Expanding the implementation of Macaulay's method in Python

M.J. van Gelder

TU Delft

June 2024

Preface

This report is the result of the final project for my Bachelor's degree in Civil Engineering & Geoscience. The report focuses on the implementation of Macaulay's method in Python. The report is interesting for civil engineers with knowledge of Python.

I would like to thank my supervisors, Tom van Woudenberg and Jason Moore, for their guidance and support during the project. Their insights were valuable and their feedback improved the quality of my work.

During this project I learned a lot about the steps necessary to implement mechanics calculations into computer software. The process of implementing these mechanics calculations was both challenging and rewarding. I hope this report provides useful insights and can help future improvements of the mechanics calculations in Python.

Mark van Gelder

June 2024

Abstract

The method of Macaulay is a method to determine force and deflection properties of structures. The method makes it possible to describe discontinuous beams with a single equation. The fact that the entire beam can be described in a single equation makes this method well-suited for use in programming. In past Bachelor End Projects students from the TU Delft have made advancements in the application of the method of Macaulay, making it able do be used for more complicated structures.

SymPy is a Python library that has a Beam module that specialises in calculations of beams. In this module the method of Macaulay is used for some of these calculations. The goal of this project is to use the advancements made by previous Bachelor End Projects to extend the implementation of Macaulay's method in SymPy.

This leads to the following research question: *How can the current implementation of Macaulay's method be extended in Python to be able to calculate and analyze more complicated beams and structures, based on the advancement made by previous Bachelor End Projects at the TU Delft?*

The advancements made by previous Bachelor End Projects can be divided up into different subjects. These subjects can be implemented one by one. During the course of the project there has been focus on implementing two of these subjects. The first subject is rotation and sliding hinges. The second subject is calculations of influence line diagrams. Both these subjects are completely implemented into the SymPy code.

In the implementation of rotation and sliding hinges, the mechanics calculation done by the SymPy code are changed. These hinges are now directly added to the load equation of the beam using singularity functions. The main advantage of this is that calculations on beams with hinges can be done on the beam as a whole, instead of having to cut the beam up into different parts. The fact that the beam can be calculated as a whole also makes this method able to be scaled to calculate beams with multiple hinges.

There was already an existing implementation to calculate influence line diagrams, but this implementation could be improved. In the new implementation the moving load is added to the load equation using a singularity function. The ability to apply the boundary conditions that come with hinges is also added in the new implementation. This makes the new implementation able to calculate correct influence lines for beams with and without hinges.

There are more advancements made by previous Bachelor End Projects than implemented during this project. The advancements on some other subjects, for example normal forces or spring connections, can still be implemented into the current Beam module. For other advancements, those regarding 2D structures, it will be better if they are implemented in a new module. This new module could specialize in 2D structures, while the current module stays focused on calculations on single beams.

In the end it can be concluded that the implementation of Macaulay's method can be extended in Python by first taking a look at the mechanics calculations. In these calculations the method of Macaulay should be used in the most optimal way. After changing these calculations, the software design can be adjusted to be able to make these calculations. With the implementation of these new mechanics calculations the software can calculate and analyse more complicated beams and structures.

Contents

1 Introduction

The calculation and analysis of beams are a fundamental part of structural engineering. There are a lot of computer tools that provide possibilities to do this, one of these tools is SymPy. The current possibilities to calculate and analyse beams in SymPy are limited. The focus of this project will be to address these limitations by extending SymPy's capabilities to calculate more complex beams using the method of Macaulay. This extension of SymPy's capabilities using Macaulay's method will be based on the work of previous Bachelor End Projects of TU delft students.

1.1 Background

The method of Macaulay makes it possible to describe discontinuous beams with a single equation. The fact that the entire beam can be described in a single equation makes this method well-suited for use in programming. More information on the method of Macaulay and it's application can be found in chapter 1 of the document "Complete description of current extensions"(in Dutch) (TU Delft, 2024).

The current implementations of Macaulay's method are made in SymPy. SymPy is a Python library designed for symbolic mathematics. It has a specialized module that focuses on beam calculations. SymPy is an open-source Python library that aims to make contributing accessible to new users. It uses Git for source control and is hosted on GitHub. This means that all SymPy code is available in a repository on GitHub. Users can pull and push code from and to this repository. When a user makes an improvement to the code, a pull request can be made. This gives other contributors the possibility to review the code and give feedback if necessary. A contributor with commit rights, who is someone that has contributed a lot and has authority to accept code, can accept the pull request if it is good enough.

To improve the implementation of the method of Macaulay in SymPy, research done by previous bachelor students during their Bachelor End Projects can be used. The last year multiple TU Delft students have focused on expanding the current applications of the method of Macaulay's during their Bachelor End Projects. Their research focused on applying the method of Macaulay on different topics. The findings of these students are described in the document "Complete description of current extensions"(in Dutch) (TU Delft, 2024). The topics these student researched can be divided up into 6 main parts.

- Normal forces (Van der Wulp, 2023)
- Rotation and sliding hinges (Van der Wulp, 2023)
- Spring supports and spring connections (Van der Wulp, 2023)
- Influence lines (Jankie, 2023)
- Inclined and curved beams (Qadriyeh, 2023)
- 2D structures (Van der Wulp, 2023) & (Qadriyeh, 2023) & (Baudoin, 2024)

1.2 Issues identified in previous research

1.2.1 Modelling hinges

In the report of Van der Wulp it is described how to model rotation and sliding hinges using singularity functions. The report states that a rotation hinge can be described by a singularity function of order -3 multiplied by an unknown ϕ. This ϕ is the rotation of the hinge. In total this gives the possibility to describe a rotation hinge at a location a as $\phi < x - a >^{-3}$ (Van der Wulp, 2023).

It is also stated that a sliding hinge can be described in the same way. This can be done using a singularity function of order -4 multiplied by an unknown w. Here, w is the deflection in the sliding hinge. A sliding hinge at location a can thus be modelled as $w < x - a >^{-4}$.

While making some calculations with these rotation and sliding hinges it came to light that the values found for ϕ and w were not correct. This issue will impact the python implementation of these rotation and sliding hinges and should be solved beforehand.

1.2.2 Singularity functions that return infinity

In the method of Macaulay forces on the beam and connections in the beam are modelled in the load equation using singularity functions. For a lot of these forces and connections singularity functions of negative order are used to describe them. A singularity function of negative order looks like $< x - a >^{-n}$ with $n > 0$ and a the location of the singularity. These singularity functions of negative order are 0 everywhere except for when $x = a$, then the function is evaluated as infinity.

A singularity function of negative order does not have a physical meaning in the equations for the beam responses. These singularity functions should not influence equations until they are integrated to a non-negative order. The fact that singularity functions of negative order can return infinity gives an issue.

To illustrate the problem an example structure is given in figure 1.

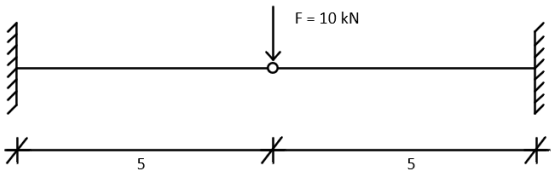


Figure 1: Example 1: Structure with a rotation hinge.

The load equation for the structure shown in figure 1 is given in equation 1.

The code for this example is visible here.

Load:

$$EIP_5 < x-5 >^{-3} -F < x-5 >^{-1} +M_0 < x >^{-2} +M_{10} < x-10 >^{-2} +R_0 < x >^{-1} +R_{10} < x-10 >^{-1} \tag{1}$$

Integrating this load equation twice gives the bending moment equation.

Bending moment:

$$-EIP_5 < x-5 >^{-1} +F < x-5 >^{1} -M_0 < x >^{0} -M_{10} < x-10 >^{0} -R_0 < x >^{1} -R_{10} < x-10 >^{1} \tag{2}$$

One of the boundary conditions for this structure is that the bending moment is zero at the location of the hinge. Filling in $x = 5$ into equation 2 and setting it equal to 0 results in the following equation.

$$-\infty EIP_5 - M_0 - 5R_0 = 0 \tag{3}$$

This equation is incorrect. Solving this equation will not lead to the correct reaction forces. The term EIP_5 should not be present in this equation. This equation should be:

$$-M_0 - 5R_0 = 0 \tag{4}$$

This issue impacts the python implementation. A solution for this problem should be found, in order to get the correct results.

1.3 Coordinate system used in SymPy

The standard coordinate system in SymPy is different to the one used at the TU Delft. In this document all plots and equations will be given in the SymPy coordinate system. In this section the SymPy coordinate system is explained with the use of an example structure.

The Beam module in SymPy has a horizontal x-axis. When a beam is defined it is placed on this axis. The beam starts at $x = 0$ and ends at $x = l$, with l the length of the beam. The positive x-axis is to the right. There is also a vertical axis in SymPy, the name of this axis is not defined. The positive direction of this vertical axis is upward.

In figure 2 an example structure is given. This example structure will be used to demonstrate the coordinate system in SymPy. The positive directions of the axis are shown in the figure.

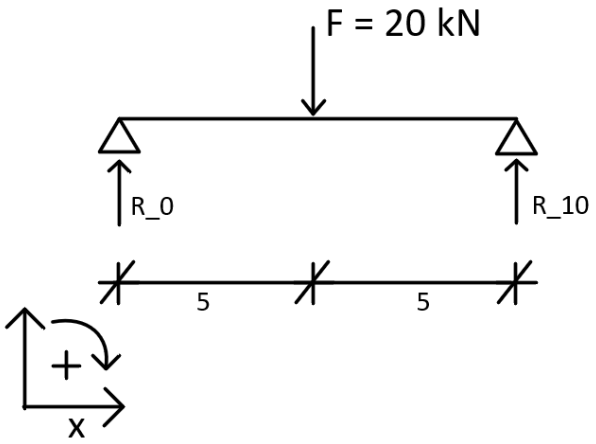


Figure 2: Example 2: Structure with point load.

There are 2 reaction loads visible in the figure, 1 at each support. These reaction loads are named automatically by SymPy when a support is placed. A point load in a support is named as R_loc and a moment load in a support is named as M_loc, with loc the x coordinate of the support. The reaction loads are shown in math notation as R_{loc} and M_{loc}. The positive direction for a point load is upward. The positive direction for a moment load is clockwise.

The positive directions for the internal forces are shown in figure 3.

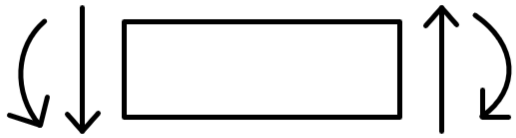
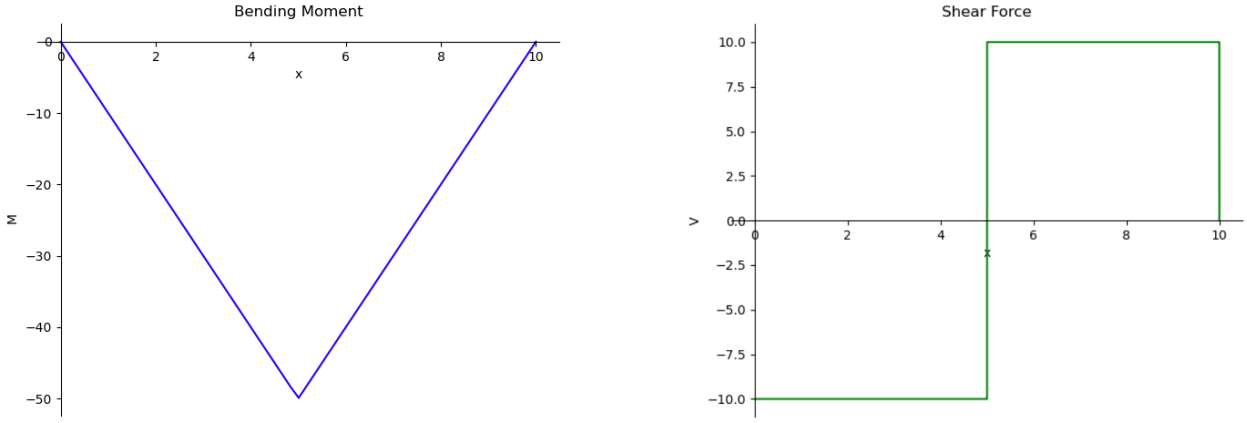


Figure 3: Positive directions of internal forces.

In the structure shown in figure 2 it is visible that there is a point load working downward on the beam. This point load will be modelled as a point load of magnitude -20 in the code. The minus is added because the load works in the negative vertical direction.

After defining this structure using the Beam module, the reaction loads and bending moment line can be calculated. The output for the reaction forces is shown in equation 5. The bending moment line and shear force line are plotted in figure 4.

The code for this example is visible here.



(a) Bending moment line

(b) Shear force line

Figure 4: Example 2: Plots of cross sectional properties.

$$\{R_0 : 10, \quad R_{10} : 10\} \tag{5}$$

In equation 5 it is visible that the methods return positive values for the reaction loads. This means that they work in upward direction, which is correct.

The bending moment line and shear force line also return the correct values following the positive directions for internal forces specified in figure 3. Note that the negative axis is placed downwards in the plots. This differs from the standard TU Delft notation.

1.4 Current possibilities in SymPy

The current implementations in SymPy make it possible to calculate simple beams and provide several possibilities to analyse results, such as printing and plotting equations. The functions that make this possible already use the method of Macaulay. However, there are some limitations to what beams can be calculated.

Some of the topics researched by previous bachelor students are not implemented in SymPy. There are no functions present in the Beam module that describe these phenomena. This is the case for:

- Normal forces
- Sliding hinges
- Spring supports and spring connections
- Inclined and curved beams
- 2D structures

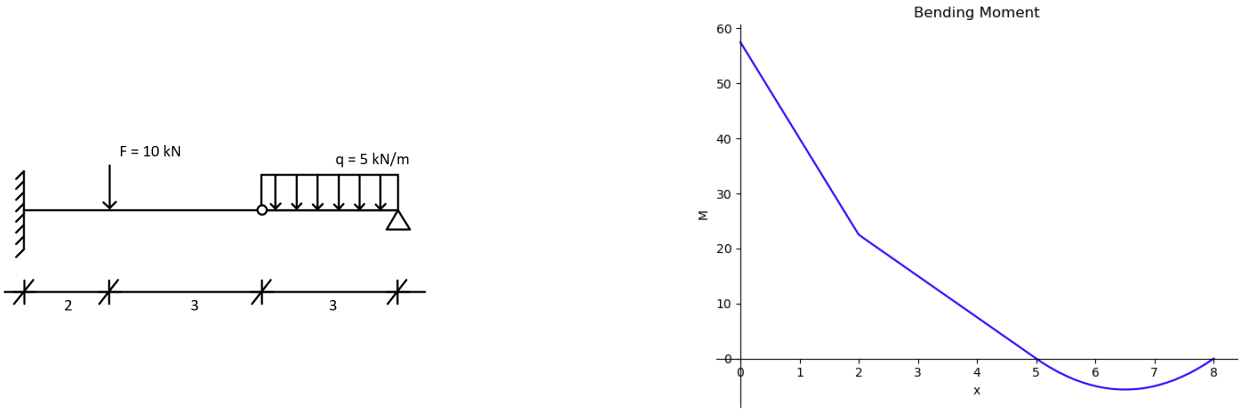
The other two topics, rotation hinges and influence lines, are partly implemented. The possibilities in the current implementation will be discussed and shown with examples. First the rotation hinges and then the influence lines.

1.4.1 Rotation hinges

It is already possible to calculate beams with a single rotation hinge. Adding multiple rotation hinges in a beam is not possible.

An example of a beam with a single hinge that can be calculated is given in figure 5. The bending moment line is given as output.

The code for this example is visible here.



(a) Drawing of structure

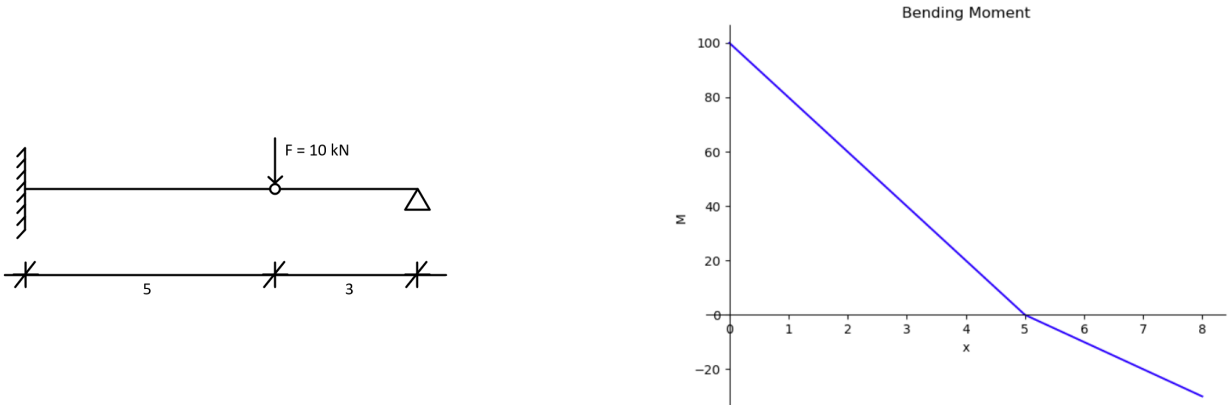
(b) Plot of the bending moment line

Figure 5: Example 3: Beam with a single hinge.

The method to calculate beams with a single hinge is not perfect. Sometimes the method generates the wrong results. This happens when a force is placed directly on the hinge.

An example of a structure that generates a wrong result is given in figure 6. Again the bending moment line is given as output. This line is incorrect, the correct bending moment line is given in figure 7.

The code that calculates the incorrect result for this example is visible here.
The code that is used to calculate the correct result is visible here.



(a) Drawing of structure

(b) Plot of the bending moment line

Figure 6: Example 4: Beam with a single hinge that generates an incorrect output.

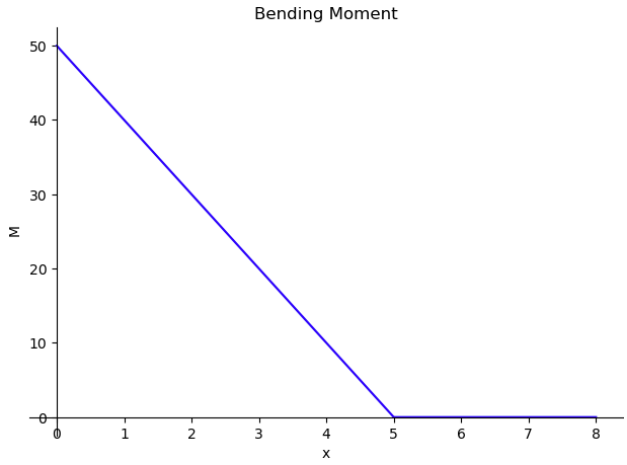


Figure 7: Plot of the correct bending moment line for example 4.

The Beam module should not return incorrect outputs. The code does not calculate beams with a single hinge as well as it should right now. In addition to this it is also unable to calculate beams with more than one hinge. In figure 8 an example is given of a structure with multiple hinges, this structure cannot be calculated right now.

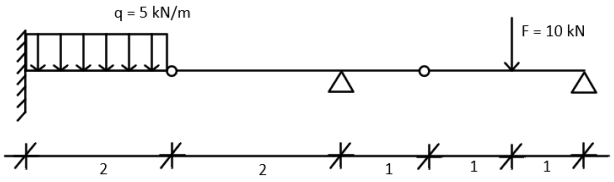


Figure 8: Structure with multiple hinges.

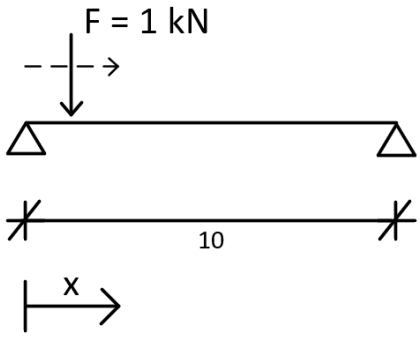
1.4.2 Influence lines

There is a possibility to calculate influence line diagrams using SymPy. The reaction loads due to a moving load over the beam can be found and plotted. These reaction loads are also used to find the influence line equations for shear force and bending moment at a specified point in a Beam object.

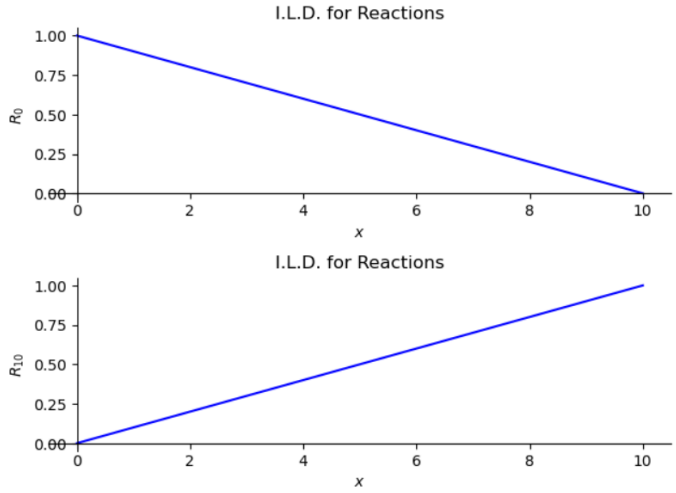
The current implementation of influence line calculations is limited in the amount of structures for which it can calculate the correct influence line diagrams. The current methods can only calculate influence line diagrams for statically determinate structures without hinges. For structures with hinges or statically indeterminate structures the methods will output incorrect results.

In figure 9 a statically determinate structure is given for which the methods output the correct results. The structure is drawn and the influence lines for the reaction forces are plotted.

The code for this example is visible here.



(a) Drawing of structure



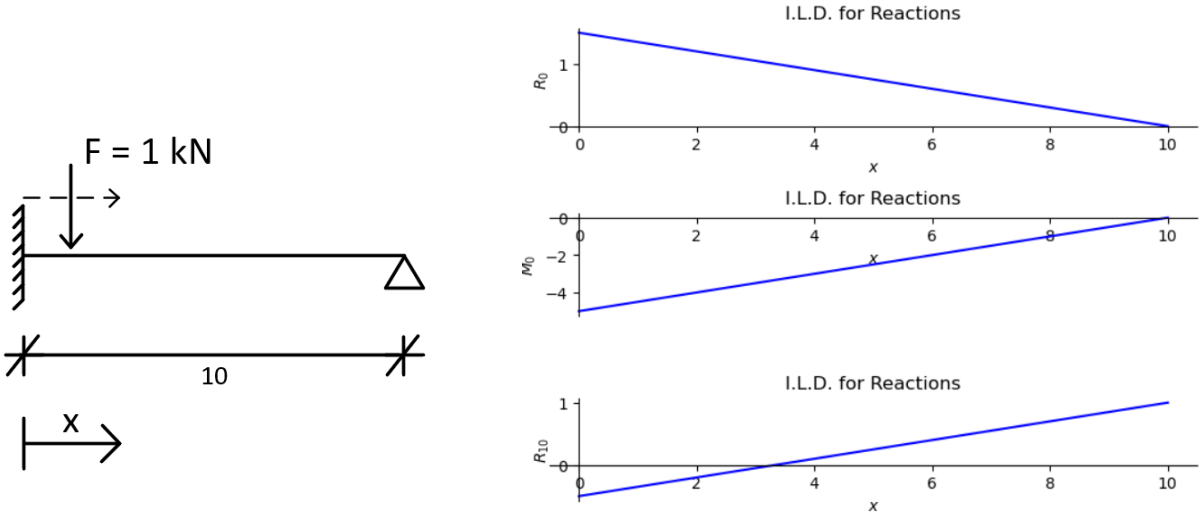
(b) Plot of the influence lines for the reaction forces

Figure 9: Example 5: Correct influence lines for statically determinate structure.

If a structure is statically indeterminate or includes hinges the methods will produce incorrect outputs.

An example of this for a statically indeterminate structure is visualised in figure 10. The plot of the influence lines for the reaction loads is incorrect. The correct plot for these reactions loads is shown in figure 11

The code that calculates the incorrect result for this example is visible here.
The code that is used to calculate the correct result is visible here.



(a) Drawing of structure

(b) Plot of the incorrect influence lines for the reaction forces

Figure 10: Example 6: Incorrect influence lines returned by SymPy.

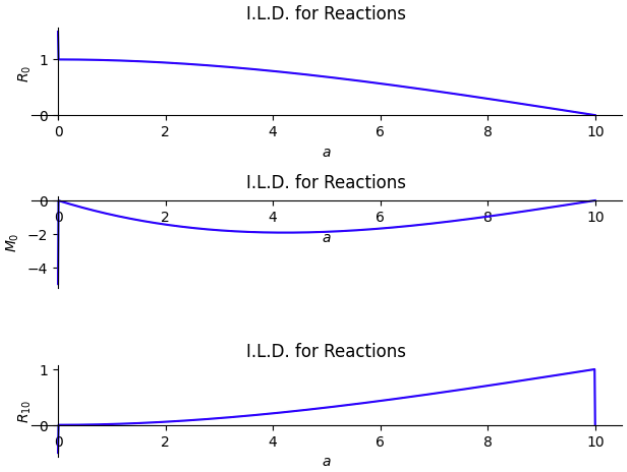


Figure 11: Plot of correct influence lines for reactions loads for example 6.

The methods to calculate influence line diagrams have no code that check for and handle the calculation of hinges. This means that the methods will produce output like the hinge does not exist.

1.5 Objective

The goal of this project is to extend the current Python implementation in such a way that it can calculate more complicated beams, based on the work done in previous Bachelor End Projects by TU delft students.

With this goal in mind we can formulate a research question:
How can the current implementation of Macaulay's method be extended in Python to be able to calculate and analyze more complicated beams and structures, based on the advancement made by previous Bachelor End Projects at the TU Delft?

1.6 Methodology

To investigate how the use of the method of Macaulay's can be extended in the current SymPy implementation, this project will use a step wise approach. First the issues identified in previous research will be addressed. The correct way to model rotation and sliding hinges should be found. There should also be a solution for the problem that is posted by the singularity functions that return infinity.

After these issues are resolved, the correct models will be implemented in SymPy. The implementation of the correct models will be split up into different topics. The topics will be implemented one by one in order to keep it clear which changes are completed. This one by one approach will also make sure that implementations are complete and correct before switching focus to different topics.

The implementation of a topic will be done by first focusing on the mechanics calculations regarding that topic. If these calculations are already present they will be analysed. If the current calculations can be improved by making use of the method of Macaulay they will be changed. If there are no mechanics calculations present on a topic they can be set up using the method of Macaulay.

After the new mechanics calculations are defined the software will be changed. The software will be adjusted to give it the ability to perform the newly defined mechanics calculations.

1.7 Scope

The project focuses on extending the possibilities to calculate beams in SymPy using Macaulay's method. During the project the newfound applications of the method of Macaulay will be implemented into the SymPy code. This implementation into the SymPy code will happen by making pull requests on GitHub. These requests will be made during the project and results of these requests will be discussed.

As this project has a deadline it will not be possible to continue with the project as long as it takes to implement all newfound application of the method of Macaulay. The newfound applications are separated by topic and these topics will be implemented one by one. During the project as much topics as possible will be implemented, while maintaining the necessary standard for the code.

The SymPy module that specializes in beam calculations can be improved in a lot of different ways. For example by removing bugs, adding tests for existing methods and writing new methods. This project does not focus on the overall improvement of the Beam module in SymPy. It only focuses on writing new methods that implement extra possibilities of Macaulay's method. This means that the removal of bugs and addition of tests is not a priority and will not be done if not relevant to the project. Of course, the improvements made during the project should be free of bugs and tested extensively.

1.8 Examples

The code for the examples shown in this report is saved in two repositories on GitHub. One of these repositories includes the examples that are calculated using the current SymPy code. These examples are made using SymPy 1.12.1. The repository that includes these examples can be found here.

The other repository includes the examples that are made in the SymPy 1.14-dev version. This version includes the changes made during the project. This repository uses the SymPy fork in which the changes are made in order to present the most up to date results. This repository can be found here.

2 Addressing identified issues

2.1 Modelling hinges

During research on previous works an issues was identified that had to to with the modelling of rotation and sliding hinges in the load equation. Van der Wulp states that rotation hinges can be represented by $\phi < x - a >^{-3}$ and sliding hinges by $w < x - a >^{-4}$ in the load equation. The ϕ is the rotation in the rotation hinge. This can be seen as a jump in the slope plot. The w is the deflection in the sliding hinge, which can be seen as a jump in the deflection plot (Van der Wulp, 2023).

If a rotation hinge is modelled as $\phi < x - a >^{-3}$ in the load equation it will be $\frac{\phi}{EI} < x - a >^{0}$ in the slope equation. The singularity function of $< x - a >^{0}$ represents a jump from 0 to 1 at location $x = a$. This means that the jump in the slope line would be of size $\frac{\phi}{EI}$, while it should be of size ϕ.

The correct way to model the rotation hinge, to get ϕ to be the rotation in hinge, would be $EI\phi < x - a >^{-3}$. In the slope equation this would become $\phi < x - a >^{0}$, which represents a jump of size ϕ.

With the same reasoning the model of the sliding hinge can be changed. This should be $EIw < x - a >^{-4}$ in the load equation. In the deflection equation this would become $w < x - a >^{0}$, which represents a jump of magnitude w.

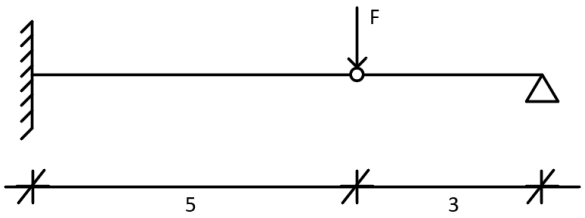


Figure 12: Structure with hinge.

As an example the load equation for the structure in figure 12 is given. Note that the rotation hinge is modelled as $EI\phi < x - 5 >^{-3}$.

$$\text{Load} : EI\phi < x - 5 >^{-3} - F < x - 5 >^{-1} + M_0 < x >^{-2} + R_0 < x >^{-1} + R_8 < x - 8 >^{-1} \tag{6}$$

This equation can be integrated to get the equations for shear force and bending moment.

$$\text{Shear force} : -EI\phi < x - 5 >^{-2} + F < x - 5 >^{0} - M_0 < x >^{-1} - R_0 < x >^{0} - R_8 < x - 8 >^{0} \tag{7}$$

$$\text{Bending moment} : -EI\phi < x - 5 >^{-1} + F < x - 5 >^{1} - M_0 < x >^{0} - R_0 < x >^{1} - R_8 < x - 8 >^{1} \tag{8}$$

The equation for the bending moment can be integrated and divided by EI to get the equation for the slope.

$$\text{Slope} : \phi < x - 5 >^{0} + \frac{-F < x - 5 >^{2} + M_0 < x >^{1} + R_0 < x >^{2} + R_8 < x - 8 >^{2}}{EI} \tag{9}$$

In the equation for the slope the hinge is represented as $\phi < x - 5 >^{0}$. This is a jump in the slope line of ϕ at the location $x = 5$, which represents the rotation of the hinge.

2.2 Singularity functions that return infinity

The fact that singularity functions of negative order can return infinity can lead to problems when they are used in mechanics. For a use in mechanics singularity functions of negative order do not have a physical meaning and it would be best if they would always return 0.

The singularity functions that are available in SymPy follow the mathematical definition of singularity functions, and can thus return infinity when they are of negative order. These functions should not be changed, as it is mathematically correct. A possible solution would be to create a new type of singularity function that can be used in mechanics. This singularity function would be the same as the original singularity function except for one thing, it would not return infinity. This new singularity function would always return 0 when of negative order.

These new singularity functions for a use in mechanics are not implemented in SymPy right now. Implementing them lies outside the scope of this project. While these new singularity functions are not available a temporary solution can be to remove singularity functions of negative order from some equations before substituting values. This temporary solution will make sure that there will be no infinities in the equations. This will make sure that calculated values, like reactions loads, are still correct.

3 Implementation of rotation and sliding hinges

The goal of this implementation is to give the Beam module the ability to calculate beams with multiple rotation and sliding hinges. This chapter will describe the changes made to reach this goal. First, the changes in the way the beams are calculated are discussed. Then, in the software design section, the necessary changes in the SymPy code will be explained. Finally, the results of the implementation are shown.

3.1 Mechanics calculations

To give the Beam module in SymPy the ability to calculate beams with multiple hinges, a change is needed in the way beams with hinges are calculated. Currently, as described in the introduction, there is the possibility in the Beam module to calculate beams with a single rotation hinge. In this section it will first be described how the beam module currently calculates beams with a hinge. After this, it will be described how this will be changed to be able to calculate beams with multiple hinges.

The change in the calculations will be shown with an example structure. This structure is shown in figure 13.

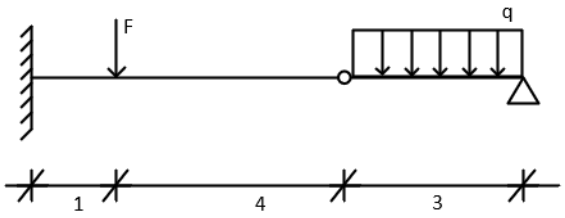


Figure 13: Structure with hinge.

3.1.1 Current calculations

The Beam module does not use singularity function to calculate beams with hinges in the current implementation. When calculating the reaction forces the module splits the beam into two parts: one part before and one part after the hinge. Both parts of the beam get their own load equation, which is initially 0. Then the loads in the original load equation are divided up over the load equations of the two parts of the beam, depending on their location. The shear force in the hinge is added as an unknown to both load equations. This results in two parts of the beam, both with their own load equation.

How this will look like for the example structure is shown in figure 14. The unknown reaction force in the hinge is called h.

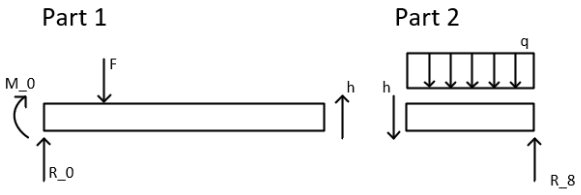


Figure 14: Beam divided into 2 parts.

For both parts of the beam the load equations are shown in equations 10 and 11.

$$\text{Load 1: } M_0 < x >^{-2} + R_0 < x >^{-1} - F < x-1 >^{-1} + h < x-5 >^{-1} \tag{10}$$

$$\text{Load 2: } -h < x-5 >^{-1} - q < x-5 >^{0} + q < x-8 >^{0} + R_8 < x-8 >^{-1} \tag{11}$$

The load equations of both parts of the beam are integrated to get equations for shear force, bending moment, slope and deflection. In these equations the values coming from the known boundary conditions are filled in. There are three boundary conditions at the location of the hinge. The first one is that the shear force is 0, note that this is between both forces of h. The second and third ones are bending moment is 0 and deflection left and right of the hinge is equal to one another.

Filling in all boundary conditions results in a set of equations containing the unknown reaction loads, shear force in the hinge and integration constants. This set of equations is solved using SymPy. The results are saved and substituted back into the load equation.

This method works and can be expended to be used for beams with multiple hinges. However this method is not the most efficient. Each hinge adds 3 unknowns, the unknown shear force and 2 integration constants, to the equations in this method. The next section will describe how beams with hinges can be calculated in a more efficient way.

3.1.2 New calculations

In chapter 2 of the document "Complete description of current extensions"(in Dutch) it is described how rotation and sliding hinges can be modelled as singularity functions of order -3 and -4 respectively. This chapter also gives some extra information on the hinges (TU Delft, 2024). With the method to model hinges described in the document, rotation and sliding hinges can be added to the load equation. This way beams do not have to be split up at the location of a hinge.

The load equation for the example structure using this method is shown in equation 12.

$$\begin{aligned}\text{Load: } M_0 < x >^{-2} + R_0 < x >^{-1} - F < x-1 >^{-1} \\ + EIP_5 < x-5 >^{-3} - q < x-5 >^{0} + q < x-8 >^{0} + R_8 < x-8 >^{-1}\end{aligned} \tag{12}$$

Note that this is the load equation the structure would have without the hinge, with only the addition of the term $EIP_5 < x-5 >^{-3}$. The symbol P_5 represents the rotation of the hinge, this can be seen as a jump in the slope line.

The load equation can be integrated and boundary conditions can be applied to generate equations with the unknowns. As there is only one extra unknown introduced by a hinge, there is only one extra boundary condition necessary. For rotation hinges this boundary condition is that the bending moment is equal to 0 at the location of the hinge. For sliding hinges the shear force is equal to 0 at the location of the hinge.

The equations that are created by applying the boundary conditions can be solved by SymPy. The result will be the values of the unknowns. These can be substituted back in the load equation.

The big upside of using this method is that the hinges can be added directly to the load equation. This makes it possible to calculate the beam as a whole. In this method each hinges adds only one extra unknown to the equations.

3.2 Software design

To implement the mechanics change discussed in the previous section, some new methods need to be added and some current ones changed. The most important changes will have to do with hinge placement and solving hinge beams.

3.2.1 Hinge placement

Currently adding a hinge is done by using the `join` method. This method adds two Beam objects to each other by using a fixed connection or a rotation hinge. The idea of adding these beams together this way is to give a user the possibility to create beams with different elastic modulus and second moment. Using this method only for hinge placement is not very efficient and it is not possible to add multiple hinges to a beam.

As it is the idea to be able to place multiple hinges in a beam, this way of placing hinges needs to change and become more efficient. This will be done by adding to methods: `apply_rotation_hinge` and `apply_sliding_hinge`. These methods will take the location where the hinge should be placed as input and will output the symbol of the hinge jump. For rotation hinges this jump will be a rotation jump and for sliding hinges this will be a deflection jump.

In figure 15 the difference between the current and new method to apply hinges is visualised. In this figure an example is shown for applying a rotation hinge in a beam.

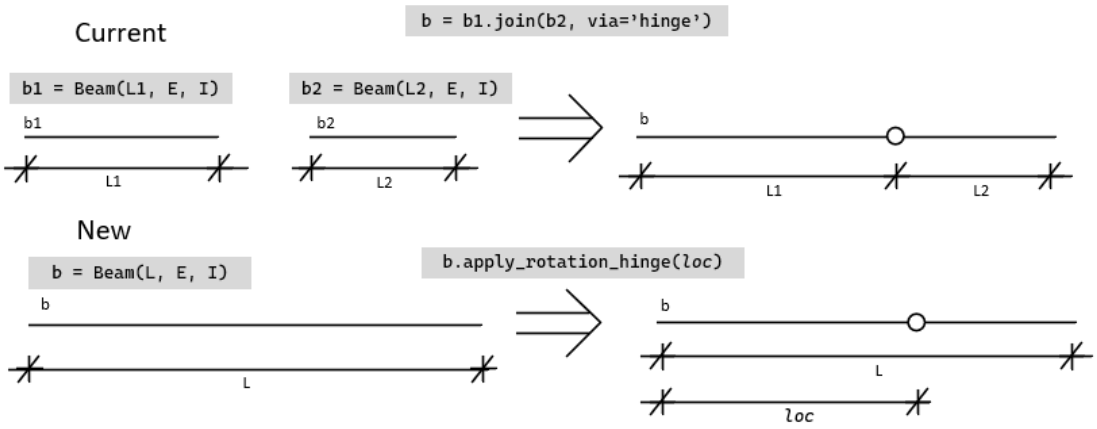


Figure 15: Beam divided into 2 parts.

It is visible that the new method to apply hinges in a structure requires a bit less code, 2 lines instead of 3. For a single hinge the difference is small, but it adds up for multiple hinges. In addition to being more efficient the new method is also more consistent with other methods to add something to the beam. Supports are added with the `apply_support` method and loads are added using `apply_load`. This consistency will help the user understand for what a method can be used.

The use of two separate methods to apply the different hinges has preference over the use of one method with a keyword argument. This keyword argument could specify which kind of hinge should be applied at a certain location. The use of two separate functions has the preference, because in this design the method would not return different things depending on the keyword argument. Sliding hinges are also a lot less common then rotation hinges. Making two separate methods makes sure it is clear that these hinges should not be seen as similar.

The `apply_rotation_hinge` method will replace the original `join` method to place rotation hinges in beams. To make sure old code does not break the `join` method will call the `apply_rotation_hinge` method when a user specifies that two beams should be joined by a hinge.

3.2.2 Solving hinge beams

The `solve_for_reaction_loads` method takes unknown reaction loads as inputs and solves them for the Beam object it is called upon. This method currently has a helper method `_solve_hinge_beams` for when Beam objects have a rotation hinge in them. This helper method is necessary because beams with hinges are currently solved in a different way than beams without hinges. The `_solve_hinge_beams` splits beams up, as described in chapter 3.1.1.

In the new design, the hinge placement methods, `apply_rotation_hinge` and `apply_sliding_hinge`, will add the correct functions for rotation and sliding hinges to the load equation. These placement methods will also add the boundary conditions that come with the hinge to the list of boundary conditions. The new way of modelling hinges in the load equation makes it possible to solve beams with hinges as a whole. This can be done by the `solve_for_reaction_loads` method. The helper method `_solve_hinge_beams` can be removed as it will no longer be used.

There will be some changes necessary in the `solve_for_reaction_loads` method. The method needs some extra code to be able to apply the boundary conditions that come with the hinges. The rest of the method can stay the same.

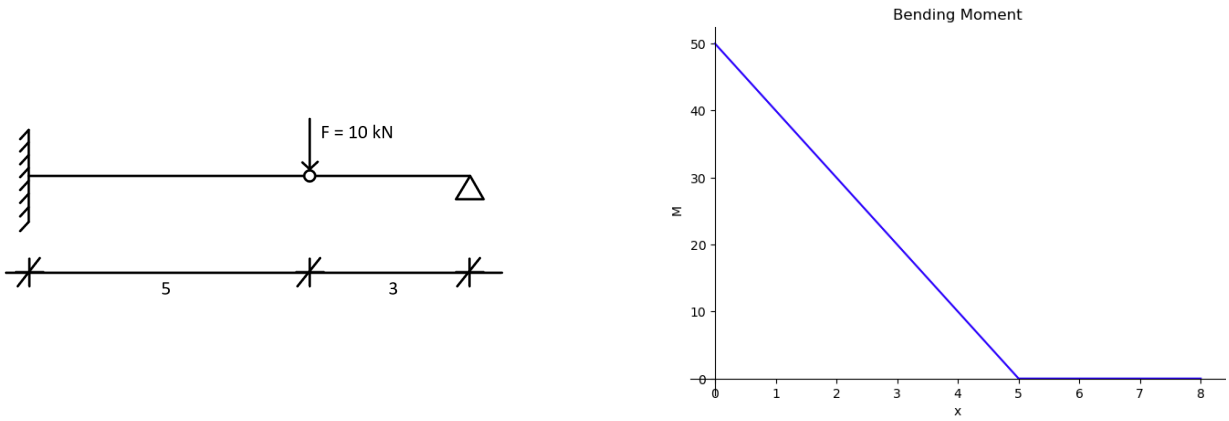
3.3 Results

The pull request containing all changes regarding the implementation of rotation and sliding hinges can be found here. This pull request was accepted by the SymPy community and is merged into the SymPy code. With this merged pull request the Beam module in SymPy gained the possibility to calculate beams with multiple rotation and sliding hinges. In addition to this the methods that calculate beams with hinges no more return incorrect solutions.

To display the added possibilities following from the pull request, three structures and their output will be shown. Two of these structures are also shown in the introduction.

First the structure from example 4 that used to give incorrect results will be calculated. The structure is shown again in figure 16. Using the new implementation of rotation hinges the bending moment line can be plotted again. It is visible that this time the correct bending moment line is returned by the methods.

The code for this example is visible here.



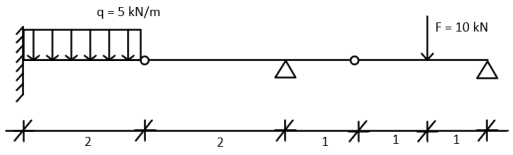
(a) Drawing of structure

(b) Plot of the bending moment line

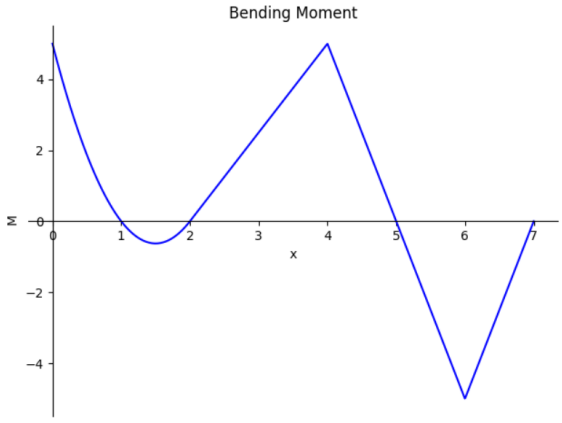
Figure 16: Example 7: Correct bending moment line returned by SymPy.

The most prominent reason to make this implementation was to give the code the ability to calculate beams with multiple hinges. In the introduction a structure containing multiple hinges is shown, this structure is shown again in figure 17. This structure could not be calculated with the old implementation of rotation hinges. Using the new implementation this is possible. In figure 17 the correct bending moment line that is returned by the SymPy methods is shown.

The code for this example is visible here.



(a) Drawing of structure

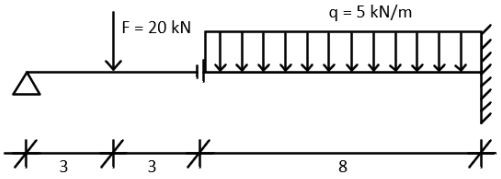


(b) Plot of the bending moment line

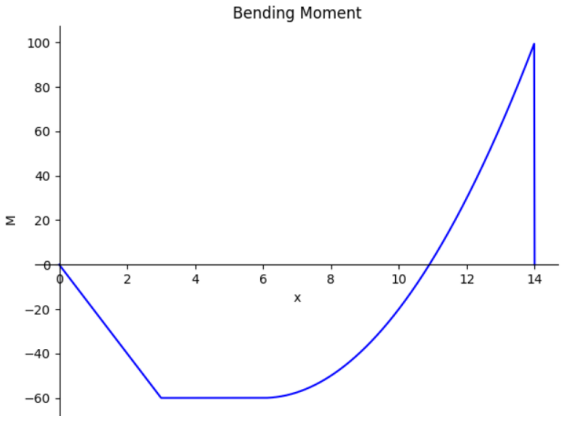
Figure 17: Example 8: Calculation of beam with multiple rotation hinges.

Next to rotation hinges, the pull request also added the possibility to calculate beams with sliding hinges. An example of the calculation of a beam with a sliding hinge is given in figure 18.

The code for this example is visible here.



(a) Drawing of structure



(b) Plot of the bending moment line

Figure 18: Example 9: Calculation of beam with sliding hinge.

The three examples given in this chapter could not be correctly calculated before the new implementation of rotation and sliding hinges. The results shown in this chapter are correct. These are calculated using the new implementation. This new implementation makes it possible to generate correct results for calculations on more complex beams using the Beam module in SymPy.

4 Implementation of Influence Line Diagram calculations

The goal of this implementation is to improve the calculations for influence line diagrams. The calculations should return correct results for all possible beams that can be made using the Beam module. This includes statically indeterminate beams and beams with hinges.

This chapter will start with a description of the changes made in the mechanics calculations. Then the software design that implements these changes will be given. Next, the considerations during the implementation phase will be discussed. At the end of this chapter the results of the implementation will be shown.

4.1 Mechanics calculations

In this section the changes in the mechanics calculations will be discussed. This will start with a description of problems with the current calculations. After this, the plan for the new calculations will be given.

4.1.1 Current calculations

Currently, the Beam module is only able to produce correct influence line diagrams for statically determinate beams. The reason for this has to do with the way the beams are calculated. Right now, the moving force over the beam is not added to the load equation. It is only used in equations for the overall vertical force equilibrium and the overall bending moment equilibrium. In the other equations, that come from applying boundary conditions, the moving force is not taken into account. This is the reason that the current calculations only produce correct output for statically determinate structures. For those structures the equations for vertical force equilibrium and bending moment equilibrium are enough. For more complicated structures where boundary conditions are used the current method gives incorrect output, because the moving force is not taken into account in these equations.

This problem will be shown using an example structure. This structure is shown in figure 19. Note that this structure is statically indeterminate.

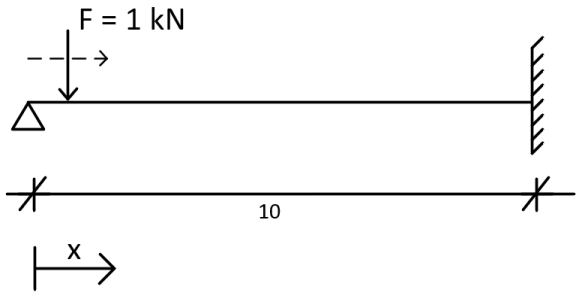


Figure 19: Structure with moving load.

For this example the problem will be shown for the boundary condition that the slope should be 0 at the location of the fixed support. First the load equation is shown in equation 13.

$$\text{Load: } R_0 < x >^{-1} + R_{10} < x - 10 >^{-1} + M_{10} < x - 10 >^{-2} \tag{13}$$

The moving load is not present in this equation, while it should be. Integrating the load equation three times and dividing by EI will give the slope equation.

$$\text{Slope: } \left(-\frac{R_0 < x >^2}{2} - \frac{R_{10} < x - 10 >^2}{2} - M_{10} < x - 10 >^1 + C_3\right) / EI \tag{14}$$

Filling in the boundary condition that the slope should be 0 at $x = 10$ gives:

$$-50R_0 + C_3 = 0 \tag{15}$$

This equation will lead to incorrect reaction loads. The moving force should be present in this equation, because it has an effect on the slope at the fixed support. Currently it is not applied in the load equation and is thus missing in this equation.

The current calculations are also unable to calculate beams with hinges. There are no checks for hinges and the the correct equations using the boundary conditions are not set up.

4.1.2 New calculations

In the new calculations there will be 2 big changes to the way the influence line diagrams are calculated. The first big change will have to do with the addition of the moving load to the load equation. The second change will have to do with the ability to calculate influence line diagrams for beams with hinges.

The moving load needs to be added to the load equation. This load equation is integrated and boundary conditions are applied to create equations. The moving load needs to be present in these equations in order to get the correct results. In chapter 3 of the document "Complete description of current extensions"(in Dutch) it is described how a moving load can be added to the load equation. This can be done by specifying the location of the load as a variable a in a singularity function. A moving point load of magnitude 1 will be added to the load equation as $< x - a >^{-1}$ (TU Delft, 2024). Using this method will also change the variable in the output of the influence line calculations. The location of the moving force will then be described by a instead of x.

This solves the problem that is currently present. The new load and slope equations for the structure in figure 19 will be shown.

$$\text{Load: } R_0 < x >^{-1} + R_{10} < x - 10 >^{-1} + M_{10} < x - 10 >^{-2} - < x - a >^{-1} \tag{16}$$

$$\text{Slope: } \left(-\frac{R_0 < x >^2}{2} - \frac{R_{10} < x - 10 >^2}{2} - M_{10} < x - 10 >^1 + \frac{< x - a >^2}{2} + C_3\right) / EI \tag{17}$$

The moving force is now present in the equations. Again, the boundary condition for the slope at the fixed support can be applied. Filling in $x = 10$ and setting it equal to 0 gives:

$$-50R_0 + \frac{< 10 - a >^2}{2} + C_3 = 0 \tag{18}$$

The moving force is now present in the equation. This equation will result in the correct equations for the reaction loads.

The current methods are unable to calculate influence line diagrams for beams with hinges. In the new calculations this will be fixed by adding checks for hinges. If there are hinges in the beam, the correct boundary conditions will be applied. For each hinge this will result in an extra equation, in order to solve the unknown introduced by the hinge. This same method is used to calculate the reaction loads for beams with hinges.

4.1.3 Results when moving force is not on the beam

When the influence line diagram calculations are performed on a beam, the equations for the reaction loads in terms of the location of the moving force can be shown. An example of how such equations for the reaction loads can look in the current implementation is shown in equation 19.

These are the equations for the reaction loads of the structure in figure 9. The code for these equations is visible here.

$$R_0 : 1 - \frac{x}{10}, \quad R_{10} : \frac{x}{10} \tag{19}$$

It is visible that there are no singularity functions present in these equations. This means that the equations will return non-zero values when the location of the force x is not on the beam. Take for example $x = -10$, this is not on the beam, but the equation for the reaction force R_0 will return 2. This should not be the case as there is no force on the beam.

In other methods in the Beam module results are only generated for values that on the beam. Next to the beam the result of the equations is 0. For consistency this should also be the case for the results of influence line calculations.

By using singularity functions in the calculations for the influence line equations it can be realised that they return 0 when the moving force is not on the beam. These singularity functions can make sure that the moving force is not taken into account in the calculations when it is not on the beam. If the calculations are made this way and the force is not on the beam the result will always be 0. With this method the equations for the reaction loads will also include singularity functions. The influence line equations for the reactions loads that result from using this method are shown in equation 20.

The code for these equations can be found here.

$$R_0 :< a >^0 - \frac{< a >^1}{10} + \frac{< a - 10 >^1}{10}, \quad R_{10} : \frac{< a >^1}{10} - < a - 10 >^0 - \frac{< a - 10 >^1}{10} \tag{20}$$

It is visible that these equations only include singularity functions. This makes it possible to always return 0 when the moving force is not on the beam. In this example if we fill $a = -10$, both the equations for R_0 and R_{10} will return 0.

This way of using singularity functions in the calculations for influence line diagrams will be implemented in the new calculations. This will make sure that the equations that result from these calculations will return 0 when the moving force is not on the beam.

4.2 Software design

There are no new methods needed to implement the changes in mechanics calculations. There do need to be some changes to the current methods. The first change has to do with adding the moving load to the load equation. Another change has to do with calculations on beams with hinges.

The load equation for influence line calculations is set up in a helper method `_solve_for_ild_equations`. This methods sets up the load equation and integrates it to create the necessary equations for the beam properties. The method is used by `solve_for_ild_reactions` to get the equations. Right now the `_solve_for_ild_equations` has no input, but to be able to apply the moving load to the load equation the method has to know the magnitude of the moving load. This is done by adding a input parameter: value. The `solve_for_ild_reactions` already has this value as input parameter and this value will be inputted into `_solve_for_ild_equations`, to make sure the correct magnitude of the moving load is applied in the load equation.

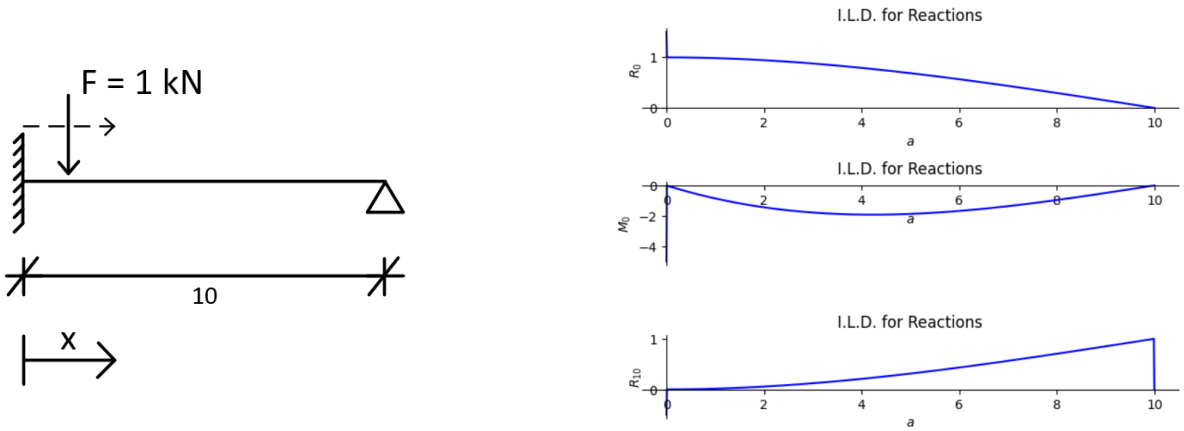
Another change has that needs to be applied is the addition of code to check for and handle hinges in the beam. This code will work in the same way as the code that handles hinges in the `solve_for_reaction_loads` method described in chapter 3.2.2. If there are hinges in the beam, the correct boundary conditions will be applied to create equations. These equations will then be solved together with the other equations generated in the `solve_for_ild_reactions` method, resulting in the correct output for beams with and without hinges.

4.3 Results

The pull request containing all changes regarding the implementation of influence line diagram calculations can be found here. This pull request was accepted by the SymPy community and is merged into the SymPy code. With this merged pull request the Beam module in SymPy gained the ability to correctly calculate influence line diagrams for beams with and without hinges. This will be shown with two examples.

The first structure for which the results will be shown is the statically indeterminate structure from the introduction for which incorrect results were returned. The structure is shown again in figure 20. In the same figure the influence lines for the reaction loads are plotted. This time, these influence lines are correct.

The code for this example is visible here.



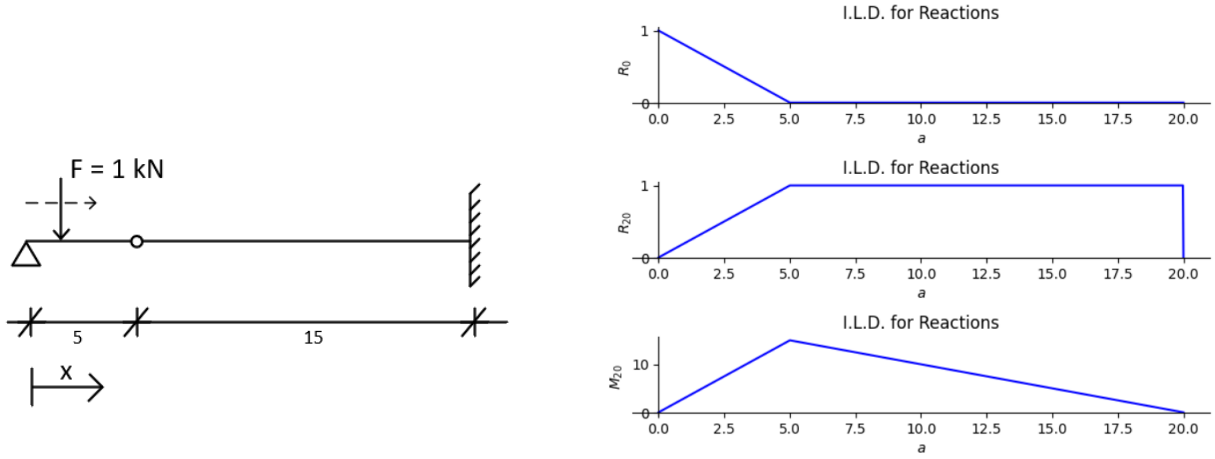
(a) Drawing of structure

(b) Plot of the bending moment line

Figure 20: Example 10: Influence line calculations for statically indeterminate structure.

The second structure for which the results will be shown is a structure with a rotation hinge. Before this implementation it was not possible to calculate correct influence line diagrams for beams with hinges. In figure 21 a structure with a hinge is shown. For this structure the influence lines for the reaction forces are shown.

The code for this example is visible here.



(a) Drawing of structure

(b) Plot of the influence lines for the reaction loads

Figure 21: Example 11: Influence line calculations for beam with a hinge.

5 Discussion and Recommendation

During the implementations described in chapter 3 and 4 certain issues within SymPy came to light. Some of these issues could not directly be solved during the implementation. These issues will be discussed in this chapter.

In addition to reflecting on the challenges during the project, there will be a look forward. Not all subjects described in the introduction were implemented during the project. It will be discussed how these subjects could be best implemented, using the current knowledge of the Beam module in SymPy.

5.1 Unresolved issues

During the implementation of rotation and sliding hinges and the implementation of calculations of influence line diagrams certain problems in the code came to light. Some of these problems were not fixed by the implementation. These problems were already present before the implementation and were outside the scope of this project. In this chapter these problems will be discussed. These problems are also described on GitHub in issues, so future developers can work on them. Links to these issues on GitHub are added in each problem description.

5.1.1 Error reporting when solving reaction loads

The method `solve_for_reaction_loads` is used to calculate the values of the reaction loads for a certain structure. The method sets up equations and then calls the `linsolve` method of SymPy to solve for the unknowns.

When the `linsolve` method cannot solve the equations it will return an error. This error will be the same regardless of the problem. This means that it is not possible to trace back the problem from the error message, making it difficult to find the problem.

It would be better if the `solve_for_reaction_loads` method contains some checks for often occurring problems, for example typing errors or forgetting to input reactions. This way the error can specify the problem, making it easier for the user to solve it.

The issue on GitHub about this problem can be found here.

5.1.2 Joining beams with different elastic modulus

The Beam module in SymPy has a `join` method. This method is used to create beams with discontinuous elastic modulus (E) and second moment (I), by joining two different beams. It is currently not possible to create a beam with a discontinuous E. This is because the `join` method cannot handle inputs of beams with different E. Joining two beams with different I is no problem. To solve the problem the input of different E should be handled in the same way as inputs of beams with different I.

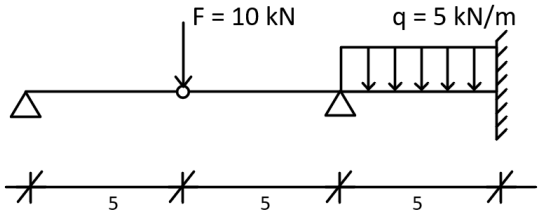
The issue on GitHub about this problem can be found here.

5.1.3 Calculation of contraflexure points in beams

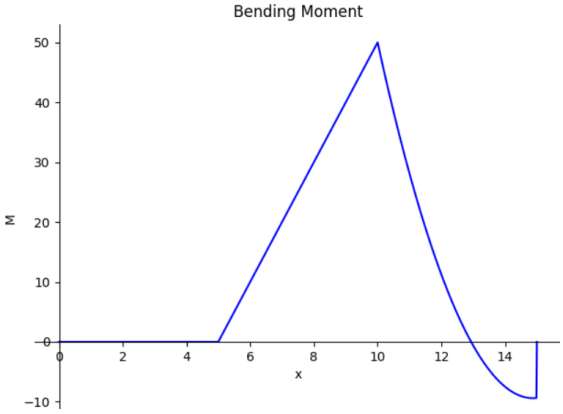
In the Beam module the `point_cflexure` method is used to find points of contraflexure. These are points where the bending moment is zero and changes from negative to positive or the other way around. This can be seen in the bending moment line by the crossing of the horizontal axis.

The `point_cflexure` method works for most beams. The problem with the method is that it cannot calculate points of contraflexure for beams where a whole region of the bending moment line is equal to 0. An example of this is given in figure 22.

The code for this example is visible here.



(a) Drawing of structure



(b) Plot of the bending moment line

Figure 22: Example 12: Bending moment line that will result in error.

For this bending moment line the `point_cflexure` method will not be able to calculate the points of contraflexure. It is visible that there is one point of contraflexure somewhere around $x = 13$, but the method will return an error. It sets the bending moment equation equal to zero and solves for x. In this case the result will be a whole region of x. It is not yet implemented how the method should handle this.

The issue on GitHub about this problem can be found here.

5.1.4 Specifying whether locations are just before or after each other

In numerical examples this can be done by specifying the location of something as the location of something else added with a very small number. For example $loc^+ = loc + 0.001$. This way of defining if locations are just before or after each other is not yet possible in symbolic notation.

This results in problems in the implementation of influence line diagrams. In this implementation singularity functions are used that make sure the results are always zero when the moving force is not on the beam. The problems occur exactly at the start and end of the beam. At these places the singularity functions can have a jump in the value they return. This can lead to problems when the moving load is exactly at those locations.

The problem is shown with an example.

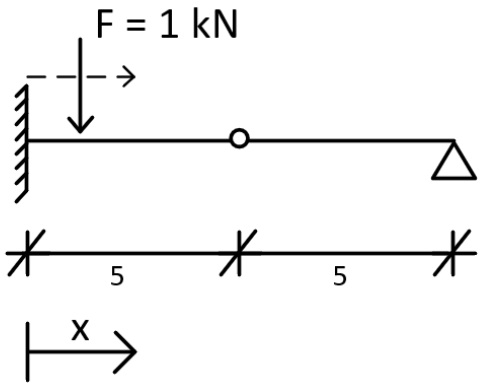


Figure 23: Structure to show influence line calculations.

Take the equation for the reaction load at the pin support in figure 23. This equation for the reaction load is given in equation 21.

$$R_{10} = \frac{< a - 5 >^1}{5} - < a - 10 >^0 - \frac{< a - 10 >^1}{5} \tag{21}$$

The reaction load is 0 if a is not on the beam and the correct value if a is on the beam. The problem occurs when a is exactly at the end of the beam. When $a = 10$ is filled in in the equation for R_{10} the output is 0, while it should be 1 because the force is directly placed on the pin support. It can be shown that this problem only occurs when a is exactly 10. Filling in $a = 9.9999$ gives $R_0 = 0.99998$.

This problem occurs because the singularity function that cuts of the moving load at the end of the beam is specified to do this at $a = l$ with l the length of the beam. In this example $l = 10$. When the moving load is exactly at $a = 10$ the singularity function evaluates it as outside of the beam.

This problem can be solved by being able to specify if a location is just before or just after another location. Using this it could be specified that the singularity function to cut of the beam should work just after $a = l$.

This specifying if locations are just before or after each other can also be useful in other mechanical problems. For example when trying to place a moment load just before a rotation hinge.

A possible solution would be to specify the locations using an extra symbol which can be added to the location. This symbol can then go to zero in it's limit. While testing this solution a bug in the implementation of singularity functions came up. There is a proposed solution for this bug which should get the original solution to work. This is yet to be tested. The bug in the singularity function and the proposed solution for this are outside the scope of this project.

The issue on GitHub about this problem can be found here.

5.1.5 Singularity functions that return infinity

The definition of singularity functions of negative order, $< x - a >^n$ with $n < 0$, states that the function can be evaluated as 0 for all x, except when $x = a$, then it is evaluated as infinity. Singularity functions that are evaluated as infinity can cause problems in mechanics calculations. In mechanics calculation this singular point of infinity has no meaning and should actually be evaluated as 0. An example of this is given in chapter 1.2.2.

The temporary solution discussed in chapter 2.2 removes all negative order singularity functions from equations before evaluating them. This solution works, but it would be better if this would not be necessary.

The solution would be to define an extra type of singularity function for a use in mechanics calculations. In a mechanical application a singularity function of negative order has no meaning. It only gets meaning when integrated to a non-negative order singularity function. It would thus be better if the singularity functions used in mechanics would always be evaluated as 0 when they are of negative order. This would make the step of manually removing them from calculations obsolete.

There is still some research required on the best way to implement this. Also, the SymPy community would have to agree that this change is wanted, as the mechanics application is only a small part of SymPy.

The issue on GitHub about this problem can be found here.

5.1.6 Simplification of equations with singularity functions

The equations that result from the influence line calculations almost only exist out of singularity functions. This is not a problem in general, but for some more complicated structures these equations can have some unusual notation.

SymPy is not able to simplify equations with singularity functions. This becomes apparent when looking at some reaction loads for certain structures. In equation 22 an example of such a reaction load is shown.

$$R_0 = -\frac{< -a >^3}{500} + \frac{< 5 - a >^3}{250} - \frac{< 10 - a >^3}{500} + \frac{3(< -a >^0)^3}{2} + \frac{3(< -a >^0)^2 < -a >^1}{10} + \frac{< a >^0}{2} - \frac{< a >^1}{20} + \frac{< a - 10 >^1}{20} \quad (22)$$

This reaction returns the correct values but does not have the correct notation. The singularity functions that are squared or to the power of 3 can be simplified. As well as the singularity functions that are multiplied with each other. For example $3(< -a >^0)^2 < -a >^1$ can be rewritten to $3 < -a >^1$.

The fact that SymPy is not able to simplify these equations with singularity functions has no effect on the correctness of the final result, but it can produce results with unusual notation. It would be better if SymPy would simplify these equations, this would make equations shorter and easier to understand.

These simplifications are not necessary when solving these problems by hand or by using the mathematical functions of SymPy. In these calculations it is not possible to make sure the results are zero when the moving force is exactly not on the beam. It is common to make sure results are 0 before $x = -1$ and after $x = l + 1$, with l the length of the beam. These calculations are easier and the equations that result from them do not have to be simplified. The downside of these calculations is that there are also non-zero results when the moving force is not on the beam.

The issue on GitHub about this problem can be found here.

5.2 Future implementations

The research done by TU Delft students about the application of the method of Macaulay focused not only on hinges and influence lines. Also other subjects were researched. These subjects are:

- Normal forces
- Spring supports and spring connections
- Inclined and curved beams
- 2D structures

There is no current implementation of these subjects in the Beam module in SymPy. In this section it will be discussed how a start can be made with implementing these subjects. It must be noted that these possibilities to start are not actually tested and follow from the current knowledge of the Beam module. While implementing these subjects it can be possible that there are other ways to do it that might work better.

5.2.1 Normal forces

Normal forces are axial forces in a beam. Right now it is not possible to add these forces to a beam in the Beam module. There is also a Beam3D module available in SymPy, in which it is possible to add normal forces. This module is focused on calculations of cross-sectional properties in beams and not on the calculation of structures. In the Beam3D module normal forces can be added by specifying a direction when applying a load, by the use of a keyword argument.

This specifying of a direction can be added to the Beam module. Right now the `apply_load` method always applies the load in the vertical direction. A possible change would be to add a keyword argument, similar to the one used in Beam3D, to `apply_load` in which the direction of the load can be specified.

The reaction loads in axial direction can be solved by integrating the axial load equation and setting the correct boundary conditions.

5.2.2 Spring supports and spring connections

The implementation of spring supports and spring connections can possibly be done without large changes to the existing code. These spring supports and spring connections can be modeled the same way as normal supports and normal connections. The only difference will be the boundary condition that comes with the support or connection.

Take for example a fixed support and a pin support with a rotation spring. For both of these support the boundary condition holds that the deflection should be 0 at the location of the support. For the fixed support it also holds that the slope should be 0 at the location of the support. For the pin support with a rotation spring the boundary condition will be that the bending moment at the location of the support should be equal to the slope times the spring constant.

More information on the modeling and boundary condition of spring supports and spring connections can be found in chapter 2 of the document "Complete description of current extensions"(in Dutch) (TU Delft, 2024).

5.2.3 Inclined and curved beams and 2D structures

The implementation of inclined and curved beams and 2D structures will probably be a lot more difficult than the other implementations discussed in this chapter. It would be best if these beams and 2D structures are implemented in a new module instead of integration into the existing beam module. There are two main reasons for this.

The first reason has to do with the specification of locations in the different modules. In the current Beam module each place on the beam has only one value that specifies it's location, the x coordinate. In the implementation of 2D beams (inclined and curved) and 2D structures places on the beam or structure would need two values to specify their location. This will result in a different way of defining locations and is very difficult to implement in the current Beam module. If this would be implemented in the current beam module nearly every calculation would have to be changed to be able to handle this.

The second reason to implement curved beams and 2D structures in a new module is to keep the current Beam module simple and clear. Right now it is relatively easy to define a structure in the Beam module and calculations work as they should. Nearly every possible beam can be calculated. If users want to calculate a single beam it is nice if they can easily define it using the current Beam module. If users want to do more complicated calculations of 2D structures they can use the new module, that is specialized in those types of calculations.

The creation of this new Beam module will be a large task that should be done with much precision and testing. It is expected that new problems will rise up during implementation, which are not known right now. For this implementation a lot of extra research should be done on the best way of defining and calculating 2D structures using SymPy.

6 Conclusion

This report addressed the following research question: *How can the current implementation of Macaulay's method be extended in Python to be able to calculate and analyze more complicated beams and structures, based on the advancement made by previous Bachelor End Projects at the TU Delft?*

The advancements made by previous Bachelor End Project can be categorised into different subjects. It has become apparent during this project that the implementation of Macaulay's method can be extended in Python by implementing these subjects one by one. These subjects can be implemented into the Beam module in SymPy. This will give the module the ability to calculate and analyse more complicated beams and structures.

For the implementations that are described in this report a structured approach is used. In this approach the first step is analysing the current mechanics calculations that are performed. If the use of the method of Macaulay can improve these mechanics calculations they can be changed. The advancements made during previous Bachelor End Projects can be used to improve the mechanics calculations.

The second step in this approach is changing the software. Certain changes in the code will be necessary for the software to be able to perform the new mechanics calculations.

Using this approach the implementation of rotation and sliding hinges and the implementation of influence line diagram calculations have been completed. Following from these results it can be concluded that the described approach works.

This method to extend the implementation of the method of Macaulay in Python can be used to implement more subjects following from the advancements made by previous Bachelor End Projects. It is expected that for some future improvements the implementations will be in a new Beam module in SymPy. This will probably be the case for the calculations of 2D structures. The method to extend the implementation of Macaulay's method in Python, described in this report, is only used for implementations in the current Beam module. It cannot be concluded that this method would also work effectively for extending implementations in other modules.

References

Baudoin, A. (2024). *Continue macaulay methode voor geknikte, vertakte en gesloten constructies.*

Jankie, J. (2023). *Maucalay's methode toegepast met invloedslijnen.*

Qadriyeh, E. (2023). *Analyse van schuine en kromme liggers met de macaulay methode: Een diepgaande studie.*

TU Delft, . (2024). *Complete description of current extensions.* Retrieved from `https://teachbooks.github.io/Macaulays_method/current_extensions.html` (Accessed: 2024-06-14)

Van der Wulp, J. (2023). *De methode van macaulay voor tweedimensionale constructies.*