.vertical_cover #first layer first point
ActiveReinfo.z1_2 = ActiveReinfo.vertical_cover - 0.07 #first layer second point
ActiveReinfo.z1_3 = ActiveReinfo.vertical_cover - 0.1 #first layer third
ActiveReinfo.z2_1 = Beam.depth / 2 #second layer first point
ActiveReinfo.z2_2 = (Beam.depth / 2) - 0.07 #second layer second point
ActiveReinfo.z2_3 = (Beam.depth / 2) - 0.1 #second layer third point
ActiveReinfo.z3_1 = Beam.depth - ActiveReinfo.vertical_cover #third layer first point
ActiveReinfo.z3_2 = (Beam.depth - ActiveReinfo.vertical_cover) - 0.07 #third layer second point
ActiveReinfo.z3_3 = (Beam.depth - ActiveReinfo.vertical_cover) - 0.1 #third layer third point
ActiveReinfo.x_positions = [0, 1, 2, 3, 4,5]  # we define an array with random numbers(except the
first one that has to be a 0)
ActiveReinfo.distance_points = Beam.span / 5  # we divide the lenght of the beam into 5 exact pieces
counter = 1  # we define counter again
for counter in range(1, 6):
    ActiveReinfo.x_positions[counter] = ActiveReinfo.x_positions[counter - 1] +
ActiveReinfo.distance_points  # we fill the array declared before with the distances calculated
before

class LoadsClass:
    pointload = 1000  #force in Newtons
    pointload_direction = 3 #1 stands for x, 2 for y and 3 for z
    pointload_name = "Point Load" #name of the point load applied
    selfweight_name = "Self weight"#name of the selfweight applied
    tendon_force = 80000 # we define the load that we want to apply to the post-tension
    posttension_name = "Post-tensioning"
    friction_coefficient = 0.1
    pass
Loads = LoadsClass()

class CategoriesClass:
    beams = "Beams"
    plates = "Plates"
    rebar = "Rebar"
    stirrups = "Stirrups"
    prestressing = "Active reinforcement"
    pass
Categories = CategoriesClass()

class MaterialsClass:
    concrete = "Concrete"
    steel = "Steel"
    reinfo_steel = "Reinforcement steel"
    prestress_steel = "Prestressing steel"
    pass
Materials = MaterialsClass()

class SupportsClass:
    name = "Supports"
    x_translation = 0
    y_translation = 0
    z_translation = 1
    x_rotation = 0
    y_rotation = 0
```

```
    z_rotation = 0
    pass
Supports = SupportsClass()

class PrincipalAxisClass: #numbers that correspond to the principal axis. Cannot be changed
    x = 1
    y = 2
    z = 3
    pass
PrincipalAxis = PrincipalAxisClass()

class FactorCombinationClass:
    c1 = 1
    c2 = 2
    c3 = 3
#no other combination is needed because the combination of the loads takes place in the Analysis
    pass
FactorCombination = FactorCombinationClass()

class MeshPropertiesClass:
    size = 0.1
    pass
MeshProperties = MeshPropertiesClass()

class AnalysisClass:
    name = "Analysis 1"
    type = "Structural nonlinear"
    factor_and_numberIterations = "4(60)" #the first number is the factor and the one in the
parenthesis is the nº of iterations
    number_iterations_calculation = 10
    pass
Analysis = AnalysisClass()
```

3. Commands: without post-tensioning (Python script)

```
exec(open('C:/Users/marti/Desktop/Python_Scripts/Definitions_no_prestressing.py').read()) #we link
this document to the one where all the classes and variables are declared

#WE START THE PROGRAM

createBlock(Beam.name, [Beam.x0, Beam.y0, Beam.z0], [Beam.span, Beam.width, Beam.depth]) #we create
the beam
createBlock( SteelPlate.name, [ SteelPlate.x0, SteelPlate.y0, SteelPlate.z0 ], [ SteelPlate.span,
SteelPlate.width, SteelPlate.depth ] ) #we create the steel plate
createPointBody( Point.name, [ Point.x, Point.y, Point.z ] ) #we create the point in which the load
will be applied

projection( "SHAPEFACE", SteelPlate.name, [[ Point.x, Point.y, Point.z ]], [ Point.name ], [ 0, 0, -
1 ], True ) #we imprint the point on the steel plate
removeShape( [ Point.name ] ) #we remove the point created before. now it is part of the steel plate

#we use a F0R loop in order to create the inferior passive reinforcement
counter = 1
for counter in range(1, Rebar.number_inferior+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_inferior ], [Beam.span,
Rebar.y, Rebar.z_inferior])
    Rebar.y = Rebar.y + Rebar.distance_between

#we use a FOR loop in order to create the superior passive reinforcement
Rebar.y = Covers.horizontal #we define again the distance of the rebar from the edge
for counter in range(Rebar.number_inferior+1, Rebar.number_total+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_superior ], [Beam.span,
Rebar.y, Rebar.z_superior])
    Rebar.y = Rebar.y + Rebar.distance_between

#we define the COUNTER again and create a loop to create the polyline
counter = 1
for a in range(1, Stirrups.number+1):
    createPolyline(Stirrups.name + str(a), [[Stirrups.x, Stirrups.y1, Stirrups.z1], [Stirrups.x,
Stirrups.y2, Stirrups.z1], [Stirrups.x, Stirrups.y2, Stirrups.z2], [Stirrups.x, Stirrups.y1,
Stirrups.z2]], True )
    Stirrups.x = Stirrups.x + Stirrups.distance_between

#we rename and create the needed categories
rename( "SHAPESET", "Shapes", Categories.beams ) #we change the name of the folder. from shapes to
beam
addSet( "SHAPESET", Categories.plates ) #we add the category for plates
addSet( "SHAPESET", Categories.rebar ) #we add the category for rebar
addSet( "SHAPESET", Categories.stirrups ) #we add the category for stirrups

#we move all the elements to their respectives categories
moveToShapeSet( [ SteelPlate.name ], Categories.plates ) #we move the plate from the category "beam"
to the category "plates"
#we define the COUNTER again and use a FOR loop to move all the lines from the category "beam" to
the category "rebar"
counter = 1
for counter in range(1, Rebar.number_total + 1):
    moveToShapeSet( [ Rebar.name + str(counter) ], Categories.rebar )

#we define the COUNTER again and create a FOR loop to move all the lines from the category "beam" to
the category "stirrups"
counter = 1
for counter in range(1, Stirrups.number + 1):
    moveToShapeSet( [ Stirrups.name + str(counter) ], Categories.stirrups )

#we add the materials that we will deal with
#we add the concrete
addMaterial( Materials.concrete, "CONCDC", "EN1992", [ "TOTCRK" ] ) #we add the material
setParameter( "MATERIAL", Materials.concrete, "EC2CON/NORMAL/CLASS", Beam.typeOfConcrete) #we set
the properties of the material
#we add the steel used in the plate
addMaterial( Materials.steel, "STEEDC", "EN1993", [] ) #we add the material
setParameter( "MATERIAL", Materials.steel, "EURO93/EN931/GRADE", SteelPlate.typeOfSteel ) #we set
the properties of the material
#we add the steel used in the reinforcement
addMaterial( Materials.reinfo_steel, "REINFO", "LINEAR", [ "DESIGN" ] ) #we add the material
setParameter( "MATERIAL", Materials.reinfo_steel, "LINEAR/ELASTI/YOUNG", Rebar.young_modulus ) #we
```

```
set the properties of the material
setParameter( "MATERIAL", Materials.reinfo_steel, "REDESI/YLDSTR", Rebar.yield_stress ) #we set the
properties of the material

#we assign the materials to the geometries created before
#we assign the concrete to the beam
setCurrentShapeSet( Categories.beams )
setElementClassType( "SHAPE", [Beam.name], "STRSOL")
assignMaterial(Materials.concrete, "SHAPE", [Beam.name]) #we assign concrete to block1

#we assign the steel to the plate
setCurrentShapeSet( Categories.plates )
setElementClassType( "SHAPE", [ SteelPlate.name ], "STRSOL" )
assignMaterial( Materials.steel, "SHAPE", [ SteelPlate.name ] ) #we assign steel to block 2

#we define a diametre for the rebar
setCurrentShapeSet( Categories.rebar )
addGeometry( Rebar.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/DIAMET", Rebar.diametre )

#we define a diametre for the stirrups
setCurrentShapeSet( Categories.stirrups )
addGeometry( Stirrups.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/DIAMET", Stirrups.diametre )

#we define the COUNTER again and do a FOR loop to assign the geometry and properties to all the
passive horizontal rebar
counter = 1
for counter in range(1, Rebar.number_total+1):
    setReinforcementAspects( [ Rebar.name + str(counter) ] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Rebar.name + str(counter) ] )
    assignGeometry( Rebar.name_diametre, "SHAPE", [ Rebar.name + str(counter) ] )
    resetElementData( "SHAPE", [ Rebar.name + str(counter) ] )
    setReinforcementDiscretization( [ Rebar.name + str(counter) ], "ELEMENT" )

#we define the COUNTER again and do a FOR loop to assign the geometry and properties to all the
stirrups
counter = 1
for counter in range(counter, Stirrups.number+1):
    setReinforcementAspects( [ Stirrups.name + str(counter)] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Stirrups.name + str(counter) ] )
    assignGeometry( Stirrups.name_diametre, "SHAPE", [ Stirrups.name + str(counter) ] )
    resetElementData( "SHAPE", [ Stirrups.name + str(counter) ] )
    setReinforcementDiscretization( [ Stirrups.name + str(counter) ], "ELEMENT" )

#we define the supports
addSet( "GEOMETRYSUPPORTSET", Supports.name )
createLineSupport( Supports.name, Supports.name )
setParameter( "GEOMETRYSUPPORT", Supports.name, "AXES", [ PrincipalAxis.x, PrincipalAxis.y ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "TRANSL", [ Supports.x_translation,
Supports.y_translation, Supports.z_translation ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "ROTATI", [ Supports.x_rotation,
Supports.y_rotation, Supports.z_rotation ] )
attach( "GEOMETRYSUPPORT", Supports.name, Beam.name, [[Beam.x0, Beam.width / 2, Beam.z0],
[Beam.span, Beam.width / 2, Beam.z0]]) #coordinates of the lines in both ends of the beam

#we apply the self-weight
addSet( "GEOMETRYLOADSET", Loads.selfweight_name )
createModelLoad( Loads.selfweight_name, Loads.selfweight_name ) #the first name  is the name of the
force and the second the category

#we apply the load in the coordinates of the point created before
addSet( "GEOMETRYLOADSET", Loads.pointload_name )
createPointLoad( Loads.pointload_name, Loads.pointload_name )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/VALUE", -Loads.pointload )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/DIRECT", Loads.pointload_direction )
attach( "GEOMETRYLOAD", Loads.pointload_name, SteelPlate.name, [[ Point.x, Point.y, Point.z ]] )

#we define the load combinations which will be the default ones
setDefaultGeometryLoadCombinations(  )
addGeometryLoadCombination( "" )
```

```
#we mesh the beam and the plate and generate the mesh
setElementSize([Beam.name, SteelPlate.name], MeshProperties.size, -1, True)
setMesherType([Beam.name, SteelPlate.name], "HEXQUAD")
setMidSideNodeLocation([Beam.name, SteelPlate.name], "LINEAR")
generateMesh( [] )

#we set the properties of the analysis and execute it
addAnalysis( Analysis.name ) #we create the analysis
addAnalysisCommand( Analysis.name, "NONLIN", Analysis.type ) #we define that the analysis will be
nonlinear
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)", Loads.selfweight_name ) #we
define the name of the first load phase (self-weight)
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR",
FactorCombination.c1 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will add the point-load
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)", Loads.pointload_name ) #we
define the name of the third load phase (point-load)
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/ITERAT/MAXITE",
Analysis.number_iterations_calculation ) #we define the number of iterations in the calculation
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/ITERAT/CONVER/DISPLA/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/ITERAT/CONVER/FORCE/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/STEPS/EXPLIC/SIZES",
Analysis.factor_and_numberIterations )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR",
FactorCombination.c2 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/SELTYP", "USER" ) #we define all
the outputs that we want to receive when the analysis is done
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/DISPLA" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRESS" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(1)/CRACK" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(2)/CRKWDT" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STATUS(1)/CRACK" )

runSolver(Analysis.name) #we run the analysis
```

4. Commands: with post-tensioning (Python script)

```
exec(open('C:/Users/marti/Desktop/Python_Scripts/Definitions_prestressing.py').read()) #we link this
document to the one where all the classes and variables are declared

#WE START THE PROGRAM

createBlock(Beam.name, [Beam.x0, Beam.y0, Beam.z0], [Beam.span, Beam.width, Beam.depth]) #we create
the beam
createBlock( SteelPlate.name, [ SteelPlate.x0, SteelPlate.y0, SteelPlate.z0 ], [ SteelPlate.span,
SteelPlate.width, SteelPlate.depth ] ) #we create the steel plate
createPointBody( Point.name, [ Point.x, Point.y, Point.z ] ) #we create the point in which the load
will be applied

projection( "SHAPEFACE", SteelPlate.name, [[ Point.x, Point.y, Point.z ]], [ Point.name ], [ 0, 0, -
1 ], True ) #we imprint the point on the steel plate
removeShape( [ Point.name ] ) #we remove the point created before. now it is part of the steel plate

#we create a for loop in order to create the inferior passive reinforcement
counter = 1
for counter in range(1, Rebar.number_inferior+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_inferior ], [Beam.span,
Rebar.y, Rebar.z_inferior])
    Rebar.y = Rebar.y + Rebar.distance_between

#we create a for loop in order to create the superior passive reinforcement
Rebar.y = Covers.horizontal #we define again the distance of the rebar from the edge to create the
rebar in the superior part
for counter in range(Rebar.number_inferior+1, Rebar.number_total+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_superior ], [Beam.span,
Rebar.y, Rebar.z_superior])
    Rebar.y = Rebar.y + Rebar.distance_between #we separate the bars from one another with the
distance settled before by changig the Y position

#we define counter again and create a loop to create the polyline
counter = 1
for a in range(1, Stirrups.number+1):
    createPolyline(Stirrups.name + str(a), [[Stirrups.x, Stirrups.y1, Stirrups.z1], [Stirrups.x,
Stirrups.y2, Stirrups.z1], [Stirrups.x, Stirrups.y2, Stirrups.z2], [Stirrups.x, Stirrups.y1,
Stirrups.z2]], True )
    Stirrups.x = Stirrups.x + Stirrups.distance_between

#we create the active reinforcement
createCurve(ActiveReinfo.name + str(1), [[ActiveReinfo.x_positions[0], ActiveReinfo.y1,
ActiveReinfo.z1_1],[ActiveReinfo.x_positions[1], ActiveReinfo.y1,
ActiveReinfo.z1_2],[ActiveReinfo.x_positions[2], ActiveReinfo.y1, ActiveReinfo.z1_3],
[ActiveReinfo.x_positions[4], ActiveReinfo.y1, ActiveReinfo.z1_2], [ActiveReinfo.x_positions[5],
ActiveReinfo.y1, ActiveReinfo.z1_1]])
createCurve(ActiveReinfo.name + str(2), [[ActiveReinfo.x_positions[0], ActiveReinfo.y3,
ActiveReinfo.z1_1],[ActiveReinfo.x_positions[1], ActiveReinfo.y3,
ActiveReinfo.z1_2],[ActiveReinfo.x_positions[2], ActiveReinfo.y3, ActiveReinfo.z1_3],
[ActiveReinfo.x_positions[4], ActiveReinfo.y3, ActiveReinfo.z1_2], [ActiveReinfo.x_positions[5],
ActiveReinfo.y3, ActiveReinfo.z1_1]])
createCurve(ActiveReinfo.name + str(3), [[ActiveReinfo.x_positions[0], ActiveReinfo.y2,
ActiveReinfo.z2_1],[ActiveReinfo.x_positions[1], ActiveReinfo.y2,
ActiveReinfo.z2_2],[ActiveReinfo.x_positions[2], ActiveReinfo.y2, ActiveReinfo.z2_3],
[ActiveReinfo.x_positions[4], ActiveReinfo.y2, ActiveReinfo.z2_2], [ActiveReinfo.x_positions[5],
ActiveReinfo.y2, ActiveReinfo.z2_1]])
createCurve(ActiveReinfo.name + str(4), [[ActiveReinfo.x_positions[0], ActiveReinfo.y1,
ActiveReinfo.z3_1],[ActiveReinfo.x_positions[1], ActiveReinfo.y1,
ActiveReinfo.z3_2],[ActiveReinfo.x_positions[2], ActiveReinfo.y1, ActiveReinfo.z3_3],
[ActiveReinfo.x_positions[4], ActiveReinfo.y1, ActiveReinfo.z3_2], [ActiveReinfo.x_positions[5],
ActiveReinfo.y1, ActiveReinfo.z3_1]])
createCurve(ActiveReinfo.name + str(5), [[ActiveReinfo.x_positions[0], ActiveReinfo.y3,
ActiveReinfo.z3_1],[ActiveReinfo.x_positions[1], ActiveReinfo.y3,
ActiveReinfo.z3_2],[ActiveReinfo.x_positions[2], ActiveReinfo.y3, ActiveReinfo.z3_3],
[ActiveReinfo.x_positions[4], ActiveReinfo.y3, ActiveReinfo.z3_2], [ActiveReinfo.x_positions[5],
ActiveReinfo.y3, ActiveReinfo.z3_1]])


#we rename and create the needed categories
rename( "SHAPESET", "Shapes", Categories.beams ) #we change the name of the folder. from shapes to
beam
addSet( "SHAPESET", Categories.plates ) #we add the category for plates
```

```
addSet( "SHAPESET", Categories.rebar ) #we add the category for rebar
addSet( "SHAPESET", Categories.stirrups ) #we add the category for stirrups
addSet( "SHAPESET", Categories.prestressing ) #we add the category for active reinforcement

#we move all the elements to their respectives categories
moveToShapeSet( [ SteelPlate.name ], Categories.plates ) #we move the plate from the category "beam"
to the category "plates"
#we define counter again and create a loop to move all the rebar to their category
counter = 1
for counter in range(1, Rebar.number_total + 1):
    moveToShapeSet( [ Rebar.name + str(counter) ], Categories.rebar )

#we define counter again and create a loop to move all the stirrups to their category
counter = 1
for counter in range(1, Stirrups.number + 1):
    moveToShapeSet( [ Stirrups.name + str(counter) ], Categories.stirrups )

#we define counter again and create a loop to move all the tendons to their category
counter = 1
for counter in range (1,6):
    moveToShapeSet([ActiveReinfo.name + str(counter)], Categories.prestressing)

#we add the materials that we will deal with
#we add the concrete
addMaterial( Materials.concrete, "CONCDC", "EN1992", [ "TOTCRK" ] ) #we add the material
setParameter( "MATERIAL", Materials.concrete, "EC2CON/NORMAL/CLASS", Beam.typeOfConcrete) #we set
the properties of the material
#we add the steel for the plate
addMaterial( Materials.steel, "STEEDC", "EN1993", [] ) #we add the material
setParameter( "MATERIAL", Materials.steel, "EURO93/EN931/GRADE", SteelPlate.typeOfSteel ) #we set
the properties of the material
#we add the steel used in the passive reinforcement
addMaterial( Materials.reinfo_steel, "REINFO", "LINEAR", [ "DESIGN" ] ) #we add the material
setParameter( "MATERIAL", Materials.reinfo_steel, "LINEAR/ELASTI/YOUNG", Rebar.young_modulus ) #we
set the properties of the material
setParameter( "MATERIAL", Materials.reinfo_steel, "REDESI/YLDSTR", Rebar.yield_stress ) #we set the
properties of the material
#we add the steel used in the active reinforcement
addMaterial( Materials.prestress_steel, "REINFO", "LINEAR", [ "NOBOND" ] )
setParameter( MATERIAL, Materials.prestress_steel, "LINEAR/ELASTI/YOUNG", ActiveReinfo.young_modulus
)

#we assign the materials to the geometries created before
#we assign the concrete to the beam
setCurrentShapeSet( Categories.beams )
setElementClassType( "SHAPE", [Beam.name], "STRSOL")
assignMaterial(Materials.concrete, "SHAPE", [Beam.name]) #we assign concrete to block1

#we assign the steel to the plate
setCurrentShapeSet( Categories.plates )
setElementClassType( "SHAPE", [ SteelPlate.name ], "STRSOL" )
assignMaterial( Materials.steel, "SHAPE", [ SteelPlate.name ] ) #we assign steel to block 2
#we assign the other steel to the reinforcement (both rebar and stirrups)

#we define a diametre for the rebar
setCurrentShapeSet( Categories.rebar )
addGeometry( Rebar.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/DIAMET", Rebar.diametre )

#we define a diametre for the stirrups
setCurrentShapeSet( Categories.stirrups )
addGeometry( Stirrups.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/DIAMET", Stirrups.diametre )

#we define a geometry of 18mm diametre for the post-tensioning steel
setCurrentShapeSet( Categories.prestressing )
addGeometry( ActiveReinfo.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", ActiveReinfo.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", ActiveReinfo.name_diametre, "REIEMB/DIAMET", ActiveReinfo.diametre )

#we define counter again and do a loop to assign the geometry and properties to all the rebar
counter = 1
```

```
for counter in range(1, Rebar.number_total+1):
    setReinforcementAspects( [ Rebar.name + str(counter) ] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Rebar.name + str(counter) ] )
    assignGeometry( Rebar.name_diametre, "SHAPE", [ Rebar.name + str(counter) ] )
    resetElementData( "SHAPE", [ Rebar.name + str(counter) ] )
    setReinforcementDiscretization( [ Rebar.name + str(counter) ], "ELEMENT" )

#we define counter again and do a loop to assign the geometry and properties to all the stirrups
counter = 1
for counter in range(counter, Stirrups.number+1):
    setReinforcementAspects( [ Stirrups.name + str(counter)] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Stirrups.name + str(counter) ] )
    assignGeometry( Stirrups.name_diametre, "SHAPE", [ Stirrups.name + str(counter) ] )
    resetElementData( "SHAPE", [ Stirrups.name + str(counter) ] )
    setReinforcementDiscretization( [ Stirrups.name + str(counter) ], "ELEMENT" )

#we define counter again and do a loop to assign the geometry and properties to the post-tensioning
steel bars
counter = 1
for counter in range(1, 6):
    setReinforcementAspects( [ ActiveReinfo.name + str(counter) ] )
    assignMaterial( Materials.prestress_steel, "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    assignGeometry( ActiveReinfo.name_diametre, "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    resetElementData( "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    setReinforcementDiscretization( [ ActiveReinfo.name + str(counter) ], "ELEMENT" )

#we define the supports
addSet( "GEOMETRYSUPPORTSET", Supports.name )
createLineSupport( Supports.name, Supports.name )
setParameter( "GEOMETRYSUPPORT", Supports.name, "AXES", [ PrincipalAxis.x, PrincipalAxis.y ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "TRANSL", [ Supports.x_translation,
Supports.y_translation, Supports.z_translation ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "ROTATI", [ Supports.x_rotation,
Supports.y_rotation, Supports.z_rotation ] )
attach( "GEOMETRYSUPPORT", Supports.name, Beam.name, [[Beam.x0, Beam.width / 2, Beam.z0],
[Beam.span, Beam.width / 2, Beam.z0]]) #coordinates of the lines in both ends of the beam

#we apply the self-weight
addSet( "GEOMETRYLOADSET", Loads.selfweight_name )
createModelLoad( Loads.selfweight_name, Loads.selfweight_name ) #the first name  is the name of the
force and the second the category

#we apply the post-tension load to all the tendons
createBodyLoad( Loads.posttension_name, Loads.posttension_name )
setParameter( "GEOMETRYLOAD", Loads.posttension_name, "LODTYP", "POSTEN" )
setParameter( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/TENTYP", "ONEEND" )
setParameter( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/FORCE1", Loads.tendon_force  )
setParameter( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/SHEAR", Loads.friction_coefficient )
attachTo( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(1),
[[ Beam.span, ActiveReinfo.y1, ActiveReinfo.z1_1 ]] )
attachTo( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(2),
[[ Beam.span, ActiveReinfo.y3, ActiveReinfo.z1_1 ]] )
attachTo( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(3),
[[ Beam.span, ActiveReinfo.y2, ActiveReinfo.z2_1 ]] )
attachTo( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(4),
[[ Beam.span, ActiveReinfo.y1, ActiveReinfo.z3_1 ]] )
attachTo( "GEOMETRYLOAD", Loads.posttension_name, "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(5),
[[ Beam.span, ActiveReinfo.y3, ActiveReinfo.z3_1 ]] )
#we define the counter again and we attach the post-tensioning load to the tendons
counter = 1
for counter in range (1, 6):
    attach( "GEOMETRYLOAD", Loads.posttension_name, [ ActiveReinfo.name + str(counter) ] )

#we apply the load in the coordinates of the point created before
addSet( "GEOMETRYLOADSET", Loads.pointload_name )
createPointLoad( Loads.pointload_name, Loads.pointload_name )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/VALUE", -Loads.pointload )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/DIRECT", Loads.pointload_direction )
attach( "GEOMETRYLOAD", Loads.pointload_name, SteelPlate.name, [[ Point.x, Point.y, Point.z ]] )

#we define the load combinations
setDefaultGeometryLoadCombinations(  )
addGeometryLoadCombination( "" )
```

```
#we mesh the beam and the plate and generate the mesh
setElementSize([Beam.name, SteelPlate.name], MeshProperties.size, -1, True)
setMesherType([Beam.name, SteelPlate.name], "HEXQUAD")
setMidSideNodeLocation([Beam.name, SteelPlate.name], "LINEAR")
generateMesh( [] )

addAnalysis( Analysis.name ) #we create the analysis
addAnalysisCommand( Analysis.name, "NONLIN", Analysis.type ) #we define that the analysis will be
nonlinear
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)", Loads.selfweight_name ) #we
define the name of the first load phase (self-weight)
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR",
FactorCombination.c1 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will add the post-tensioning
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)", Loads.posttension_name ) #we
define the name of the second load phase (post-tensioning)
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR",
FactorCombination.c2 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will change the properties of the reinforcement in the middle section
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)", "Bonding to mother element"
) #we give it the name "bonding to the mother element"
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/LOADNR",
FactorCombination.c1 ) #we select a factorcombination but it doesn't matter which
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/STEPS/EXPLIC/SIZES", "0" )
#we give a value equal to 0 because we don't want to make any change to the beam
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC/BOND", True )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC/BOND/REINFO(11)/RNGNRS",
"ALL" )
#setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC/BOND/REINFO(9)/RNGNRS",
"\"Active rebar2\" \"Active rebar5\" \"Active rebar8\" \"Active rebar11\" \"Active rebar14\"" )#we
select the middle sections

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will add the point-load
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)", Loads.pointload_name ) #we
define the name of the third load phase (point-load)
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/MAXITE",
Analysis.number_iterations_calculation ) #we define the number of iterations in the calculation
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/CONVER/DISPLA/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/CONVER/FORCE/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/STEPS/EXPLIC/SIZES",
Analysis.factor_and_numberIterations )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/LOADNR",
FactorCombination.c3 ) #we define the load combination in the load-steps


setAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/SELTYP", "USER" ) #we define all
the outputs that we want to receive when the analysis is done
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/DISPLA" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRESS" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(1)/CRACK" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(2)/CRKWDT" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STATUS(1)/CRACK" )

runSolver(Analysis.name) #we run the analysis
```

5. Commands: with post-tensioning and poor grouting (Python script)

```
exec(open('C:/Users/marti/Desktop/Python_Scripts/Definitions_prestressing.py').read())

#WE START THE PROGRAM

createBlock(Beam.name, [Beam.x0, Beam.y0, Beam.z0], [Beam.span, Beam.width, Beam.depth]) #we create
the beam
createBlock( SteelPlate.name, [ SteelPlate.x0, SteelPlate.y0, SteelPlate.z0 ], [ SteelPlate.span,
SteelPlate.width, SteelPlate.depth ] ) #we create the steel plate
createPointBody( Point.name, [ Point.x, Point.y, Point.z ] ) #we create the point in which the load
will be applied

projection( "SHAPEFACE", SteelPlate.name, [[ Point.x, Point.y, Point.z ]], [ Point.name ], [ 0, 0, -
1 ], True ) #we imprint the point on the steel plate
removeShape( [ Point.name ] ) #we remove the point created before. now it is part of the steel plate

#we create a for loop in order to create the inferior passive reinforcement
counter = 1
for counter in range(1, Rebar.number_inferior+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_inferior ], [Beam.span,
Rebar.y, Rebar.z_inferior])
    Rebar.y = Rebar.y + Rebar.distance_between

#we create a for loop in order to create the superior passive reinforcement
Rebar.y = Covers.horizontal #we define again the distance of the rebar from the edge to create the
rebar in the superior part
for counter in range(Rebar.number_inferior+1, Rebar.number_total+1):
    createLine(Rebar.name + str(counter), [ Rebar.x, Rebar.y, Rebar.z_superior ], [Beam.span,
Rebar.y, Rebar.z_superior])
    Rebar.y = Rebar.y + Rebar.distance_between #we separate the bars from one another with the
distance settled before by changig the Y position

#we define counter again and create a loop to create the polyline
counter = 1
for a in range(1, Stirrups.number+1):
    createPolyline(Stirrups.name + str(a), [[Stirrups.x, Stirrups.y1, Stirrups.z1], [Stirrups.x,
Stirrups.y2, Stirrups.z1], [Stirrups.x, Stirrups.y2, Stirrups.z2], [Stirrups.x, Stirrups.y1,
Stirrups.z2]], True )
    Stirrups.x = Stirrups.x + Stirrups.distance_between

#we create the active reinforcement
createCurve(ActiveReinfo.name + str(1), [[ 0, 0.1, 0.25 ],[ 1, 0.1, 0.19 ],[ 2, 0.1, 0.15 ]] )
createCurve(ActiveReinfo.name + str(2), [[ 2, 0.1, 0.15 ],[ 3, 0.1, 0.12 ],[ 4, 0.1, 0.15 ]] )
createCurve(ActiveReinfo.name + str(3), [[ 4, 0.1, 0.15 ],[ 5, 0.1, 0.19 ],[ 6, 0.1, 0.25 ]] )
createCurve(ActiveReinfo.name + str(4), [[ 0, 0.3, 0.25 ],[ 1, 0.3, 0.19 ],[ 2, 0.3, 0.15 ]] )
createCurve(ActiveReinfo.name + str(5), [[ 2, 0.3, 0.15 ],[ 3, 0.3, 0.12 ],[ 4, 0.3, 0.15 ]] )
createCurve(ActiveReinfo.name + str(6), [[ 4, 0.3, 0.15 ],[ 5, 0.3, 0.19 ],[ 6, 0.3, 0.25 ]] )

createCurve(ActiveReinfo.name + str(7), [[ 0, 0.2, 0.3 ],[ 1, 0.2, 0.24 ],[ 2, 0.2, 0.2 ]] )
createCurve(ActiveReinfo.name + str(8), [[ 2, 0.2, 0.2 ],[ 3, 0.2, 0.17 ],[ 4, 0.2, 0.2 ]] )
createCurve(ActiveReinfo.name + str(9), [[ 4, 0.2, 0.2 ],[ 5, 0.2, 0.24 ],[ 6, 0.2, 0.3 ]] )

createCurve(ActiveReinfo.name + str(10), [[ 0, 0.1, 0.35 ],[ 1, 0.1, 0.29 ],[ 2, 0.1, 0.25 ]] )
createCurve(ActiveReinfo.name + str(11), [[ 2, 0.1, 0.25 ],[ 3, 0.1, 0.22 ],[ 4, 0.1, 0.25 ]] )
createCurve(ActiveReinfo.name + str(12), [[ 4, 0.1, 0.25 ],[ 5, 0.1, 0.29 ],[ 6, 0.1, 0.35 ]] )
createCurve(ActiveReinfo.name + str(13), [[ 0, 0.3, 0.35 ],[ 1, 0.3, 0.29 ],[ 2, 0.3, 0.25 ]] )
createCurve(ActiveReinfo.name + str(14), [[ 2, 0.3, 0.25 ],[ 3, 0.3, 0.22 ],[ 4, 0.3, 0.25 ]] )
createCurve(ActiveReinfo.name + str(15), [[ 4, 0.3, 0.25 ],[ 5, 0.3, 0.29 ],[ 6, 0.3, 0.35 ]] )

#we rename and create the needed categories
rename( "SHAPESET", "Shapes", Categories.beams ) #we change the name of the folder. from shapes to
beam
addSet( "SHAPESET", Categories.plates ) #we add the category for plates
addSet( "SHAPESET", Categories.rebar ) #we add the category for rebar
addSet( "SHAPESET", Categories.stirrups ) #we add the category for stirrups
addSet( "SHAPESET", Categories.prestressing ) #we add the category for stirrups

#we move all the elements to their respectives categories
moveToShapeSet( [ SteelPlate.name ], Categories.plates ) #we move the plate from the category "beam"
to the category "plates"
#we define counter again and create a loop to move all the rebar to their category
```

```
counter = 1
for counter in range(1, Rebar.number_total + 1):
    moveToShapeSet( [ Rebar.name + str(counter) ], Categories.rebar )

#we define counter again and create a loop to move all the stirrups to their category
counter = 1
for counter in range(1, Stirrups.number + 1):
    moveToShapeSet( [ Stirrups.name + str(counter) ], Categories.stirrups )

#we define counter again and create a loop to move all the tendons to their category
counter = 1
for counter in range (1,16):
    moveToShapeSet([ActiveReinfo.name + str(counter)], Categories.prestressing)

#we add the materials that we will deal with
#we add the concrete
addMaterial( Materials.concrete, "CONCDC", "EN1992", [ "TOTCRK" ] ) #we add the material
setParameter( "MATERIAL", Materials.concrete, "EC2CON/NORMAL/CLASS", Beam.typeOfConcrete) #we set
the properties of the material
#we add the steel for the plate
addMaterial( Materials.steel, "STEEDC", "EN1993", [] ) #we add the material
setParameter( "MATERIAL", Materials.steel, "EURO93/EN931/GRADE", SteelPlate.typeOfSteel ) #we set
the properties of the material
#we add the steel used in the passive reinforcement
addMaterial( Materials.reinfo_steel, "REINFO", "LINEAR", [ "DESIGN" ] ) #we add the material
setParameter( "MATERIAL", Materials.reinfo_steel, "LINEAR/ELASTI/YOUNG", Rebar.young_modulus ) #we
set the properties of the material
setParameter( "MATERIAL", Materials.reinfo_steel, "REDESI/YLDSTR", Rebar.yield_stress ) #we set the
properties of the material
#we add the steel used in the active reinforcement
addMaterial( Materials.prestress_steel, "REINFO", "LINEAR", [ "NOBOND" ] )
setParameter( MATERIAL, Materials.prestress_steel, "LINEAR/ELASTI/YOUNG", ActiveReinfo.young_modulus
)

#we assign the materials to the geometries created before
#we assign the concrete to the beam
setCurrentShapeSet( Categories.beams )
setElementClassType( "SHAPE", [Beam.name], "STRSOL")
assignMaterial(Materials.concrete, "SHAPE", [Beam.name]) #we assign concrete to block1

#we assign the steel to the plate
setCurrentShapeSet( Categories.plates )
setElementClassType( "SHAPE", [ SteelPlate.name ], "STRSOL" )
assignMaterial( Materials.steel, "SHAPE", [ SteelPlate.name ] ) #we assign steel to block 2
#we assign the other steel to the reinforcement (both rebar and stirrups)

#we define a diametre for the rebar
setCurrentShapeSet( Categories.rebar )
addGeometry( Rebar.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Rebar.name_diametre, "REIEMB/DIAMET", Rebar.diametre )

#we define a diametre for the stirrups
setCurrentShapeSet( Categories.stirrups )
addGeometry( Stirrups.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", Stirrups.name_diametre, "REIEMB/DIAMET", Stirrups.diametre )

#we define a geometry of 18mm diametre for the post-tensioning steel
setCurrentShapeSet( Categories.prestressing )
addGeometry( ActiveReinfo.name_diametre, "RELINE", "REBAR", [] )
setParameter( "GEOMET", ActiveReinfo.name_diametre, "REIEMB/RDITYP", "RDIAME" )
setParameter( "GEOMET", ActiveReinfo.name_diametre, "REIEMB/DIAMET", ActiveReinfo.diametre )

#we define counter again and do a loop to assign the geometry and properties to all the rebar
counter = 1
for counter in range(1, Rebar.number_total+1):
    setReinforcementAspects( [ Rebar.name + str(counter) ] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Rebar.name + str(counter) ] )
    assignGeometry( Rebar.name_diametre, "SHAPE", [ Rebar.name + str(counter) ] )
    resetElementData( "SHAPE", [ Rebar.name + str(counter) ] )
    setReinforcementDiscretization( [ Rebar.name + str(counter) ], "ELEMENT" )

#we define counter again and do a loop to assign the geometry and properties to all the stirrups
```

```
counter = 1
for counter in range(counter, Stirrups.number+1):
    setReinforcementAspects( [ Stirrups.name + str(counter)] )
    assignMaterial( Materials.reinfo_steel, "SHAPE", [ Stirrups.name + str(counter) ] )
    assignGeometry( Stirrups.name_diametre, "SHAPE", [ Stirrups.name + str(counter) ] )
    resetElementData( "SHAPE", [ Stirrups.name + str(counter) ] )
    setReinforcementDiscretization( [ Stirrups.name + str(counter) ], "ELEMENT" )

#we define counter again and do a loop to assign the geometry and properties to the post-tensioning
steel bars
counter = 1
for counter in range(1, 16):
    setReinforcementAspects( [ ActiveReinfo.name + str(counter) ] )
    assignMaterial( Materials.prestress_steel, "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    assignGeometry( ActiveReinfo.name_diametre, "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    resetElementData( "SHAPE", [ ActiveReinfo.name + str(counter) ] )
    setReinforcementDiscretization( [ ActiveReinfo.name + str(counter) ], "ELEMENT" )

#we define the supports
addSet( "GEOMETRYSUPPORTSET", Supports.name )
createLineSupport( Supports.name, Supports.name )
setParameter( "GEOMETRYSUPPORT", Supports.name, "AXES", [ PrincipalAxis.x, PrincipalAxis.y ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "TRANSL", [ Supports.x_translation,
Supports.y_translation, Supports.z_translation ] )
setParameter( "GEOMETRYSUPPORT", Supports.name, "ROTATI", [ Supports.x_rotation,
Supports.y_rotation, Supports.z_rotation ] )
attach( "GEOMETRYSUPPORT", Supports.name, Beam.name, [[Beam.x0, Beam.width / 2, Beam.z0],
[Beam.span, Beam.width / 2, Beam.z0]]) #coordinates of the lines in both ends of the beam

#we apply the self-weight
addSet( "GEOMETRYLOADSET", Loads.selfweight_name )
createModelLoad( Loads.selfweight_name, Loads.selfweight_name ) #the first name  is the name of the
force and the second the category

#we apply the post-tension load to all the tendons
#first the set of sections on the left
addSet( "GEOMETRYLOADSET", Loads.posttension_name)
createBodyLoad( "Left prestressing", Loads.posttension_name )
setParameter( "GEOMETRYLOAD", "Left prestressing", "LODTYP", "POSTEN" )
setParameter( "GEOMETRYLOAD", "Left prestressing", "POSTEN/TENTYP", "ONEEND" )
setParameter( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/FORCE1", Loads.tendon_force  )
setParameter( "GEOMETRYLOAD", "Left prestressing", "POSTEN/SHEAR", Loads.friction_coefficient )
attachTo( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(1), [[
0,0.1,0.25 ]] )
attachTo( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(4), [[
0,0.3,0.25 ]] )
attachTo( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(7), [[
0,0.2,0.3 ]] )
attachTo( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(10),
[[ 0,0.3,0.35  ]] )
attachTo( "GEOMETRYLOAD", "Left prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(13),
[[ 0,0.2,0.35]] )
attach( "GEOMETRYLOAD", "Left prestressing", [ ActiveReinfo.name + str(1), ActiveReinfo.name +
str(4), ActiveReinfo.name + str(7), ActiveReinfo.name + str(10), ActiveReinfo.name + str(13) ] )
#second the set of sections on the left
createBodyLoad( "Right prestressing", Loads.posttension_name )
setParameter( "GEOMETRYLOAD", "Right prestressing", "LODTYP", "POSTEN" )
setParameter( "GEOMETRYLOAD", "Right prestressing", "POSTEN/TENTYP", "ONEEND" )
setParameter( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/FORCE1", Loads.tendon_force  )
setParameter( "GEOMETRYLOAD", "Right prestressing", "POSTEN/SHEAR", Loads.friction_coefficient )
attachTo( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(3),
[[ 6,0.1,0.25 ]] )
attachTo( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(6),
[[ 6,0.3,0.25 ]] )
attachTo( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(9),
[[ 6,0.2,0.3 ]] )
attachTo( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(12),
[[ 6,0.3,0.35  ]] )
attachTo( "GEOMETRYLOAD", "Right prestressing", "POSTEN/ONEEND/PNTS1", ActiveReinfo.name + str(15),
[[ 6,0.2,0.35]] )
attach( "GEOMETRYLOAD", "Right prestressing", [ ActiveReinfo.name + str(3), ActiveReinfo.name +
str(6), ActiveReinfo.name + str(9), ActiveReinfo.name + str(12), ActiveReinfo.name + str(15) ] )

#we apply the load in the coordinates of the point created before
```

```
addSet( "GEOMETRYLOADSET", Loads.pointload_name )
createPointLoad( Loads.pointload_name, Loads.pointload_name )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/VALUE", -Loads.pointload )
setParameter( "GEOMETRYLOAD", Loads.pointload_name, "FORCE/DIRECT", Loads.pointload_direction )
attach( "GEOMETRYLOAD", Loads.pointload_name, SteelPlate.name, [[ Point.x, Point.y, Point.z ]] )

#we define the load combinations
setDefaultGeometryLoadCombinations(  )
addGeometryLoadCombination( "" )

#we mesh the beam and the plate and generate the mesh
setElementSize([Beam.name, SteelPlate.name], MeshProperties.size, -1, True)
setMesherType([Beam.name, SteelPlate.name], "HEXQUAD")
setMidSideNodeLocation([Beam.name, SteelPlate.name], "LINEAR")
generateMesh( [] )

addAnalysis( Analysis.name ) #we create the analysis
addAnalysisCommand( Analysis.name, "NONLIN", Analysis.type ) #we define that the analysis will be
nonlinear
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)", Loads.selfweight_name ) #we
define the name of the first load phase (self-weight)
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(1)/LOAD/LOADNR",
FactorCombination.c1 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will add the prestressing
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)", Loads.posttension_name ) #we
define the name of the second load phase (prestressing)
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(2)/LOAD/LOADNR",
FactorCombination.c2 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will change the properties of the reinforcement in the middle section
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)", "Bonding to mother element"
) #we give it the name "bonding to the mother element"
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/LOADNR",
FactorCombination.c1 ) #we select a factorcombination but it doesn't matter which
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/LOAD/STEPS/EXPLIC/SIZES", "0" )
#we give a value equal to 0 because we don't want to make any change to the beam
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC/BOND", True )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(3)/PHYSIC/BOND/REINFO(9)/RNGNRS",
"\"Active rebar2\" \"Active rebar5\" \"Active rebar8\" \"Active rebar11\" \"Active rebar14\"" )#we
select the middle sections

setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT/EXETYP", "LOAD" ) #we create another
phase where we will add the point-load
renameAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)", Loads.pointload_name ) #we
define the name of the third load phase (point-load)
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/MAXITE",
Analysis.number_iterations_calculation ) #we define the number of iterations in the calculation
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/CONVER/DISPLA/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/ITERAT/CONVER/FORCE/NOCONV",
"CONTIN" ) #we want the analysis to continue even though it doesn't converge
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/STEPS/EXPLIC/SIZES",
Analysis.factor_and_numberIterations )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/LOADNR" )
setAnalysisCommandDetail( Analysis.name, Analysis.type, "EXECUT(4)/LOAD/LOADNR",
FactorCombination.c3 ) #we define the load combination in the load-steps

setAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/SELTYP", "USER" ) #we define all
the outputs that we want to receive when the analysis is done
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/DISPLA" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRESS" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(1)/CRACK" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STRAIN(2)/CRKWDT" )
addAnalysisCommandDetail( Analysis.name, Analysis.type, "OUTPUT(1)/USER/STATUS(1)/CRACK" )

runSolver(Analysis.name) #we run the analysis
```