Extension to Macaulay's method @ TU Delft

van Woudenberg, T.R.; van der Wulp, J.D.; Jankie, J.A.C.; Qadriyeh, E.; Baudoin, A.J.D.; van Gelder, M.J.; Saheli, B.

DOI
10.5281/zenodo.15099635

Publication date
2025

Document Version
Proof

Citation (APA)
van Woudenberg, T. R. (Ed.), van der Wulp, J. D., Jankie, J. A. C., Qadriyeh, E., Baudoin, A. J. D., van Gelder, M. J., & Saheli, B. (2025). *Extension to Macaulay's method @ TU Delft*. (v1.0.1 ed.) Delft University of Technology. https://doi.org/10.5281/zenodo.15099635

Macaulays methode

Contents

Macaulays methode is een methode waarmee de verplaatsingsfunctie van een ligger van meerdere segmenten kan worden bepaald in één functie (Macaulay, 1919). Daarbij is het niet nodig om een balk onder te verdelen in segmenten met randvoorwaarden (afhankelijk van verdeelde belastingen, puntlasten, opleggingen, etc.) en blijft de notatie direct herkenbaar bij integratie/differentiatie (in tegenstelling tot de Dirac delta / Heaviside functies). Er bestaat een implementatie voor deze methode voor balken, maar een uitbreiding naar 2d-constructies zoals portalen ontbreekt.

Voorbeeld Macaulays methode zonder uitbreidingen

Laten we de volgende constructie bekijken:

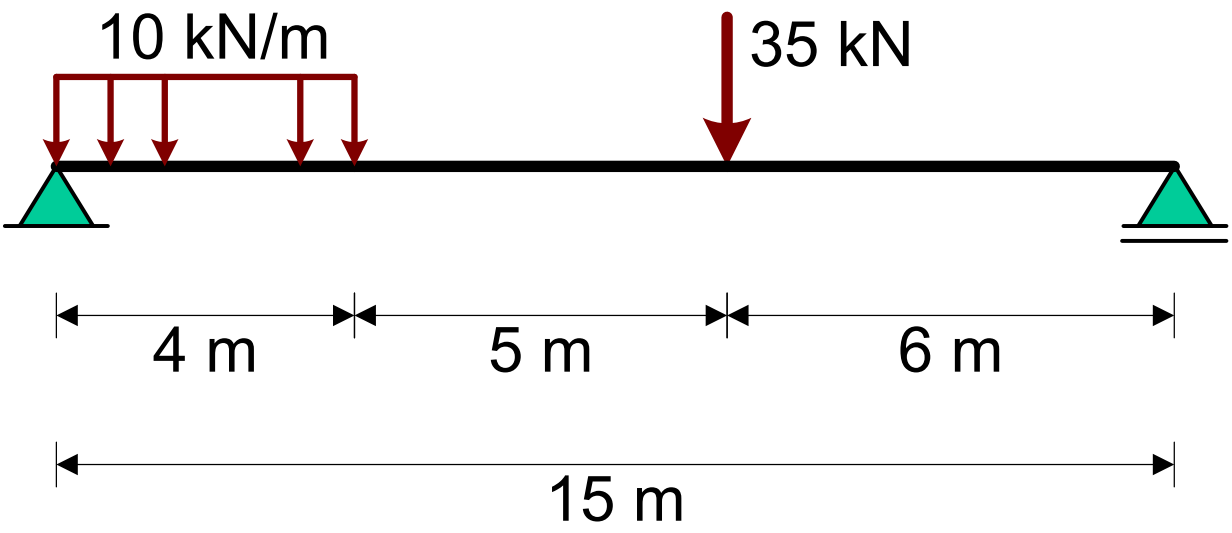


Fig. 1 Voorbeeld constructie

Definieer belasting

Deze belasting (inclusief oplegreacties) kan als volgt worden beschreven volgens Macaulay:

$$q\left(x\right) = \underbrace{A_v\langle x\rangle^{-1} + B_v\langle x-15\rangle^{-1}}_{\text{oplegreacties}} \underbrace{+10\langle x\rangle^{0} - 10\langle x-4\rangle^{0}}_{\text{verdeelde belasting}} \underbrace{+35\langle x-9\rangle^{-1}}_{\text{puntlast}}$$

Dit kan in sympy worden genoteerd als:

```
import sympy as sym
sym.init_printing()
```

```
x = sym.symbols('x')
EI = sym.symbols('EI')
C_1, C_2, C_3, C_4 = sym.symbols('C_1 C_2 C_3 C_4')
A_v, B_v = sym.symbols('A_v B_v')

Q = 10
L = 15
a = 4
b = 5
F = 35

q =  A_v * sym.SingularityFunction(x,0,-1) + Q * sym.SingularityFunction(x,0,0) - Q * sym
display(q)
```

$$A_v\langle x\rangle^{-1} + B_v\langle x-15\rangle^{-1} + 10\langle x\rangle^{0} - 10\langle x-4\rangle^{0} + 35\langle x-9\rangle^{-1}$$

Definieer randvoorwaarden

Voor dit probleem gelden de volgende randvoorwaarden:

$$\begin{aligned} w\left(0\right) &= 0 \\ w\left(15\right) &= 0 \\ M\left(0\right) &= 0 \\ M\left(15\right) &= 0 \\ V\left(0^{-}\right) &= 0 \\ V\left(15^{+}\right) &= 0 \end{aligned}$$

Dit kan in sympy als volgt worden gedefinieerd:

```
V = -sym.integrate(q, x) + C_1
M = sym.integrate(V, x) + C_2
kappa = M / EI
phi = sym.integrate(kappa, x) + C_3
w = -sym.integrate(phi, x) + C_4
display(w)
```

$$-C_3 x + C_4 - \frac{-\frac{A_v \langle x\rangle^3}{6} - \frac{B_v \langle x-15\rangle^3}{6} + \frac{C_1 x^3}{6} + \frac{C_2 x^2}{2} - \frac{5\langle x\rangle^4}{12} + \frac{5\langle x-4\rangle^4}{12} - \frac{35\langle x-9\rangle^3}{6}}{EI}$$

De dwarskracht net links van 0 en net rechts van 15 wordt gemodelleerd met een 'dummy'-variabele van $\frac{1}{\infty}$.

```
oo = sym.Dummy('oo', prime=True)
very_small = 1/oo
```

Nu kunnen in SymPy de vergelijkingen voor de randvoorwaarden worden opgesteld:

```
Eq1 = sym.Eq(w.subs(x,0),0)
Eq2 = sym.Eq(w.subs(x,L),0)
Eq3 = sym.Eq(M.subs(x,0),0)
Eq4 = sym.Eq(M.subs(x,L),0)
Eq5 = sym.Eq(V.subs(x,0-very_small),0).subs(oo,sym.oo)
Eq6 = sym.Eq(V.subs(x,L+very_small),0).subs(oo,sym.oo)
display(Eq1, Eq2, Eq3, Eq4, Eq5, Eq6)
```

$$C_4 = 0 \quad -15C_3 + C_4 - \frac{-\frac{1125A_v}{2} + \frac{1125C_1}{2} + \frac{225C_2}{2} - \frac{48760}{3}}{EI} = 0 \quad C_2 = 0$$
$$-15A_v + 15C_1 + C_2 - 730 = 0 \quad C_1 = 0 \quad -A_v - B_v + C_1 - 75 = 0$$

Los op voor randvoorwaarden

Oplossen naar de randvoorwaarden geeft:

$$w(x) = \frac{26692x \underbrace{-292\langle x\rangle^3 - 158\langle x-15\rangle^3}_{\text{oplegreacties}} \underbrace{+15\langle x\rangle^4 - 15\langle x-4\rangle^4}_{\text{verdeelde belasting}} \underbrace{+210\langle x-9\rangle^3}_{\text{puntlast}}}{36EI}$$

welke oplossing ook met code kan worden gevonden:

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6),(C_1,C_2,C_3,C_4,A_v,B_v))
display(sol)

display(w.subs(sol).factor(EI))
```

$$\left\{A_v : -\frac{146}{3},\ B_v : -\frac{79}{3},\ C_1 : 0,\ C_2 : 0,\ C_3 : -\frac{6673}{9EI},\ C_4 : 0\right\}$$

$$\frac{26692x - 292\langle x\rangle^3 + 15\langle x\rangle^4 - 15\langle x-4\rangle^4 + 210\langle x-9\rangle^3 - 158\langle x-15\rangle^3}{36EI}$$

Dit kan ook worden omgeschreven naar een functie per domein:

$$\begin{cases} \frac{79x^3-3555x^2+40781x-78465}{18EI} & \text{for } x > 9 \\ \frac{-13x^3-360x^2+7633x-960}{9EI} & \text{for } x > 4 \\ \frac{x\left(15x^3-292x^2+26692\right)}{36EI} & \text{for } x > 0 \end{cases}$$

wat met de volgende code kan worden gevonden:

```
display(sym.simplify(w.subs(sol).rewrite(sym.Piecewise)))
```

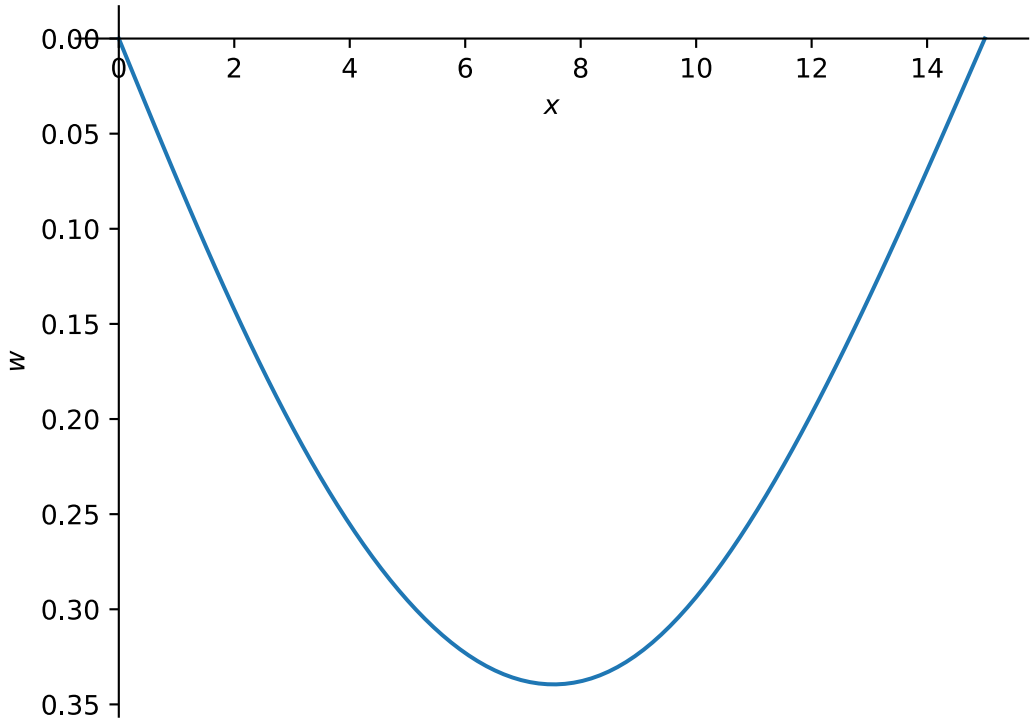
$$\begin{cases} \frac{6272\cdot(15-x)}{9EI} & \text{for } x > 15 \\ \frac{79x^3-3555x^2+40781x-78465}{18EI} & \text{for } x > 9 \\ \frac{-13x^3-360x^2+7633x-960}{9EI} & \text{for } x > 4 \\ \frac{x\left(15x^3-292x^2+26692\right)}{36EI} & \text{for } x > 0 \\ \frac{6673x}{9EI} & \text{otherwise} \end{cases}$$

Dit resultaat kan ook worden geplot voor een EI van 10000:

```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

```
w_numpy = sym.lambdify(x, w.subs(sol).subs(EI,10000).rewrite(sym.Piecewise))
```

```
x_numpy = np.linspace(0,15,100)
plt.figure()
plt.plot(x_numpy,w_numpy(x_numpy))
plt.xlabel('$x$')
plt.ylabel('$w$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```



The dwarskrachtverdeling kan ook worden gevonden:

$$V\left(x\right) = \underbrace{\frac{146\langle x\rangle^{0}}{3} + \frac{79\langle x-15\rangle^{0}}{3}}_{\text{oplegreacties}} \underbrace{-10\langle x\rangle^{1} + 10\langle x-4\rangle^{1}}_{\text{verdeelde belasting}} \underbrace{-35\langle x-9\rangle^{0}}_{\text{puntlast}}$$

met de volgende code:

```
display(V.subs(sol))
```

$$\frac{146\langle x\rangle^{0}}{3} - 10\langle x\rangle^{1} + 10\langle x-4\rangle^{1} - 35\langle x-9\rangle^{0} + \frac{79\langle x-15\rangle^{0}}{3}$$

En ook deze formule kan worden omgeschreven naar een formule per domein:

$$\begin{cases} -\frac{79}{3} & \text{for } x > 9 \\ \frac{26}{3} & \text{for } x > 4 \\ \frac{146}{3} - 10x & \text{for } x > 0 \end{cases}$$

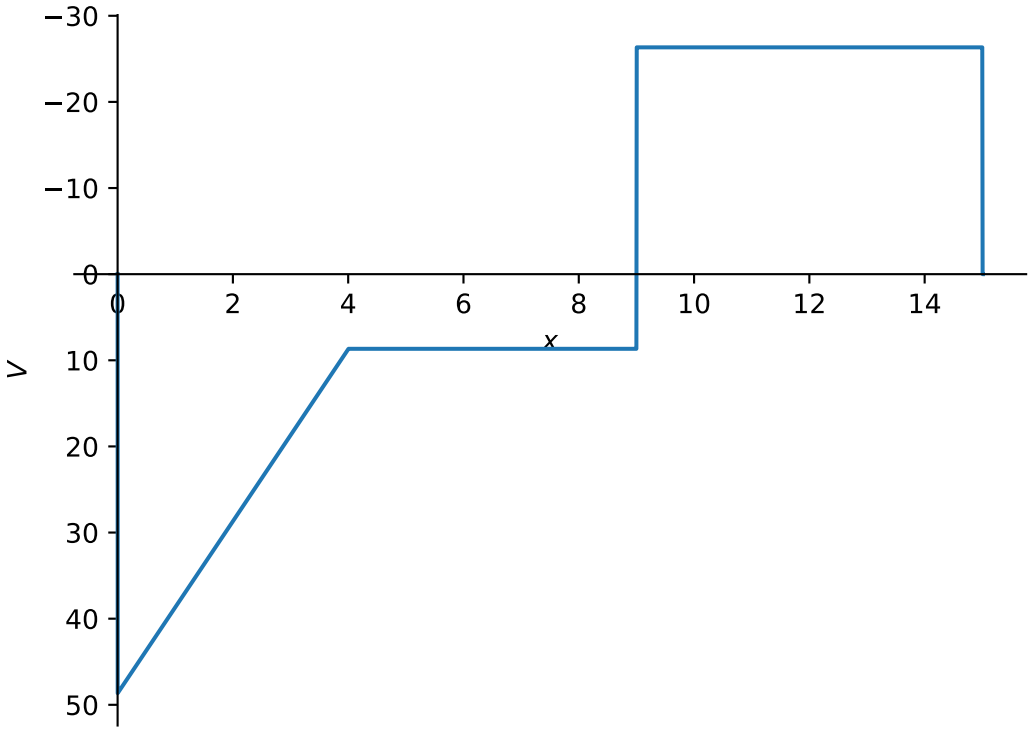
```
display(sym.simplify(V.subs(sol).rewrite(sym.Piecewise)))
```

$$\begin{cases} 0 & \text{for } x > 15 \\ -\frac{79}{3} & \text{for } x > 9 \\ \frac{26}{3} & \text{for } x > 4 \\ \frac{146}{3} - 10x & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

en kan ook worden geplot:

```
V_numpy = sym.lambdify(x, V.subs(sol).rewrite(sym.Piecewise))
```

```
x_numpy = np.linspace(0,15.01,10000)
plt.figure()
plt.plot(x_numpy,V_numpy(x_numpy))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```



Voorbeeld Macaulays methode in SymPy

De methode van Macaulay is ook geïmplementeerd in SymPy en kan daar worden toegepast zonder vergelijkingen op te moeten stellen.

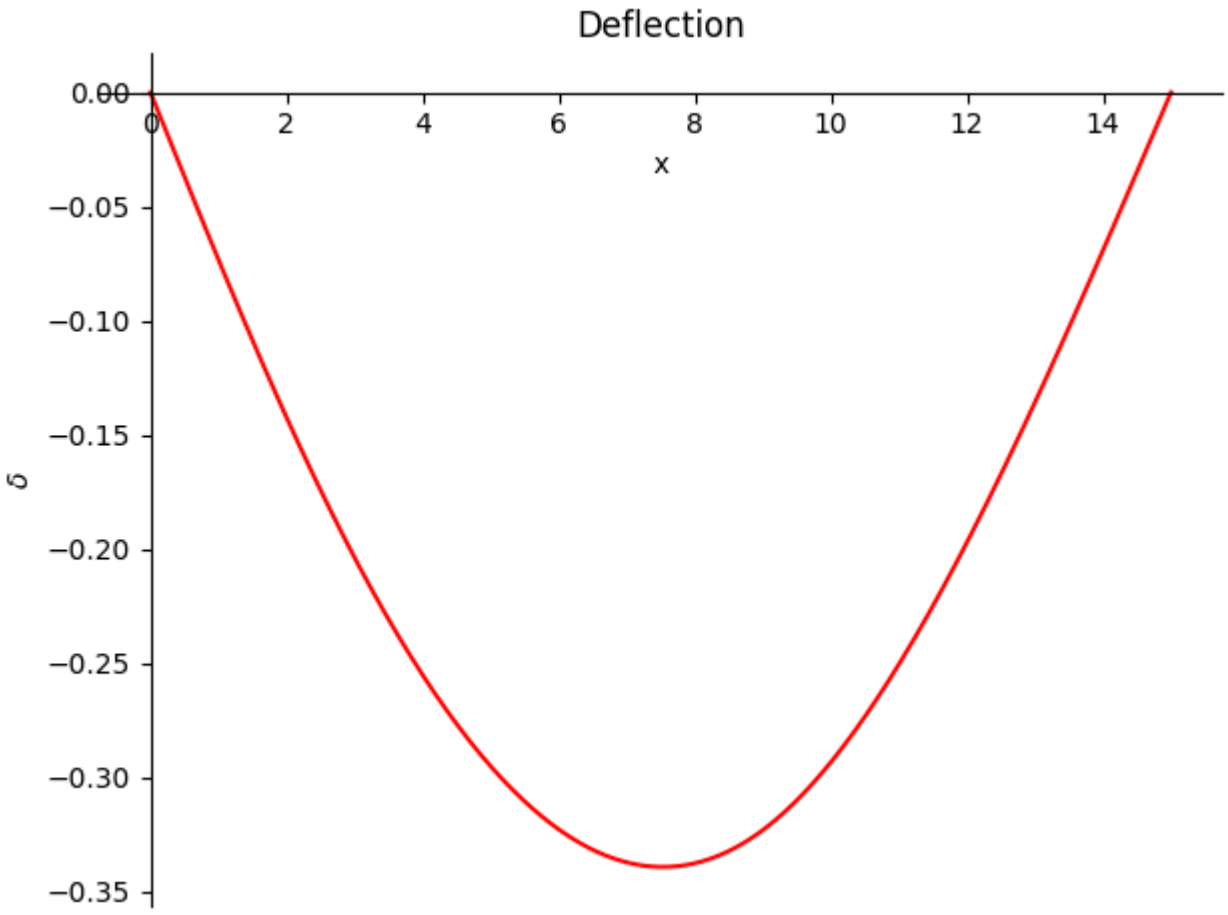
Let op: het assenstelsel is in SymPy gedefinieerd met de verticale as positief naar boven.

```
import sympy as sym
from sympy.physics.continuum_mechanics.beam import Beam
sym.init_printing()
```

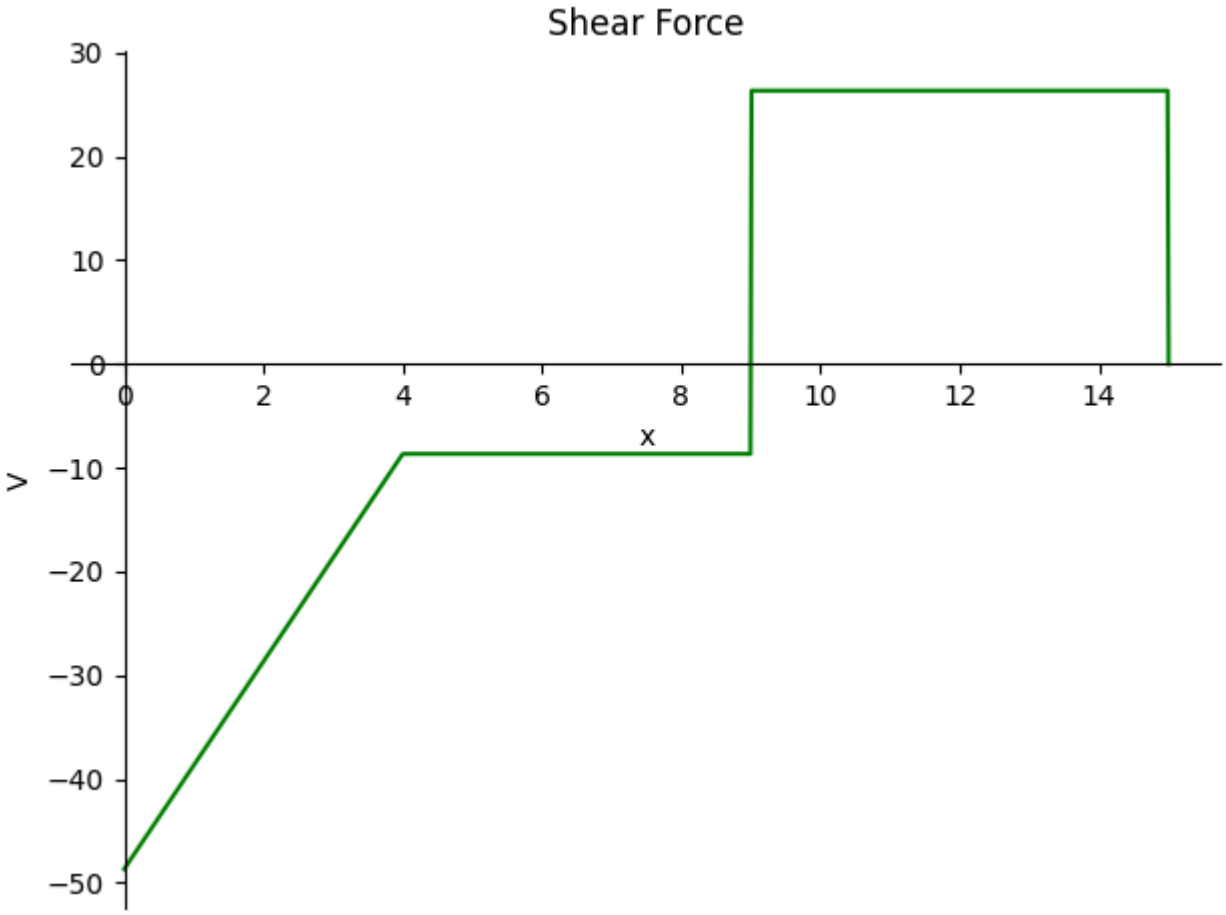
```
E, I = sym.symbols('E, I')
b = Beam(15, E, I)
A_v = b.apply_support(0,type='pin')
B_v = b.apply_support(15,type='roller')
b.apply_load(-10, 0, 0, 4)
b.apply_load(-35, 9, -1)
b.solve_for_reaction_loads(A_v, B_v)
b.deflection().simplify()
```

$$\frac{-26692x + 292\langle x\rangle^3 - 15\langle x\rangle^4 + 15\langle x-4\rangle^4 - 210\langle x-9\rangle^3 + 158\langle x-15\rangle^3}{36EI}$$

```
b.plot_deflection({E : 10000, I : 1});
```




```
b.plot_shear_force();
```


Shear Force
30
20
10
0
−10
−20
−30
−40
−50
0
2
4
6
8
10
12
14
x
V

List of current and future extensions

Extension	Status research	Status implementation SymPy
Normal forces	23 June 2023: Completed by Justus	-
Hinges	23 June 2023: Completed by Justus	24 June 2024: Completed by Mark
Spring connections	23 June 2023: Completed by Justus	-
Spring supports	23 June 2023: Completed by Justus	-
2D kinked structures in discrete definition	23 June 2023, 24 June 2024 : Completed by Justus and alternative by Alex	29-10-2024: Completed by Borek
Influence line for 1D structures	31 October 2023: Completed by Julia	24 June 2024: Completed by Mark
Influence line for 2D structures	31 October 2023: Completed by Julia	24 June 2024: Completed by Mark
(Curved) 1D structures with continuous parts with load in global coordinate system	17 December 2023: Completed by Ezzat	-
(Curved) 2D structures	-	-
Branching 2D structures with kinked parts	23 June 2023, 24 June 2024: Suggestions provided by Ezzat, completed by Alex	-
Branching curved 2D structures	23 June 2023: Suggestion provided by Ezzat	-
Looping 2D structures	24 June 2023: Completed by Alex	-

Extension	Status research	Status implementation SymPy
Truss structures	24 June 2023: Completed by Alex but not specifically for truss	-
Collapse mechanisms plasticity	-	-
Virtual work	-	-
3D structures	-	-
Develop GUI to use Macaulay’s method	-	-
Varying stiffness of elements	-	-
… (all ideas are welcome)		

Huidige uitbreidingen

Contents

Dit rapport zal een overzicht bevatten van de opgedane kennis met betrekking tot de methode van Macaulay. Dit rapport is bedoeld voor personen met mechanica kennis die nog onbekend zijn met de methode van Macaulay. In dit rapport zal de methode van Macaulay worden toegelicht en zal er worden aangegeven hoe deze toegepast kan worden.

Dit rapport bestaat uit 4 hoofdstukken:

1. De methode van Macaulay
2. Modelleren van krachten, opleggingen en verbindingen
3. Invloedslijnen met Macaulay
4. Oplossen van 2D constructies

In dit rapport wordt er gerefereerd naar de werken van van der Wulp (2023), Jankie (2023), Baudoin (2024) en van Gelder (2024) en wordt er hevig gebruik gemaakt van citaties en parafraseringen.

De methode van Macaulay

Belastingvergelijking

Uit de balkentheorie van Euler-Bernoulli kunnen de differentiaalvergelijkingen voor buiging (1) en extensie (2) worden afgeleid. Begrip over de afleiding en het gebruik van deze vergelijkingen wordt als voorkennis beschouwd.

$$EI\frac{d^4w}{dx^4} - q_z\left(x\right) = 0 \tag{1}$$

$$EA\frac{d^2w}{dx^2} + q_x\left(x\right) = 0 \tag{2}$$

Over het algemeen worden deze vergelijkingen gebruikt door liggers eerst in verschillende delen met gelijke belasting op te splitsen. Vervolgens kan ieder deel beschreven worden met de differentiaalvergelijking en wordt er gebruik gemaakt van overgangsvoorwaarden.

Bij de methode van Macaulay wordt er op een andere manier gebruik gemaakt van deze differentiaalvergelijkingen. De methode van Macaulay maakt gebruik van singulariteitsfuncties om de belasting op een ligger te beschrijven. Dit heeft als grote voordeel dat een ligger niet opgesplitst hoeft te worden in verschillende delen. Het gedrag van een ligger kan zo in één vergelijking worden beschreven.

Singulariteitsfuncties

Zoals hierboven aangegeven maakt de methode van Macaulay gebruik van singulariteitsfuncties. Dit is een familie van functies die gebaseerd is op de Dirac-delta functie. Het differentiëren en integreren van deze Dirac-delta functie vormt deze familie. De singulariteitsfunctie is wiskundig als volgt gedefinieerd:

Voor $n \geq 0$:

$$f\left(x\right) \equiv \left\langle x - \bar{x}\right\rangle^n = \begin{cases} \left(x - \bar{x}\right)^n & x \geq \bar{x} \\ 0 & x < \bar{x} \end{cases} \tag{3}$$

Voor $n < 0$:

$$f\left(x\right) \equiv \left\langle x - \bar{x}\right\rangle^n = 0 \tag{4}$$

Hierin geeft $\bar{x}$ de positie aan waar de singulariteit voorkomt, en is n de orde van de functie.

Hierbij is (4) aangepast waarbij er origineel op $x = \bar{x}$ geldt $f\left(x\right) = \infty$. Dit heeft echter geen betekening voor het balkmodel.

Het integreren van de singulariteitsfunctie is afhankelijk van zijn orde en is als volgt gedefinieerd:

$$\int \langle x-\bar{x}\rangle^n \mathrm{d}x = \begin{cases} (x-\bar{x})^{n+1} & n<0 \\ \dfrac{(x-\bar{x})^{n+1}}{n+1} & n\geq 0 \end{cases} \tag{5}$$

Een aantal voorbeelden zijn hieronder weergegeven

Voorbeeld $n=-1$

Voor $n=-1$ evalueert de functie (4) voor het gehele domein naar 0. Als de originele definitie wordt aangehouden met $f(\bar{x})=\infty$ staat deze functie ook wel bekend als de Dirac-Delta functie.

Voorbeeld $n=0$

Voor $n=0$ evalueert de functie (4) tot een stapfunctie, ook wel bekend als de Heaviside.

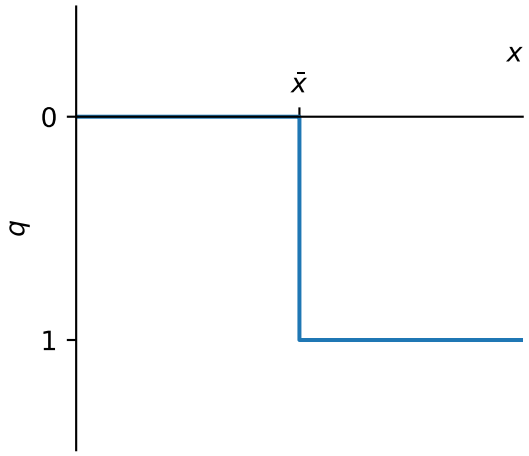


Fig. 2 Plot of $\langle x-\bar{x}\rangle^0$

Voorbeeld $n=1$

Voor $n=1$ geeft de functie (4) een linear verband na $\bar{x}$.

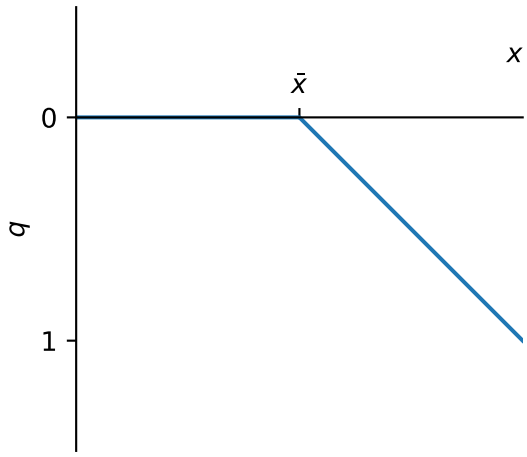


Fig. 3 Plot of $\langle x - \bar{x} \rangle^1$

Oplossingsstrategie

De eerste stap bij het oplossen van een probleem met de methode van Macaulay is het opstellen van de belastingvergelijking in de langs- en loodrechte richting. Hiervoor wordt de hele constructie gemodelleerd in één vergelijking per richting. Deze vergelijking kan vervolgens ingevuld worden als q in de differentiaalvergelijkingen in `ref`(hoofdstuk_1_1). Hoe verschillende onderdelen van een constructie gemodelleerd kunnen worden in deze vergelijking is beschreven in `ref`(hoofdstuk_2).

Als de belastingvergelijking is gevonden kan deze geïntegreerd worden. Dit geeft vergelijkingen die de normaalkrachten, dwarskracht, het moment, de rotatie, de doorbuiging en extensie van de staaf beschrijven. In deze vergelijkingen staan nog onbekende reactiekrachten en integratieconstanten.

Om deze onbekende reactiekrachten en integratieconstanten op te lossen kunnen de randvoorwaarden worden opgesteld. Met deze randvoorwaarden kunnen de onbekende waardes worden gevonden.

De gevonden waardes voor de reactiekrachten en integratieconstanten kunnen worden ingevuld in de vergelijkingen. Dit geeft de vergelijkingen die het gedrag van de staaf beschrijven.

Modelleren van belastingen, opleggingen en verbindingen

Assenstelsel

Het model maakt gebruik van een globaal en een lokaal assenstelsel. Het globale assenstelsel bestaat uit een horizontale en een verticale as waarbij de positieve richtingen naar rechts en naar beneden wijzen. Het lokale assenstelsel bestaat uit een x-as en een z-as. De positieve x-as wordt afgebeeld met een blauwe pijl zichtbaar in Fig. 4en de positieve z-as is altijd een kwartslag met de klok mee t.o.v. de positieve x-as.

Alle belastingen werken langs de globale assen, verdeelde belastingen werken langs de lengte van de staaf (zoals eigengewicht) en hoeken worden gemeten van de positieve v-as naar de positieve h-as

De externe krachten en de verplaatsingen worden globaal genoteerd en de snedekracht-diagrammen lokaal. Er zijn twee vergelijkingen die de lokale as volgen. $q_z(x)$ is voor de krachten loodrecht op de staaf en integreert naar de dwarskrachtenvergelijking, enz. en $q_x(x)$ is voor de krachten parallel aan de staaf en integreert naar de normaalkrachtenvergelijking, enz.

Voor een constructie in 1D komen de lokale en globale assen overeen en hebben alle staven dezelfde oriëntatie.

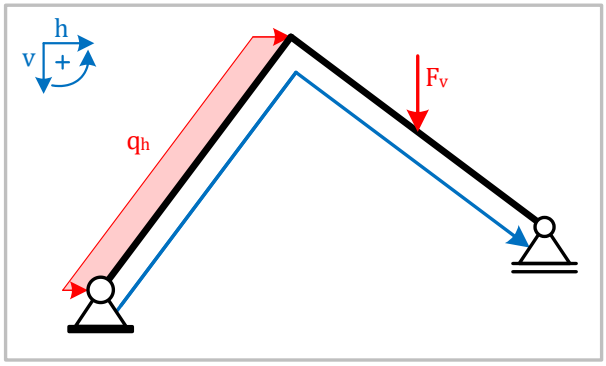


Fig. 4 Positieve assen in 2D constructie (Baudoin, 2024)

Modelleren van belastingen

Voor het modelleren van belastingen op de constructie wordt gebruik gemaakt van singulariteitsfuncties. Dit maakt het mogelijk om de volledige constructie in één regel (per richting) te beschrijven en daarmee de differentiaalvergelijking op te lossen. Er zal nu

toegelicht worden hoe verschillende krachten beschreven kunnen worden met singulariteitsfuncties.

Belastingen kunnen worden beschreven door de een singulariteitsfunctie van bijbehorende orde te vermenigvuldigen met de waarde van de kracht of verbinding. Naar aanleiding van het assenstelsel wordt een kracht positief of negatief gemodelleerd.

Puntlast

Een puntlast geeft belasting op een enkele plek in de constructie. Een puntlast kan gemodelleerd worden met de Dirac-delta functie. Een puntlast met een waarde F, ter plaatse van punt $\bar{x}$ waarbij de staaf een hoek $\theta\left(\bar{x}\right)$ heeft, wordt gemodelleerd zoals getoond in (6) en (7).

$$q_z\left(x\right) : F\langle x-\bar{x}\rangle^{-1} \qquad q_x\left(x\right) : F\langle x-\bar{x}\rangle^{-1} \tag{6}$$

In 2d wordt de kracht ontbonden in de langs- en loodrechte richting:

$$\begin{aligned} q_z\left(x\right) : &\qquad F_v\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right) + F_h\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right) \\ q_x\left(x\right) : &\qquad -F_v\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right) + F_h\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right) \end{aligned} \tag{7}$$

Koppel

Een koppel werkt ook op een enkel punt in de constructie. Een koppel wordt beschreven door de orde -2 in de krachtenvergelijking van de loodrechte richting aangezien dit koppel geen invloed heeft op de langsrichting. Een koppel met waarde T ter plaatse van punt $\bar{x}$ waarbij de staaf een hoek $\theta\left(\bar{x}\right)$ heeft, wordt als volgt gemodelleerd in 1D en 2D:

$$q_z\left(x\right) : T\langle x-\bar{x}\rangle^{-2} \tag{8}$$

$$\begin{aligned} q_z\left(x\right) : &\qquad T\langle x-\bar{x}\rangle^{-2} \\ q_x\left(x\right) : &\qquad 0 \end{aligned} \tag{9}$$

Uniform verdeelde belasting

Belasting die gelijkmatig verdeeld is over de constructie kan beschreven worden met twee Heaviside functies. Hierbij werkt de waarde van q_z eerste vanaf $\bar{x}_1$ in de daadwerkelijke richtingen en dan vanaf $\bar{x}_2$ in tegengestelde richting, dit geeft een totaal van 0 voor $x > \bar{x}_2$. Deze werking is in Fig. 5 weergegeven.

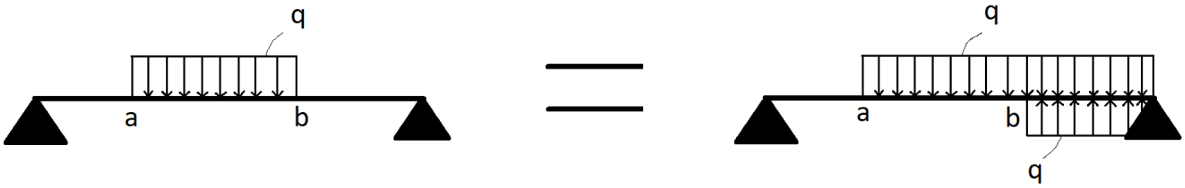


Fig. 5 Modelleren van een uniform verdeelde belasting (van der Wulp, 2023)

Dit kan beschreven worden met singulariteitsfuncties met groottes q_v en q_h van punt $\bar{x}_1$ tot x_2 waarbij de staaf een hoek $\theta\left(\bar{x}_1\right)$ en $\theta\left(\bar{x}_2\right)$ heeft in 1D en 2D:

$$\begin{aligned} q_z\left(x\right): &\quad q_v\langle x-\bar{x}_1\rangle^{-1} - q_v\langle x-\bar{x}_2\rangle^{-1} \\ q_z\left(x\right): &\quad q_h\langle x-\bar{x}_1\rangle^{-1} - q_h\langle x-\bar{x}_2\rangle^{-1} \end{aligned} \tag{10}$$

$$\begin{aligned} q_z\left(x\right): &\quad q_v\langle x-\bar{x}_1\rangle^{-1}\cos\left(\theta\left(\bar{x}_1\right)\right) - q_v\langle x_2-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}_2\right)\right) \\ &\quad + q_h\langle x-\bar{x}_1\rangle^{-1}\sin\left(\theta\left(\bar{x}_1\right)\right) - q_h\langle x-\bar{x}_2\rangle^{-1}\sin\left(\theta\left(\bar{x}_2\right)\right) \\ q_x\left(x\right): &\quad - q_v\langle x-\bar{x}_1\rangle^{-1}\sin\left(\theta\left(\bar{x}_1\right)\right) + q_v\langle x-\bar{x}_2\rangle^{-1}\sin\left(\theta\left(\bar{x}_2\right)\right) \\ &\quad + q_h\langle x-\bar{x}_1\rangle^{-1}\cos\left(\theta\left(\bar{x}_1\right)\right) - q_h\langle x-\bar{x}_2\rangle^{-1}\cos\left(\theta\left(\bar{x}_2\right)\right) \end{aligned} \tag{11}$$

Lineair en parabolisch verdeelde belastingen

Belastingen die niet uniform maar lineair of parabolisch verdeeld zijn kunnen op eenzelfde manier gemodelleerd worden als de uniform verdeelde belasting.

Modelleren van opleggingen en verbindingen

Naast belastingen kunnen ook opleggingen en verbindingen worden gemodelleerd door middel van singulariteitsfuncties. Op de plek waar een oplegging of verbinding wordt toegevoegd gelden ook randvoorwaarden. Deze randvoorwaarden worden gebruikt om de vergelijkingen uiteindelijk op te lossen. Er zal nu worden toegelicht hoe verschillende opleggingen en verbindingen kunnen worden beschreven door middel van singulariteitsfuncties en welke randvoorwaarden daarbij horen. Er wordt in dit hoofdstuk

alleen gekeken naar krachten loodrecht op de staaf en momenten. Krachten evenwijdig aan de staaf worden behandeld in het `ref`(hoofdstuk_2.3)Modelleren van opleggingen en verbindingen voor extensie.

Inklemming

Bij een inklemming kunnen een verticale oplegreactie en een inklemmingsmoment optreden. In Fig. 6 is weergegeven hoe een inklemming gemodelleerd kan worden.

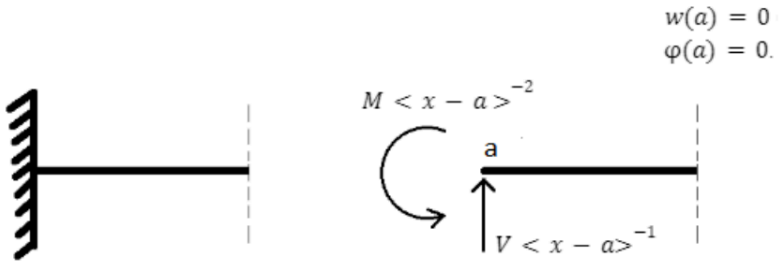


Fig. 6 Modelleren van een inklemming (van der Wulp, 2023)

Dit kan beschreven worden met singulariteitsfuncties met ordes -1 en -2 met onbekende groottes R_v, R_h en T op punt $\bar{x}$ waarbij de staaf een hoek $\theta\left(\bar{x}\right)$ heeft in 1D en 2D:

$$\begin{aligned} q_z\left(x\right): &\quad R_v\langle x-\bar{x}\rangle^{-1}+T\langle x-\bar{x}\rangle^{-2} \\ q_x\left(x\right): &\quad R_h\langle x-\bar{x}\rangle^{-1} \end{aligned} \tag{12}$$

$$\begin{aligned} q_z\left(x\right): &\quad R_h\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right)+R_v\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right) &&+T\langle x-\bar{x}\rangle \\ q_x\left(x\right): &\quad R_h\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right)-R_v\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right) \end{aligned} \tag{13}$$

Daarnaast introduceert een inklemming ook drie randvoorwaarden:

$$\begin{aligned} \varphi\left(\bar{x}\right)&=0 \\ u_v\left(\bar{x}\right)&=0 \\ u_h\left(\bar{x}\right)&=0 \end{aligned} \tag{14}$$

Scharnier- en roloplegging

Aangezien krachten in de richting van de staaf nog niet worden meegenomen, kunnen scharnier- en rolopleggingen op dezelfde manier worden gemodelleerd. In Fig. 7 en Fig. 8 is weergegeven hoe een scharnieroplegging gemodelleerd kan worden.

Fig. 7 Modelleren van en scharnieroplegging (van der Wulp, 2023)

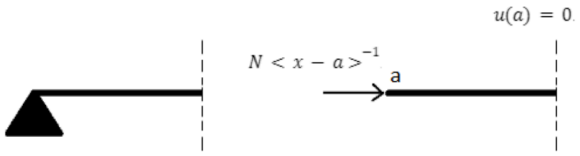


Fig. 8 Modelleren van een scharnieroplegging bij extensie (van der Wulp, 2023)

Dit kan beschreven worden met singulariteitsfuncties met ordes -1 met onbekende groottes R_v, R_h en T op punt $\bar{x}$ waarbij de staaf een hoek $\theta\left(\bar{x}\right)$ heeft in 1D en 2D:

$$\begin{aligned} q_z\left(x\right): &\qquad R_v\langle x-\bar{x}\rangle^{-1} \\ q_x\left(x\right): &\qquad R_h\langle x-\bar{x}\rangle^{-1} \end{aligned} \tag{15}$$

$$\begin{aligned} q_z\left(x\right): &\qquad R_h\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right)+R_v\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right) \\ q_x\left(x\right): &\qquad R_h\langle x-\bar{x}\rangle^{-1}\cos\left(\theta\left(\bar{x}\right)\right)-R_v\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}\right)\right) \end{aligned} \tag{16}$$

Daarnaast introduceert een scharnier ook twee randvoorwaarden:

$$\begin{aligned} u_v\left(\bar{x}\right)&=0 \\ u_h\left(\bar{x}\right)&=0 \end{aligned} \tag{17}$$

Bij een rolscharnier vervallen de krachten en vergelijkingen in vrijgemaakte richting

Scharnier

Scharnierende verbindingen in constructies kunnen worden beschreven met singulariteitsfuncties van orde -3. Dit valt te beredeneren uit het feit dat scharnieren een sprong in de krommingslijn geven, en daar dus orde 0 hebben. Terug differentiëren naar de belasting vergelijking geeft een singulariteitsfunctie van orde -3.

In Fig. 9 is weergegeven hoe een scharnier gemodelleerd kan worden.

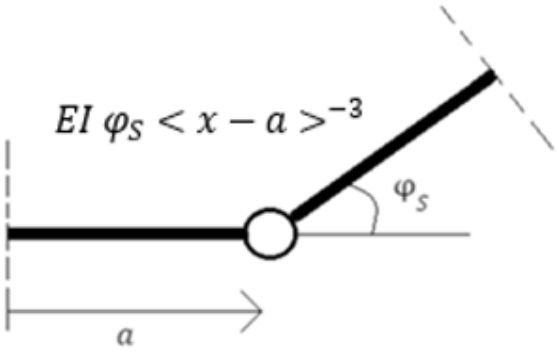


Fig. 9 Modelleren van een scharnier (van der Wulp, 2023), correctie (van Gelder, 2024)

Dit kan beschreven worden met singulariteitsfuncties met orde -3 met onbekende grootte $\Delta\varphi$ op punt $\bar{x}$ in zowel 1D als 2D met dezelfde vergelijking:

$$q_z\left(x\right) : EI\ \Delta\varphi\langle x - \bar{x}\rangle^{-3} \tag{18}$$

Een scharnier introduceert ook een randvoorwaarde. Ter plaatse van het scharnier is het moment (M) gelijk aan 0.

$$M\left(\bar{x}\right) = 0 \tag{19}$$

Schuifscharnier

Schuifscharnieren in een constructie kunnen worden beschreven met singulariteitsfuncties van orde -4. Zoals roterende scharnieren een sprong geven in de krommingslijn, geven schuifscharnieren een sprong in de doorbuigingslijn. Hieruit volgt dat schuifscharnieren in de belasting vergelijking beschreven worden met een singulariteitsfunctie van orde -4.

In Fig. 10 is weergegeven hoe een schuifscharnier gemodelleerd kan worden.

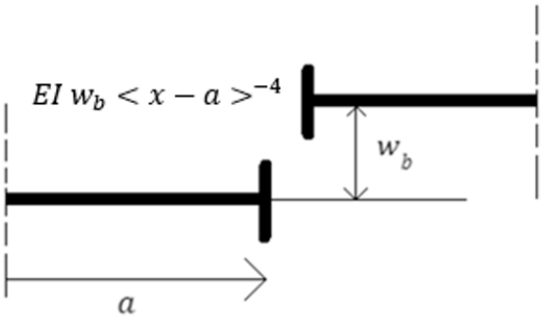


Fig. 10 Modelleren van een scharnier (van der Wulp, 2023), correctie (van Gelder, 2024)

Dit kan beschreven worden met singulariteitsfuncties met ordes -4 met onbekende grootte Δu_z op punt $\bar{x}$:

$$q_z\left(x\right) : EI\ \Delta u_z \langle x - \bar{x} \rangle^{-4} \tag{20}$$

Een schuifscharnier introduceert ook een randvoorwaarde. Ter plaatse van het schuifscharnier is de dwarskracht (V) gelijk aan 0.

$$V\left(\bar{x}\right) = 0 \tag{21}$$

Telescoopscharnieren

Deze scharnieren hebben axiale verplaatsing als vrijheidsgraad. Dit geeft een sprong in de vergelijking voor axiale verplaatsing ter plaatse van het scharnier. Deze spong kan gemodelleerd worden met de Heaviside functie. Terug differentiëren naar de belastingvergelijking geeft dan een singulariteitsfunctie met orde -2.

In Fig. 11 is weergegeven hoe een telescoopscharnier gemodelleerd kan worden.

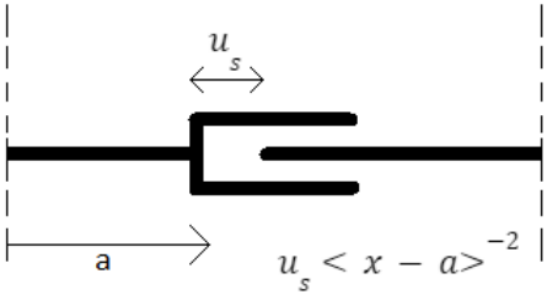


Fig. 11 Modelleren van telescoop scharnieren (van der Wulp, 2023)

Dit kan beschreven worden met singulariteitsfuncties met ordes -2 met onbekende grootte Δu_x op punt $\bar{x}$:

$$q_x\left(x\right) : EA\ \Delta u_x \langle x - \bar{x} \rangle^{-2} \tag{22}$$

Een telescoopscharnier introduceert ook een randvoorwaarde. Ter plaatse van het telescoopscharnier is de normaalkracht (N) gelijk aan 0.

$$N\left(\bar{x}\right) = 0 \tag{23}$$

Verende opleggingen

Er zijn twee soorten verende opleggingen: verende opleggingen die verplaatsing tegengaan en verende opleggingen die rotatie tegen gaan. In beide gevallen wordt de veer gemodelleerd als kracht ter plaatse van de oplegging in de belasting vergelijking.

In Fig. 12, Fig. 13 en Fig. 14 is weergegeven hoe de veren gemodelleerd kunnen worden.

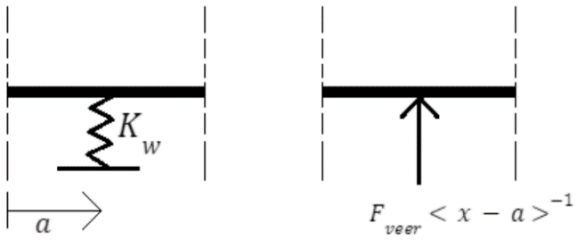


Fig. 12 Modelleren van een translerende verende oplegging (van der Wulp, 2023)

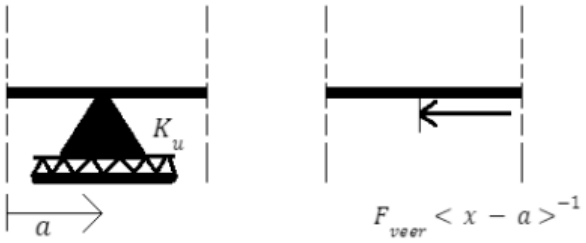


Fig. 13 Modelleren van een verende scharnieroplegging bij extensie (van der Wulp, 2023)

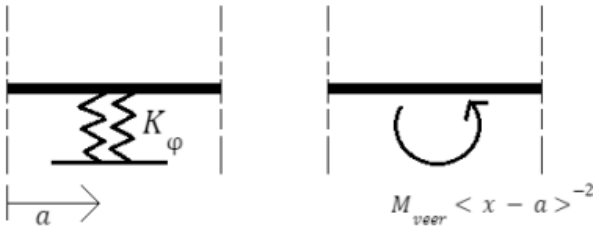


Fig. 14 Modelleren van een roterende verende oplegging (van der Wulp, 2023)

Een verende oplegging in alle richtingen die verticale verplaatsing tegengaat wordt gemodelleerd als puntlast met een grootte van $F_{\text{veer},v}$, $F_{\text{veer},h}$, T_{veer}.

Dit kan beschreven worden met singulariteitsfuncties met ordes -1 en -2 met onbekende groottes R_v, R_h en T op punt $\bar{x}$ waarbij de staaf een hoek $\theta(\bar{x})$ heeft in 1D en 2D:

$$\begin{aligned} q_z(x): &\quad F_{\text{veer},v}\langle x-\bar{x}\rangle^{-1} + T_{\text{veer}}\langle x-\bar{x}\rangle^{-2} \\ q_x(x): &\quad F_{\text{veer},h}\langle x-\bar{x}\rangle^{-1} \end{aligned} \tag{24}$$

$$
\begin{aligned}
q_z(x): \quad & F_{\text{veer},h}\langle x-\bar{x}\rangle^{-1}\sin(\theta(\bar{x})) + F_{\text{veer},v}\langle x-\bar{x}\rangle^{-1}\cos(\theta(\bar{x})) + T \\
q_x(x): \quad & R_{\text{veer},h}\langle x-\bar{x}\rangle^{-1}\cos(\theta(\bar{x})) - R_{\text{veer},v}\langle x-\bar{x}\rangle^{-1}\sin(\theta(\bar{x}))
\end{aligned} \tag{25}
$$

De randvoorwaardes die hierbij worden geintroduceerd zijn bevatten de veerconstantes K_T, K_V en K_N:

$$
\begin{aligned}
T_{\text{veer}} &= -K_M\varphi\ (\bar{x}) \\
F_{\text{veer},v} &= -K_V\, u_v(\bar{x}) \\
F_{\text{veer},h} &= -K_N\, u_h(\bar{x})
\end{aligned} \tag{26}
$$

Een deels verende oplegging is natuurlijk ook mogelijk met het weglaten van de desbetreffende termen.

Verende verbindingen

Verende verbindingen worden gemodelleerd als de verbinding waar het om gaat met een andere randvoorwaarde.

In Fig. 15, Fig. 16 en Fig. 17 is weergegeven hoe de verende verbindingen gemodelleerd kunnen worden.

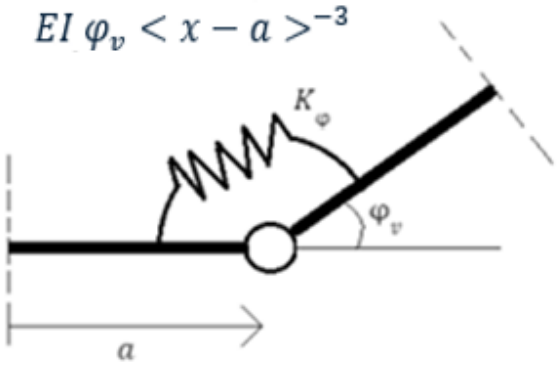


Fig. 15 Modelleren van een roterende verende verbinding (van der Wulp, 2023), correctie (van Gelder, 2024)

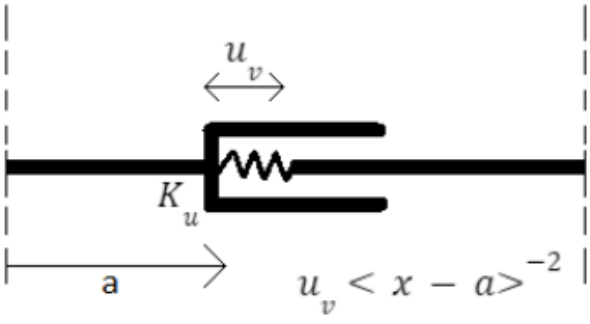


Fig. 16 Modelleren van verende telescoopscharnieren (van der Wulp, 2023)

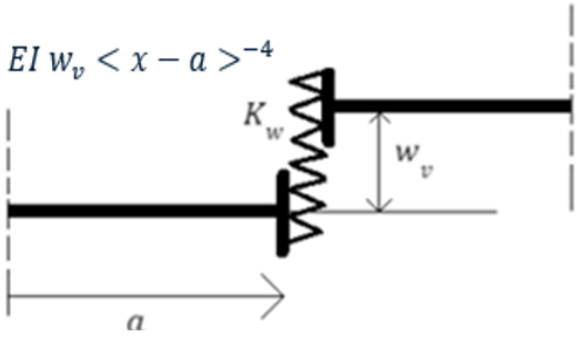


Fig. 17 Modelleren van een translerende verende verbinding (van der Wulp, 2023), correctie (van Gelder, 2024)

De veer wordt dus beschreven als een scharnier met krachten met een grootte van $F_{\text{veer},V}$, $F_{\text{veer},N}$, T_{veer} . Dit kan beschreven worden met singulariteitsfuncties met orde -3 met onbekende grootte $\Delta\varphi$, Δu_z, Δu_x op punt $\bar{x}$ in 1D:

$$\begin{aligned} q_z(x): \quad & EI\,\Delta\varphi\langle x-\bar{x}\rangle^{-3} + T_{\text{veer}}\,\langle x-\bar{x}\rangle^{-2} \\ & EI\,\Delta u_z\langle x-\bar{x}\rangle^{-4} + F_{\text{veer},V}\langle x-\bar{x}\rangle^{-1} \\ q_x(x): \quad & EA\,\Delta u_x\langle x-\bar{x}\rangle^{-2} + F_{\text{veer},N}\langle x-\bar{x}\rangle^{-1} \end{aligned} \tag{27}$$

In 2D worden de reactiekrachten berekend in het lokale assenstelsel maar moeten deze worden omgeschreven naar het globale assenstelsel in de belastingsvergelijking.

De bijbhorende voorwaardes zijn dan als volgt:

$$\begin{aligned} M(\bar{x}) = \quad & T_{\text{veer}} = -K_M \cdot \Delta\varphi \\ V(\bar{x}) = \quad & F_{\text{veer},V} = -K_V \cdot \Delta u_z \\ N(\bar{x}) = \quad & F_{\text{veer},N} = -K_N \cdot \Delta u_x \end{aligned} \tag{28}$$

Ook hier is een deels verende verbinding is natuurlijk ook mogelijk met het weglaten van de desbetreffende termen.

Vergelijking voor horizontale en verticale verplaatsingen in 2D

De doorbuiging ($u_z(x)$) en extensie ($u_x(x)$) zijn voor tweedimensionale constructies hele abstracte termen die niet bruikbaar zijn voor de oplossingsvoorwaarden. Daarom moeten ze worden gebruikt als invoer voor de vergelijkingen van de horizontale en verticale

verplaatsing van de constructie ($u_h(x)$ en $u_v(x)$). De vergelijkingen stellen telkens dat de invloed van de doorbuiging en extensie van een staaf ij ($u_{z,ij}(x)$ en $u_{x,ij}(x)$) op $u_z(x)$ en $u_z(x)$ gelijk aan nul is als $x < a_i$, een functie is als $a_i \leq x < a_j$ en constant is als $a_j \leq x$. Dit wordt afgebeeld in Figuur Fig. 18 en Fig. 19.

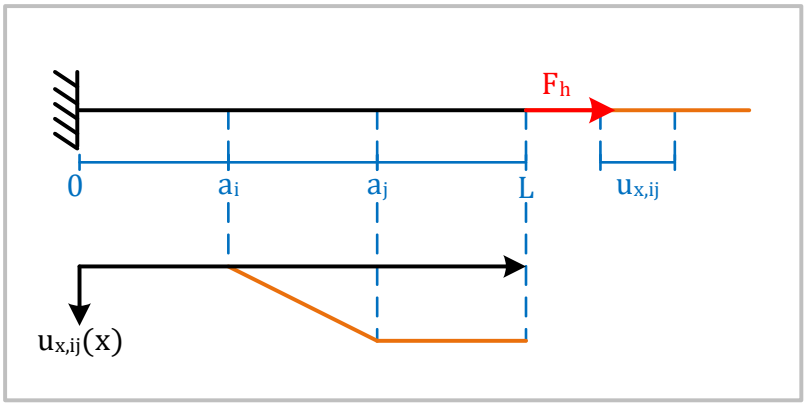


Fig. 18 De invloed van $u_{x,ij}(x)$ op $u_x(x)$ (Baudoin, 2024)

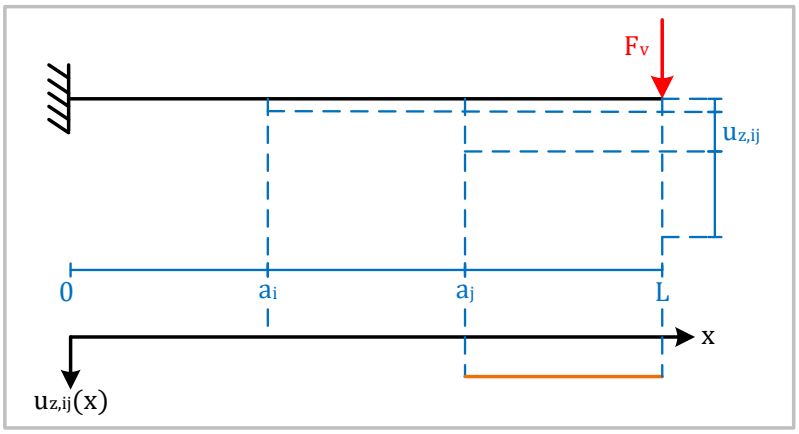


Fig. 19 De invloed van $u_{z,ij}(x)$ op $u_z(x)$ (Baudoin, 2024)

Vervolgens worden $u_{z,ij}(x)$ en $u_{x,ij}(x)$ vermenigvuldigd met een functie van de hoek van de staaf wat tot slot voor alle staven wordt opgesomd. Hieruit ontstaan vgl. (29) en (30).

$$u_v(x) = u_z(0)\cos(\theta_0) - u_x(0)\sin(\theta_0) + \sum_{i=0}^{i=n} \begin{pmatrix} \left((u_z(x) - u_z(a_i))\langle x - a_i\rangle^0 - \right. \\ \left. - \left((u_x(x) - u_x(a_i))\langle x - a_i\rangle^0 \right. \end{pmatrix} \quad (29)$$

$$u_h(x) = u_z(0)\sin(\theta_0) + u_x(0)\cos(\theta_0) + \sum_{i=0}^{i=n} \begin{pmatrix} \left((u_z(x) - u_z(a_i))\langle x - a_i\rangle^0 - \right. \\ \left. + \left((u_x(x) - u_x(a_i))\langle x - a_i\rangle^0 \right. \end{pmatrix} \quad (30)$$

Invloedslijnen met Macaulay

De methode van Macaulay kan worden gebruikt om invloedslijnen te berekenen. In dit hoofdstuk zal worden toegelicht hoe de methode hierbij kan worden ingezet. Hierbij zal de focus liggen op de basis van invloedslijnen met Macaulay, door te beschrijven hoe invloedslijnen berekend kunnen worden voor eendimensionale constructies.

Methode

De eerste stap voor het berekenen van invloedslijnen met de methode van Macaulay is het beschrijven van de krachtsverdeling op de ligger. Hierbij worden alle krachten die op de ligger spelen gedefinieerd op een positie x, waarin x de variabele is die de positie op de ligger beschrijft. De puntlast waarvoor de invloedslijnen worden bepaald wordt gedefinieerd op een positie $x = \bar{x}_F$. Deze methode is weergegeven in figuur Fig. 20.

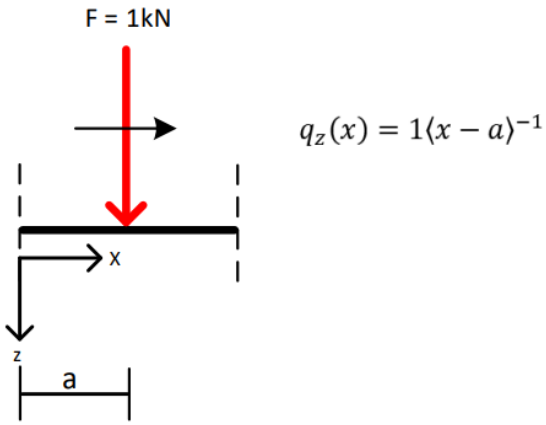


Fig. 20 Methode voor invloedslijnen met Macaulay (Jankie, 2023)

Vervolgens kan de belastingvergelijking worden geïntegreerd naar x. Dit zal resulteren in vergelijkingen met onbekende reactiekrachten en integratieconstanten.

De randvoorwaarden kunnen worden opgesteld. Deze worden gebruikt om de onbekende reactiekrachten en integratieconstanten uit de vergelijkingen op te lossen.

Invullen van de gevonden waardes voor de onbekenden in de geïntegreerde belastingvergelijking resulteert in de gewenste functies. Deze functies hangen af van x en $\bar{x}_F$. Waarin voor x een positie op de ligger ingevuld kan worden en $\bar{x}_F$ de locatie van de puntlast beschrijft. Zo kan de invloed van een puntlast op locatie $\bar{x}_F$ op de staafeigenschappen op locatie x worden bepaald.

Oplossen van geknikte, vertakte en gesloten constructies

De Macaulay methode kan worden uitgebreid zodat het ook toepasbaar is op geknikte, vertakte en gesloten, constructies. De belangrijkste sterktepunten van het model zijn dat het systematisch en robuust is doordat het is gebaseerd op de principes van evenwicht en continuïteit, de snedekrachten en verplaatsingen van elk punt kunnen worden bepaald doordat de functie continue is en dat de oplossingsvoorwaarden geringer in tal en eenvoudiger zijn dan voor de traditionele integratiemethode.

Modelleren van knikken

De belastingen worden in 2D anders gemodelleerd dan in 1D. De vergelijking voor elke belasting wordt voor elke knik aangepast waar de functie van x langs gaat na het aangrijppunt van de belasting. De hoektermen zijn bepaald door te analyseren wat de invloed is van een hoek op de snedekrachtdiagrammen van een belasting.

Voor elk segment wordt een hoek bepaald, welke kunnen worden samengevoegd in $\theta\left(\bar{x}\right)$.

Voor elke kracht j (voor elke F_j en q_j inclusief oplegreacties en krachten uit knooppunten) wordt de belastingsvergelijking aangepast voor elke knik i met n het aantal knikken en met een bijbehorende hoek $\theta\left(\bar{x}_i\right)$. Daarbij zijn de locaties :

$$
\begin{aligned}
q_z(x): \quad & \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{F_j} \right\rangle^0 F_{v,j} \langle x - \bar{x}_i \rangle^{-1} \\
& + \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{F_j} \right\rangle^0 F_{h,j} \langle x - \bar{x}_i \rangle^{-1} \\
& + \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{q_j} \right\rangle^0 q_{v,j} \left(\langle x - \bar{x}_i \rangle^0 + \langle x - \bar{x}_i \rangle^{-1} \left(\bar{x}_i - \bar{x}_{q_j} \right) \right) \\
& + \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{q_j} \right\rangle^0 q_{h,j} \left(\langle x - \bar{x}_i \rangle^0 + \langle x - \bar{x}_u \rangle^{-1} \left(\bar{x}_i - \bar{x}_{q_j} \right) \right) \\
q_x(x): \quad & \sum_j \sum_i^n - \left\langle \bar{x}_i - \bar{x}_{F_j} \right\rangle^0 F_{v,j} \langle x - \bar{x}_i \rangle^{-1} \\
& + \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{F_j} \right\rangle^0 F_{h,j} \langle x - \bar{x}_i \rangle^{-1} \\
& + \sum_j \sum_i^n - \left\langle \bar{x}_i - \bar{x}_{q_j} \right\rangle^0 q_{v,j} \left(\langle x - \bar{x}_i \rangle^0 + \langle x - \bar{x}_i \rangle^{-1} \left(\bar{x}_i - \bar{x}_{q_j} \right) \right) \\
& + \sum_j \sum_i^n \left\langle \bar{x}_i - \bar{x}_{q_j} \right\rangle^0 q_{h,j} \left(\langle x - \bar{x}_i \rangle^0 + \langle x - \bar{x}_u \rangle^{-1} \left(\bar{x}_i - \bar{x}_{q_j} \right) \right)
\end{aligned}
$$

Een koppel heeft geen invloed op de normaal- en dwarskrachten. Verder blijft het buigend moment van een koppel onveranderd bij een knikpunt.

Bij de normaal- en dwarskrachtenlijn van een puntlast ontstaat een sprong als de functie een hoekpunt passeert. Tegelijkertijd ontstaat er een knik in de momentenlijn als de functie een hoekpunt passeert. Deze sprong en knik ontstaan door de verandering van de projectie van de puntlast.

Bij de normaal- en dwarskrachtenlijn van een uniform verdeelde belasting ontstaat zowel een sprong als een knik als de functie een hoekpunt passeert. De knik ontstaat door de verandering van de projectie van de verdeelde belasting. De sprong daarentegen ontstaat door een verandering in de projectie van de arbitrale belasting vervangend puntlast.

Vertakte en gesloten constructies

Met de tot nu toe in dit hoofdstuk verstreken informatie kunnen geknikte constructies worden gemodelleerd met de Macaulay methode. Voor vertakte en gesloten constructies bestaan er drie bijzondere punten met unieke eigenschappen. Dit zijn het knooppunt (a_k), sprongpunt (a_s) en aansluitpunt (a_a).

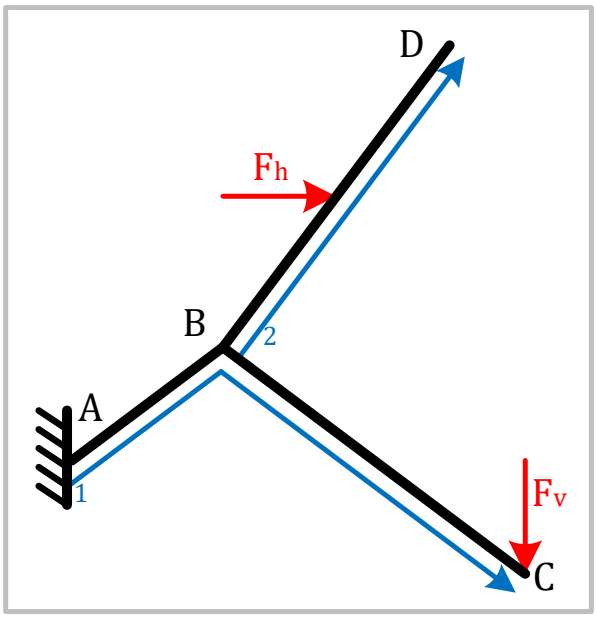


Fig. 21 Vertakte constructie (Baudoin, 2024)

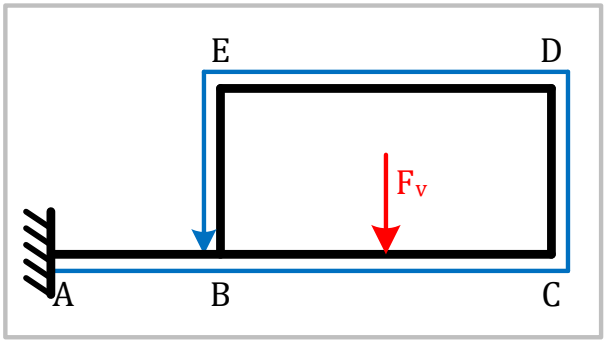


Fig. 22 Gesloten constructie (Baudoin, 2024)

Knooppunt

Het knooppunt $\bar{x}_1$ is het punt waarbij de functie van x voor het eerst langs een knoop gaat (zoals pijl 1 op knoop B in Fig. 21). Op $\bar{x}_1$ moet voor elke staaf waarnaar de functie niet direct de weg vervolgd drie onbekende krachten worden toegevoegd. De formules voor deze krachten volgen bij het hieropvolgende sprongpunt en aansluitpunt.

Sprongpunt

Het sprongpunt, (zie Fig. 23) is het punt waar de functie van x in het globale assenstelsel een sprong maakt van punt $\bar{x}_{\text{sprong}}$, wat een uiteinde of afsluiting is van een vertakking, naar punt $\bar{x}_2$, wat een knoop is waar de functie van x al eerder langs is geweest (op $\bar{x}_1$). Een voorbeeld is van punt C naar knoop B in figuur Fig. 21. In het lokale assenstelsel verloopt de functie echter continue.

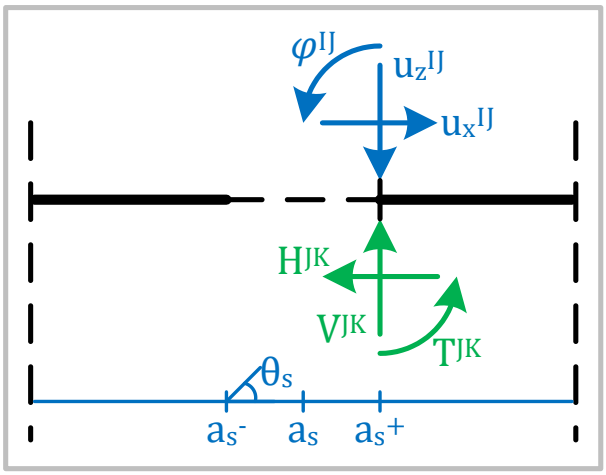


Fig. 23 Modelleren van een sprongpunt (Baudoin, 2024)

Op $\bar{x}_1$ moeten de onbekende krachten δV, δH en ΔT worden toegevoegd waarnaar de functie na de sprong de weg vervolgd. Op $\bar{x}_2$ moeten dezelfde krachten weer worden afgetrokken. Verder moeten op $\bar{x}_2$ de sprongconstanten $\Delta\varphi$, Δu_z en Δu_x toegevoegd worden voor de sprong in vervormingen tussen punt $\bar{x}_{\text{sprong}}$ en $\bar{x}_2$. Deze worden als volgt gemodelleerd:

$$\begin{aligned}
q_z(x): \quad & \Delta V\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta V\langle x-\bar{x}_2\rangle^{-1}\cos(\theta(\bar{x}_2)) \\
& + \Delta H\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}\rangle^{-1}\sin(\theta(\bar{x}_2)) \\
& + \Delta T\langle x-\bar{x}_1\rangle^{-2} - \Delta T\langle x-\bar{x}_2\rangle^{-2} \\
& + EI\Delta\varphi\langle x-\bar{x}_2\rangle^{-3} + EI\Delta u_z\langle x-\bar{x}_3\rangle^{-4} \\
q_x(x): \quad & -\Delta V\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) + \Delta V\langle x-\bar{x}_2\rangle^{-1}\sin(\theta(\bar{x}_2)) \\
& + \Delta H\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}_2\rangle^{-1}\cos(\theta(\bar{x}_2)) \\
& + EI\Delta u_x\langle x-\bar{x}_2\rangle^{-2}
\end{aligned} \tag{32}$$

De onbekenden kunnen worden opgelost met de volgende vergelijkingen:

$$\begin{aligned}
N\left(\bar{x}^{+}_{\text{sprong}}\right) &= 0 \\
V\left(\bar{x}^{+}_{\text{sprong}}\right) &= 0 \\
M\left(\bar{x}^{+}_{\text{sprong}}\right) &= 0 \\
\varphi(\bar{x}_1) &= \varphi(\bar{x}_2) \\
u_v(\bar{x}_1) &= u_v(\bar{x}_2) \\
u_h(\bar{x}_1) &= u_h(\bar{x}_2)
\end{aligned} \tag{33}$$

Als de functie springt naar een scharnierend verbonden staaf op een knooppunt, dan kunnen de oplosvergelijkingen voor de hoekverdraaiing weggelaten worden. Er hoeven dan ook geen alternatieve voorwaarden worden gesteld.

Aansluitpunt

Het aansluitpunt is het punt waarbij de functie van x opnieuw langs een knoop gaat (zoals knoop B vanuit punt E in figuur Fig. 22). Dezelfde vergelijkingen als voor een sprongpunt gelden, waarbij $\Delta\varphi$, Δu_z en Δu_x weg kunnen worden gelaten. Ditmaal is $\bar{x}_1$ de eerste keer dat de functie langs de knoop komt en $\bar{x}_2$ de twee keer:

$$\begin{aligned} q_z\left(x\right): \quad & \Delta V\langle x-\bar{x}_1\rangle^{-1}\cos\left(\theta\left(\bar{x}_1\right)\right)-\Delta V\langle x-\bar{x}_2\rangle^{-1}\cos\left(\theta\left(\bar{x}_2\right)\right) \\ & +\Delta H\langle x-\bar{x}_1\rangle^{-1}\sin\left(\theta\left(\bar{x}_1\right)\right)-\Delta H\langle x-\bar{x}\rangle^{-1}\sin\left(\theta\left(\bar{x}_2\right)\right) \\ & +\Delta T\langle x-\bar{x}_1\rangle^{-2}-\Delta T\langle x-\bar{x}_2\rangle^{-2} \\ q_x\left(x\right): \quad & -\Delta V\langle x-\bar{x}_1\rangle^{-1}\sin\left(\theta\left(\bar{x}_1\right)\right)+\Delta V\langle x-\bar{x}_2\rangle^{-1}\sin\left(\theta\left(\bar{x}_2\right)\right) \\ & +\Delta H\langle x-\bar{x}_1\rangle^{-1}\cos\left(\theta\left(\bar{x}_1\right)\right)-\Delta H\langle x-\bar{x}_2\rangle^{-1}\cos\left(\theta\left(\bar{x}_2\right)\right) \end{aligned} \tag{34}$$

De onbekenden kunnen worden opgelost met de volgende vergelijkingen:

$$\begin{aligned} \varphi\left(\bar{x}_1\right) &= \varphi\left(\bar{x}_2\right) \\ u_v\left(\bar{x}_1\right) &= u_v\left(\bar{x}_2\right) \\ u_h\left(\bar{x}_1\right) &= u_h\left(\bar{x}_2\right) \end{aligned} \tag{35}$$

Ook hier geldt dat als als de functie scharnierend aansluit op een knooppunt, de oplosvergelijkingen voor de hoekverdraaiing weggelaten kunnen worden. Er hoeven dan ook geen alternatieve voorwaarden worden gesteld.

Overzicht modellering discontinuïteiten

In Table 1 Table 2 (Baudoin, 2024) wordt voorgeschreven welke oplossingsvoorwaarden gelden voor starre onstructies als lokale afstand $\bar{x}$ de benoemde eigenschap voor 1D en 2D situaties.

Table 1 Belastingfunctie, vergelijkingen en or

Situatie op $\bar{x}$	**Belastingfunctie**
Beginpunt en eindpunt constructie	
Puntlast	$q_z\left(x\right) : F\langle x-\bar{x}\rangle^{-1} \qquad q_x\left(x\right) : F\langle x-\bar{x}\rangle^{-1}$
Koppel	$q_z\left(x\right) : T\langle x-\bar{x}\rangle^{-2}$
Verdeelde belasting	$q_z\left(x\right) : \quad q_v\langle x-\bar{x}_1\rangle^{-1} - q_v\langle x-\bar{x}_2\rangle^{-1}$ $q_z\left(x\right) : \quad q_h\langle x-\bar{x}_1\rangle^{-1} - q_h\langle x-\bar{x}_2\rangle^{-1}$ Met $\bar{x}_1$ en $\bar{x}_2$ het begin en eind van de verdeelde belasting
Scharnier-/schuif-/telescoop-verbinding	$q_z\left(x\right) : EI\ \Delta\varphi\langle x-\bar{x}\rangle^{-3} q_z\left(x\right) : EI\ \Delta u_z\langle x-\bar{x}\rangle^{-4} q_x\left(x\right.$
Inklemming	$q_z\left(x\right) : \quad R_v\langle x-\bar{x}\rangle^{-1} + T\langle x-\bar{x}\rangle^{-2}$ $q_x\left(x\right) : \quad R_h\langle x-\bar{x}\rangle^{-1}$
Verende oplegging	$q_z\left(x\right) : \quad F_{\text{veer},v}\langle x-\bar{x}\rangle^{-1} + T_{\text{veer}}\ \langle x-\bar{x}\rangle^{-2}$ $q_x\left(x\right) : \quad F_{\text{veer},h}\langle x-\bar{x}\rangle^{-1}$
Verende scharnierverbinding	$q_z\left(x\right) : \quad EI\ \Delta\varphi\langle x-\bar{x}\rangle^{-3} + T_{\text{veer}}\ \langle x-\bar{x}\rangle^{-2}$ $EI\ \Delta u_z\langle x-\bar{x}\rangle^{-4} + F_{\text{veer},V}\langle x-\bar{x}\rangle^{-1}$ $q_x\left(x\right) : \quad EA\ \Delta u_x\langle x-\bar{x}\rangle^{-2} + F_{\text{veer},N}\langle x-\bar{x}\rangle^{-1}$

Table 2 Belastingfunctie, v

Situatie op $\bar{x}$	**Belastingsfunctie**
Beginpunt en eindpunt constructie	
Puntlast	$q_z(x): \quad F_v\langle x-\bar{x}\rangle^{-1}\cos(\theta(\bar{x}))+F_h\langle x-\bar{x}\rangle^{-1}$ s $q_x(x): \quad -F_v\langle x-\bar{x}\rangle^{-1}\sin(\theta(\bar{x}))+F_h\langle x-\bar{x}\rangle^{-1}$ c
Koppel	$q_z(x): \quad T\langle x-\bar{x}\rangle^{-2}$ $q_x(x): \quad 0$
Verdeelde belasting	$q_z(x): \quad q_v\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1))-q_v\langle x_2-\bar{x}\rangle^{-}$ $+q_h\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1))-q_h\langle x-\bar{x}_2\rangle^{-}$ $q_x(x): \quad -q_v\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1))+q_v\langle x-\bar{x}_2\rangle^{-}$ $+q_h\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1))-q_h\langle x-\bar{x}_2\rangle^{-}$ Met $\bar{x}_1$ en $\bar{x}_2$ het begin en eind van de verdeelde belasting
Scharnier-/schuif-/telescoop-verbinding	$q_z(x): EI\,\Delta\varphi\langle x-\bar{x}\rangle^{-3} q_z(x): EI\,\Delta u_z\langle x-\bar{x}\rangle^{-4} q_x($
Inklemming	$q_z(x): \quad R_v\langle x-\bar{x}\rangle^{-1}+T\langle x-\bar{x}\rangle^{-2}$ $q_x(x): \quad R_h\langle x-\bar{x}\rangle^{-1}$
Verende oplegging	$q_z(x): \quad F_{\text{veer},h}\langle x-\bar{x}\rangle^{-1}\sin(\theta(\bar{x}))+F_{\text{veer},v}\langle x-$ $q_x(x): \quad R_{\text{veer},h}\langle x-\bar{x}\rangle^{-1}\cos(\theta(\bar{x}))-R_{\text{veer},v}\langle x-$
Verende scharnierverbinding	$q_z(x): \quad EI\,\Delta\varphi\langle x-\bar{x}\rangle^{-3}+T_{\text{veer}}\langle x-\bar{x}\rangle^{-2}$ $EI\,\Delta u_z\langle x-\bar{x}\rangle^{-4}+F_{\text{veer},V}\langle x-\bar{x}\rangle^{-1}$ $q_x(x): \quad EA\,\Delta u_x\langle x-\bar{x}\rangle^{-2}+F_{\text{veer},N}\langle x-\bar{x}\rangle^{-1}$

Situatie op $\bar{x}$	Belastingsfunctie
Aansluitpunt	$q_z(x):\quad \Delta V\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta V\langle x-\bar{x}$ $+\Delta H\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}$ $+\Delta T\langle x-\bar{x}_1\rangle^{-2} - \Delta T\langle x-\bar{x}$ $q_x(x):\quad -\Delta V\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) + \Delta V\langle x-\bar{x}$ $+\Delta H\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}$ met $\bar{x}_1$ de coordinaat van de eerste keer dat langs de vertak
Sprongpunt naar andere vertakking	$q_z(x):\quad \Delta V\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta V\langle x-\bar{x}$ $+\Delta H\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}$ $+\Delta T\langle x-\bar{x}_1\rangle^{-2} - \Delta T\langle x-\bar{x}$ $+EI\Delta\varphi\langle x-\bar{x}_2\rangle^{-3} + EI\Delta u_z\langle x$ $q_x(x):\quad -\Delta V\langle x-\bar{x}_1\rangle^{-1}\sin(\theta(\bar{x}_1)) + \Delta V\langle x-\bar{x}$ $+\Delta H\langle x-\bar{x}_1\rangle^{-1}\cos(\theta(\bar{x}_1)) - \Delta H\langle x-\bar{x}$ $+EI\Delta u_x\langle x$ met $\bar{x}_1$ en $\bar{x}_2$ de coordinaat van de respectievelijk eerste en coordinaat voor de sprong.
Knikpunt	$q_z(x):\quad \sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{F_j}\rangle^0 F_{v,j}\langle x-\bar{x}_i\rangle^{-1}$ $+\sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{F_j}\rangle^0 F_{h,j}\langle x-\bar{x}_i\rangle^{-1}$ $+\sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{q_j}\rangle^0 q_{v,j}\left(\langle x-\bar{x}_i\rangle^0 + \langle\right.$ $+\sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{q_j}\rangle^0 q_{h,j}\left(\langle x-\bar{x}_i\rangle^0 + \langle\right.$ $q_x(x):\quad \sum_j\sum_i^n -\langle\bar{x}_i-\bar{x}_{F_j}\rangle^0 F_{v,j}\langle x-\bar{x}_i\rangle^{-1}$ $+\sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{F_j}\rangle^0 F_{h,j}\langle x-\bar{x}_i\rangle^{-1}$ $+\sum_j\sum_i^n -\langle\bar{x}_i-\bar{x}_{q_j}\rangle^0 q_{v,j}\left(\langle x-\bar{x}_i\rangle^0 + \langle\right.$ $+\sum_j\sum_i^n \langle\bar{x}_i-\bar{x}_{q_j}\rangle^0 q_{h,j}\left(\langle x-\bar{x}_i\rangle^0 + \langle\right.$ met n het aantal knikken met voor elke knik i een bijbehorer

Getting paid for open-source contributions

In case you'd like to contribute to the SymPy code, which can be used to do calculations like this, you can get paid for your programming contribution! The Google Summer of Code initiative is organized by Google, and SymPy participates in it. See instructions here on how to apply.

Justus: Hinges and 2D discrete

Contents

> Macaulay’s methode is een methode om de kracht en doorbuiging eigenschappen van constructies te bepalen aan de hand van de differentiaalvergelijking voor buiging. Met gebruik van singularity functies is het mogelijk discontinue belastingen werkend op een constructie in één vergelijking te schrijven. Op deze wijze is de gehele constructie te beschrijven met behulp van één differentiaalvergelijking. Dit in tegenstelling tot de klassieke integratie methode, waar voor elke discontinuïteit de constructie wordt opgedeeld in verschillende vergelijkingen. De methode van Macaulay biedt als voordeel dat de integratieconstanten gering blijven. Daarnaast wordt de invloed van een desbetreffende kracht op de constructie zichtbaar. Uitbreidingen op de methode zijn er op basis van discontinuïteiten in de buigstijfheid, doorbuiging en rotatie van de constructie. De methode is echter nooit uitgebreid voor tweedimensionale constructies.
>
> In dit onderzoek wordt met behulp van exploratief en toegepast onderzoek een antwoord gezocht op de volgende onderzoeksvraag: Hoe kan de methode van Macaulay voor tweedimensionale constructies worden uitgebreid, en vervolgens worden toegepast?
>
> De differentiaalvergelijking voor extensie blijkt de mogelijkheid te bieden, om op eenzelfde wijze als voor de situatie bij buiging, de methode van macaulay toe te passen voor constructies die normaalkracht bevatten. Axiale krachten worden met singularity functies met de desbetreffende orde geschreven in deze differentiaalvergelijking. Daarnaast worden discontinuïteiten in de axiale verplaatsing van de constructie met een singularity functie met orde -2 geschreven. De betreffende axiale verplaatsing wordt hierin als onbekend verondersteld. De extra voorwaarde die wordt verkregen is dat de normaalkracht ter plaatse van deze discontinuïteit gelijk aan nul dient te zijn. Eveneens als voor de situatie bij buiging is het mogelijk verende

> verbindingen en opleggingen te modelleren. De extra voorwaarde die hiervoor wordt verkregen is gebaseerd op de theorie van lineaire veren.
>
> Vervolgens blijkt dat tweedimensionale constructies beschreven dienen te worden met de differentiaalvergelijkingen voor buiging en extensie. Met een discrete aanpak worden per hoekpunt vier onbekenden geïntroduceerd. Deze onbekenden hebben betrekking op de verplaatsing in axiale richting, de verplaatsing in dwarsrichting (doorbuiging), de normaalkracht en de dwarskracht. Op basis van vier verkregen relaties per hoekpunt is het mogelijk de onbekenden te achterhalen en zo de karakteristieken van de tweedimensionale constructie te bepalen. De relaties hebben betrekking op de wisselwerking tussen de normaalkracht en dwarskracht, en de wisselwerking tussen de axiale verplaatsing en de doorbuiging. Het is te concluderen dat de gevonden methode voor de uitbreiding van Macaulay voor tweedimensionale constructies correct en toepasbaar is. Met betrekking tot effectiviteit is de vergelijking gemaakt met de methode van het klassiek integreren. Op basis van de veronderstelling dat de gevonden methode vier extra onbekenden introduceert per hoekpunt en de klassieke integratiemethode zes, is te concluderen dat deze methode een duidelijk te overwegen alternatief is op de klassieke integratie methode.

van der Wulp (2023)

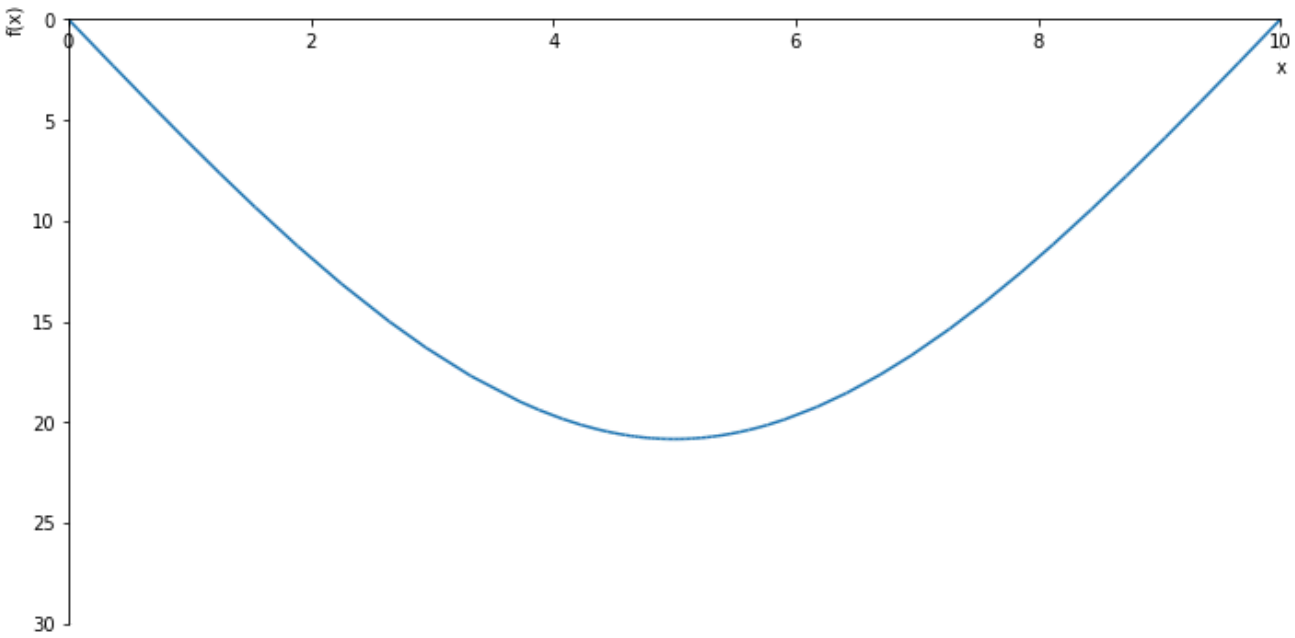
Documenten

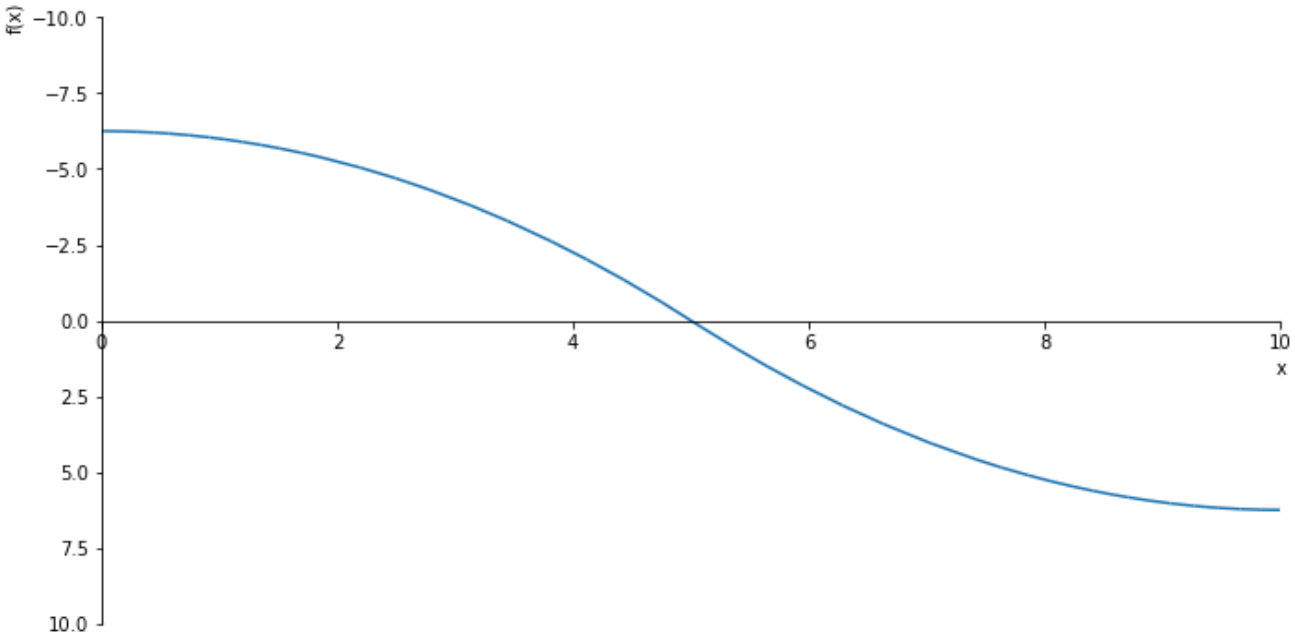
- TU Delft Education repository
- GitHub repository, examples also shown in this book

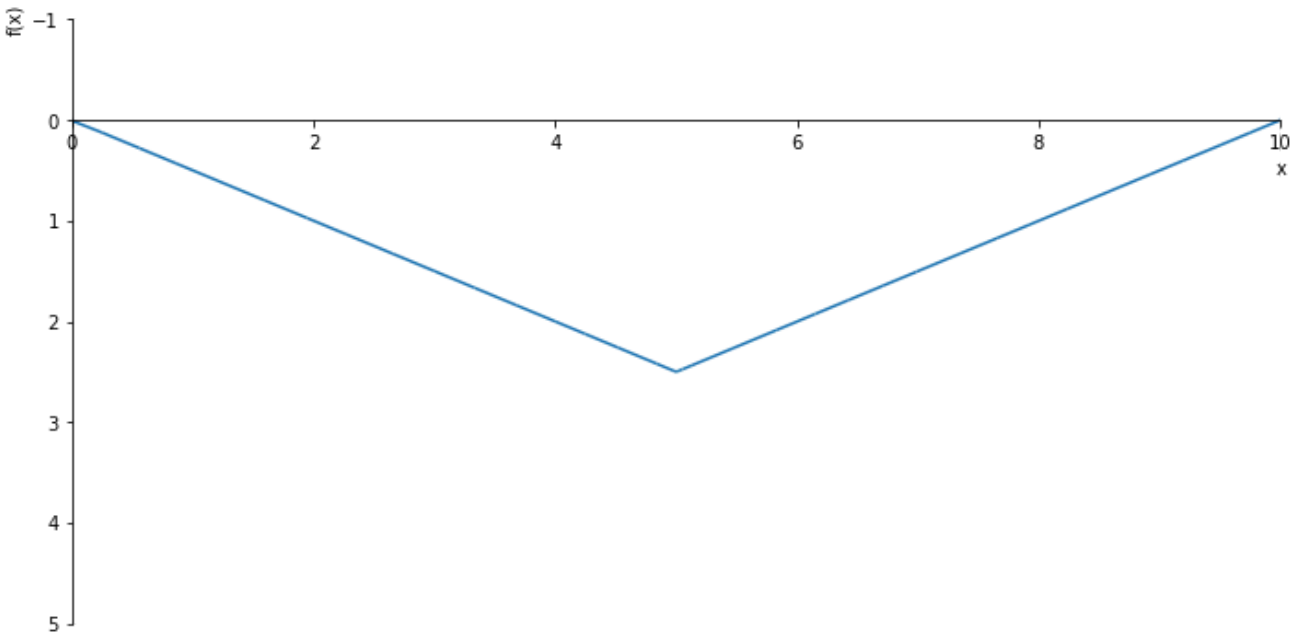
BEP voorbeelden

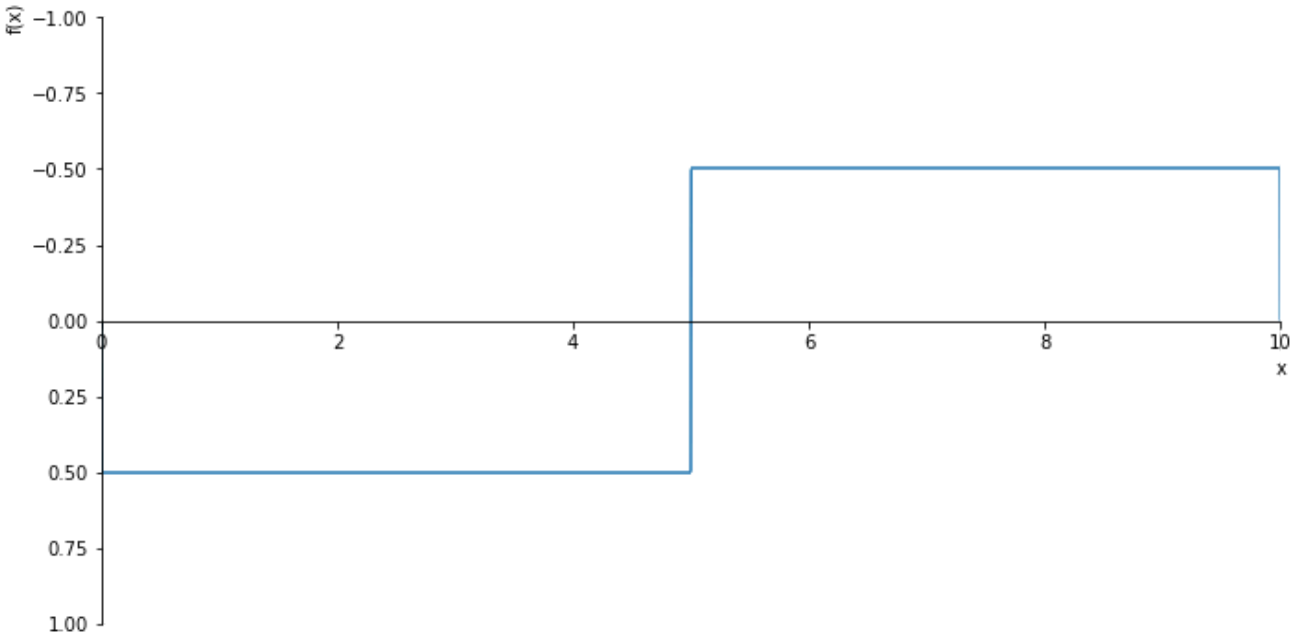
```
import sympy as sp
import numpy as np
from sympy import symbols
E, I = symbols('E, I')
x = symbols('x')
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

```
#Voorbeeld 1
w = -1/12*sf(x, 0, 3) + 1/6*sf(x, 5, 3) - 1/12*sf(x, 10, 3) + 25/4*x
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(M)
sp.plot(w, ylim=(30,0), xlim=(0, 10), size=(10,5))
sp.plot(phi, ylim=(10,-10), xlim=(0, 10), size=(10,5))
sp.plot(M, ylim=(5,-1), xlim=(0, 10), size=(10,5))
sp.plot(V, ylim=(1,-1), xlim=(0, 10), size=(10,5))
```


f(x)
0
5
10
15
20
25
30
0
2
4
6
8
10
x

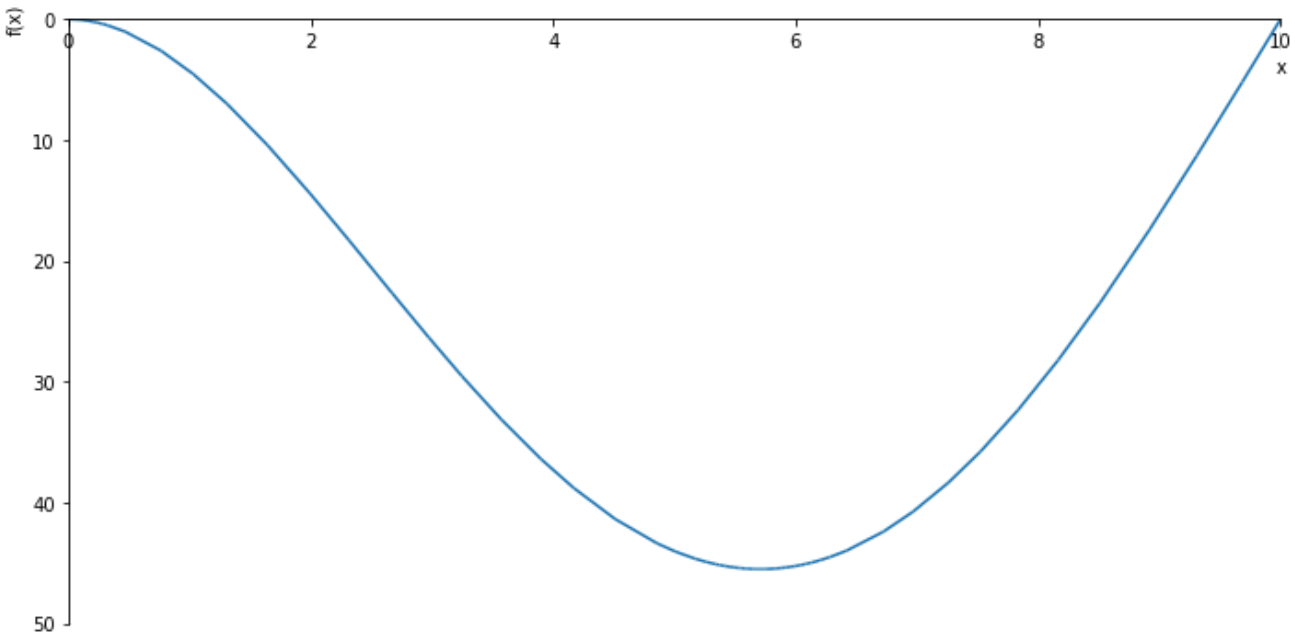

f(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10
x

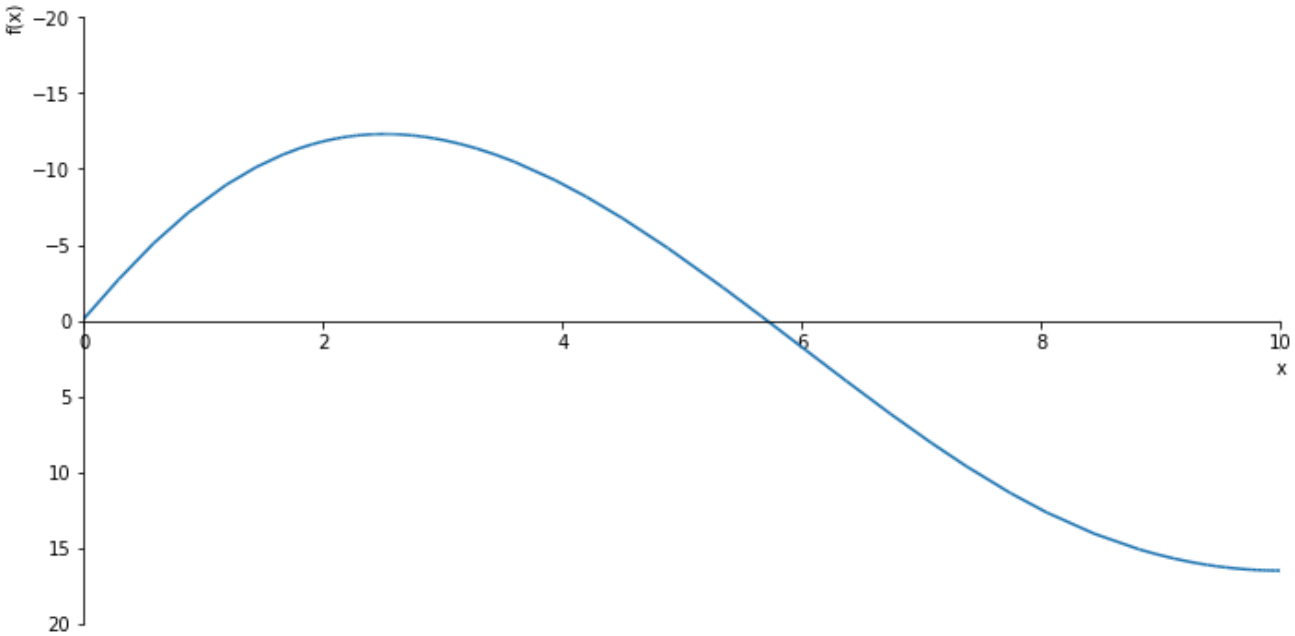


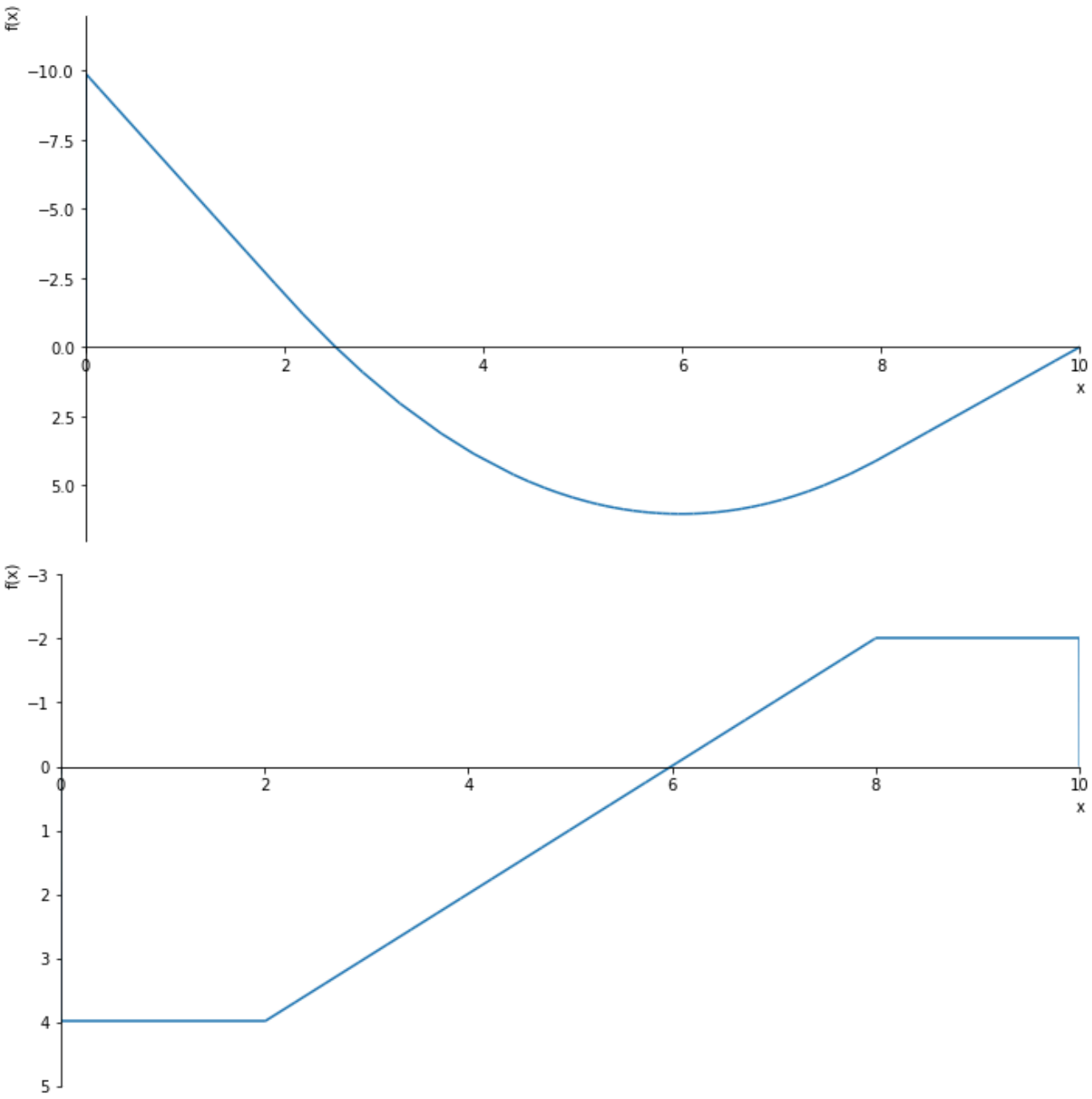



```
<sympy.plotting.plot.Plot at 0x1f3c1113460>
```

```
#Voorbeeld 2
w = 99/20*sf(x, 0, 2) - 399/600*sf(x, 0, 3) + 1/24*sf(x, 2, 4) - 1/24*sf(x, 8, 4) - 201/6
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(M)
sp.plot(w, ylim=(50,0), xlim=(0, 10), size=(10,5))
sp.plot(phi, ylim=(20,-20), xlim=(0, 10), size=(10,5))
sp.plot(M, ylim=(7,-12), xlim=(0, 10), size=(10,5))
sp.plot(V, ylim=(5,-3), xlim=(0, 10), size=(10,5))
```

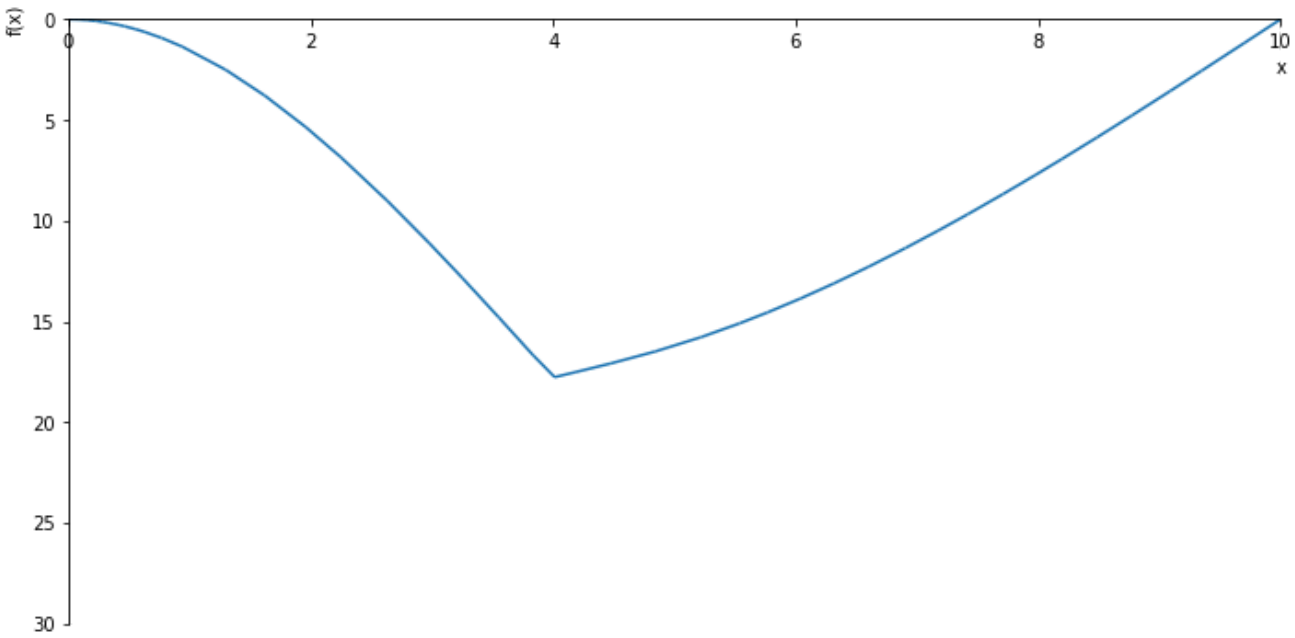

f(x)
0
10
20
30
40
50
0
2
4
6
8
10
x

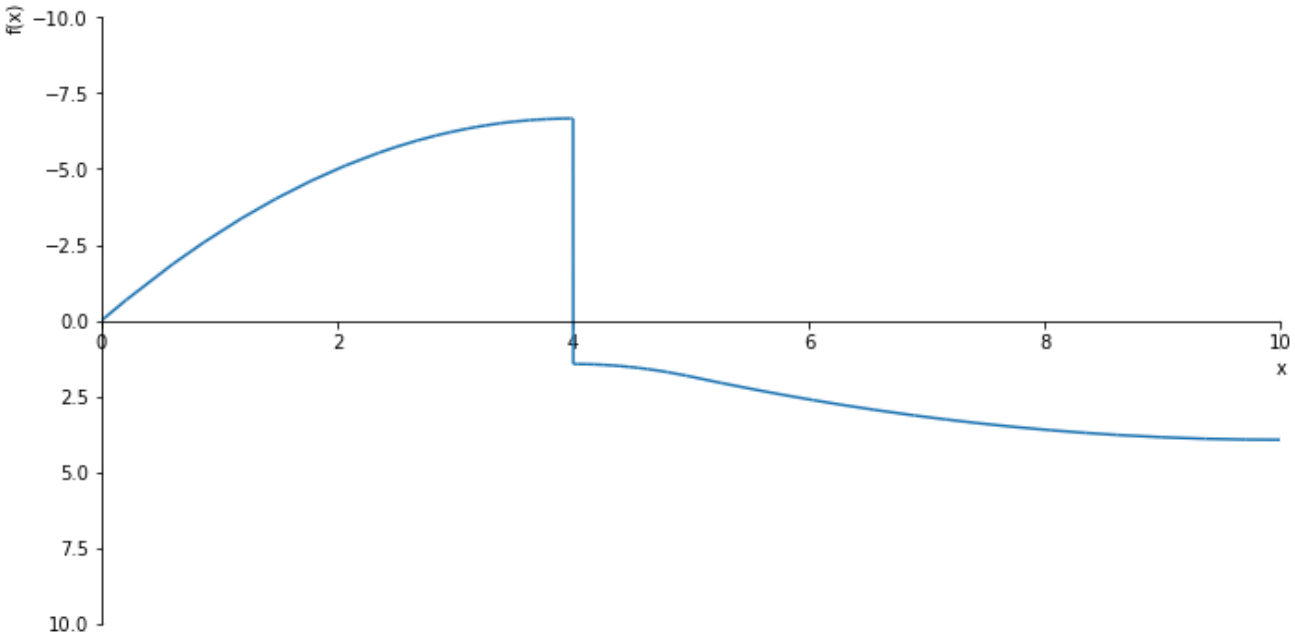

f(x)
−20
−15
−10
−5
0
5
10
15
20
0
2
4
6
8
10
x

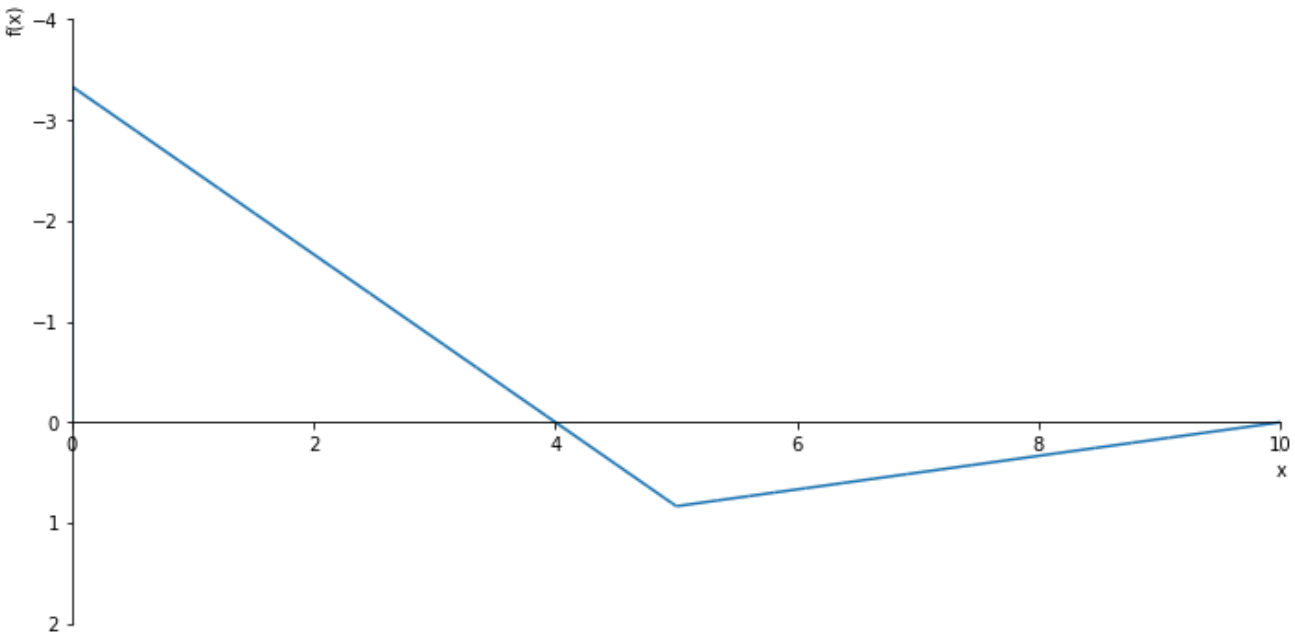


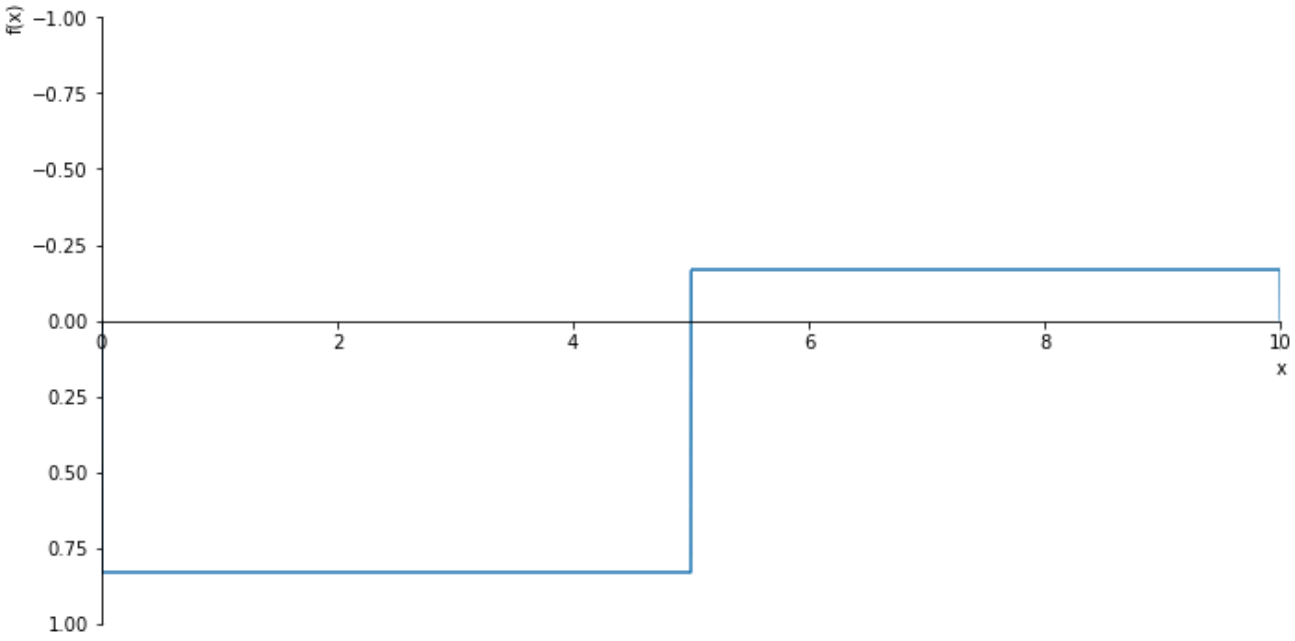

```
<sympy.plotting.plot.Plot at 0x1f3c0ef02b0>
```

```
#Voorbeeld 3
w = 20/12*sf(x, 0, 2) - 5/36*sf(x, 0, 3) - 875/108*sf(x, 4, 1) + 1/6*sf(x, 5, 3) - 1/36*s
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(M)
sp.plot(w, ylim=(30,0), xlim=(0, 10), size=(10,5))
sp.plot(phi, ylim=(10,-10), xlim=(0, 10), size=(10,5))
sp.plot(M, ylim=(2,-4), xlim=(0, 10), size=(10,5))
sp.plot(V, ylim=(1,-1), xlim=(0, 10), size=(10,5))
```


f(x)
0
5
10
15
20
25
30
0
2
4
6
8
10
x

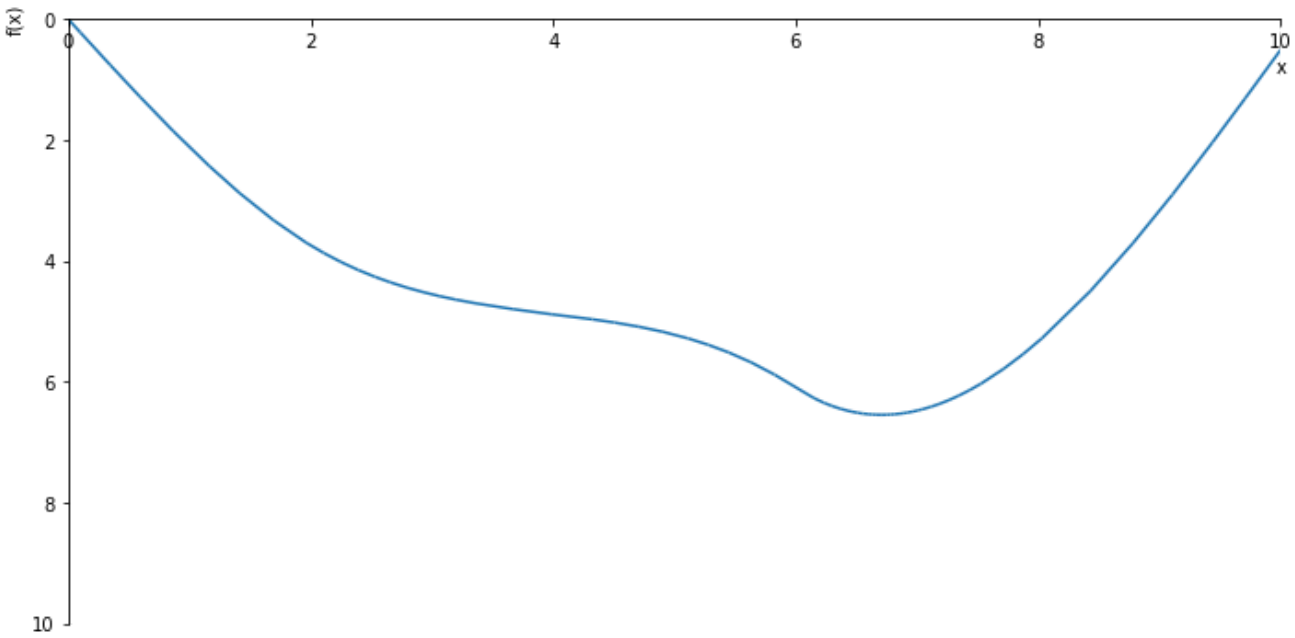

f(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10
x

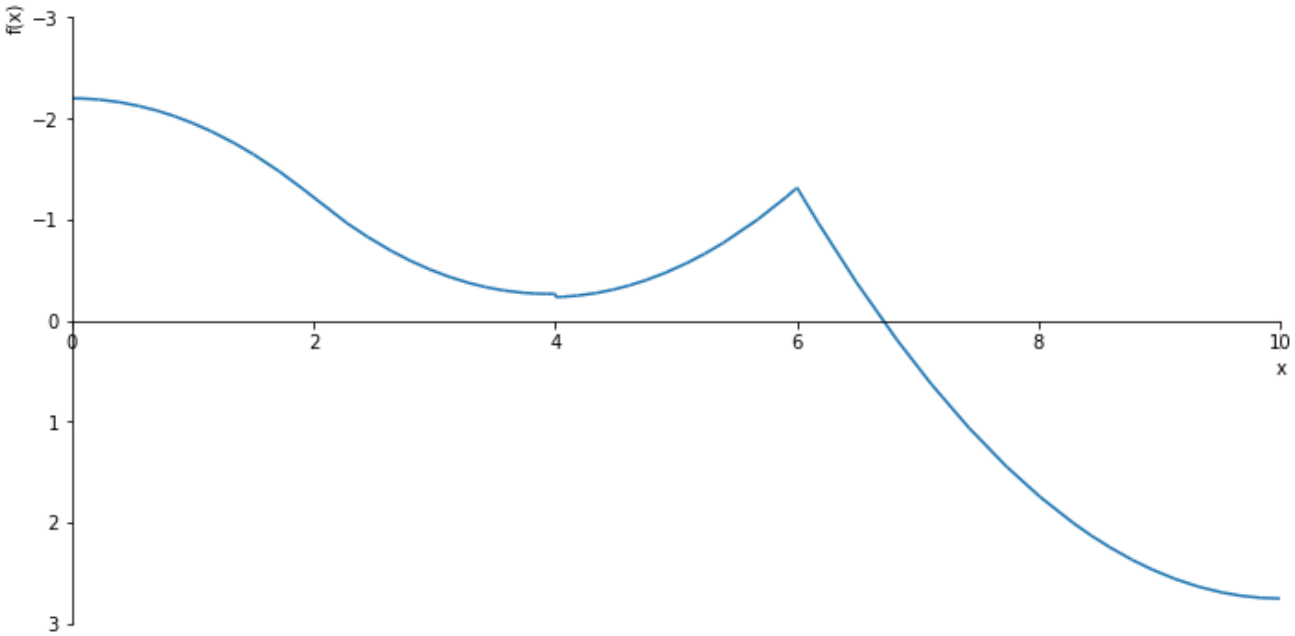


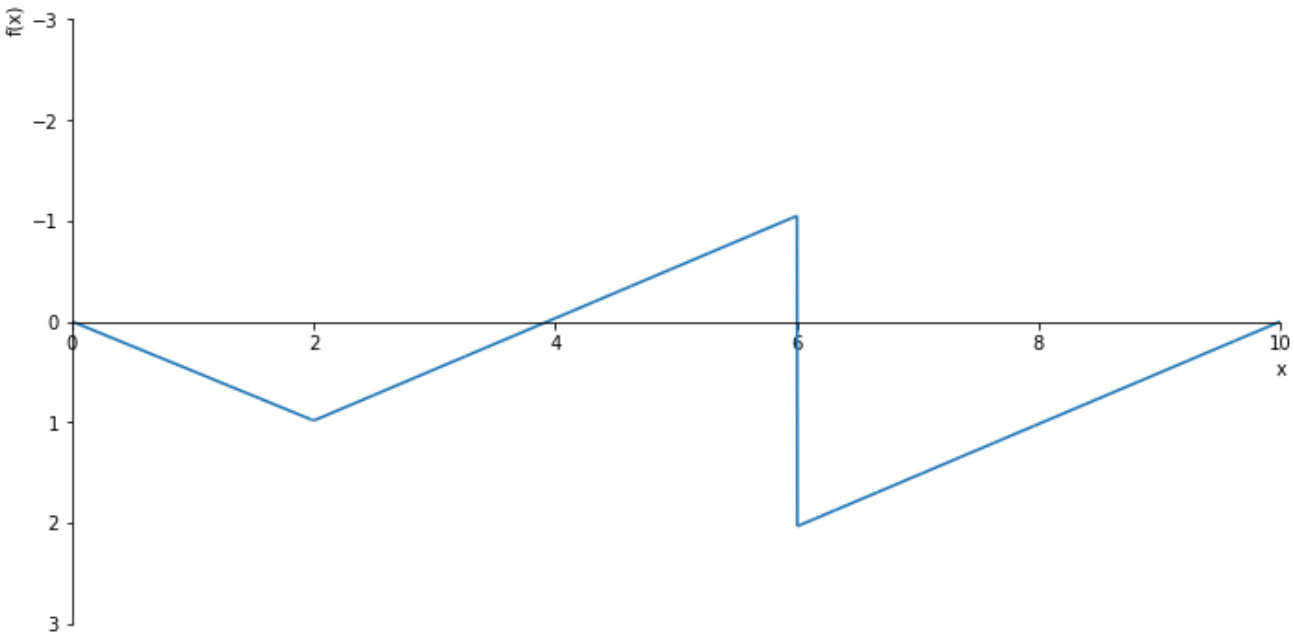


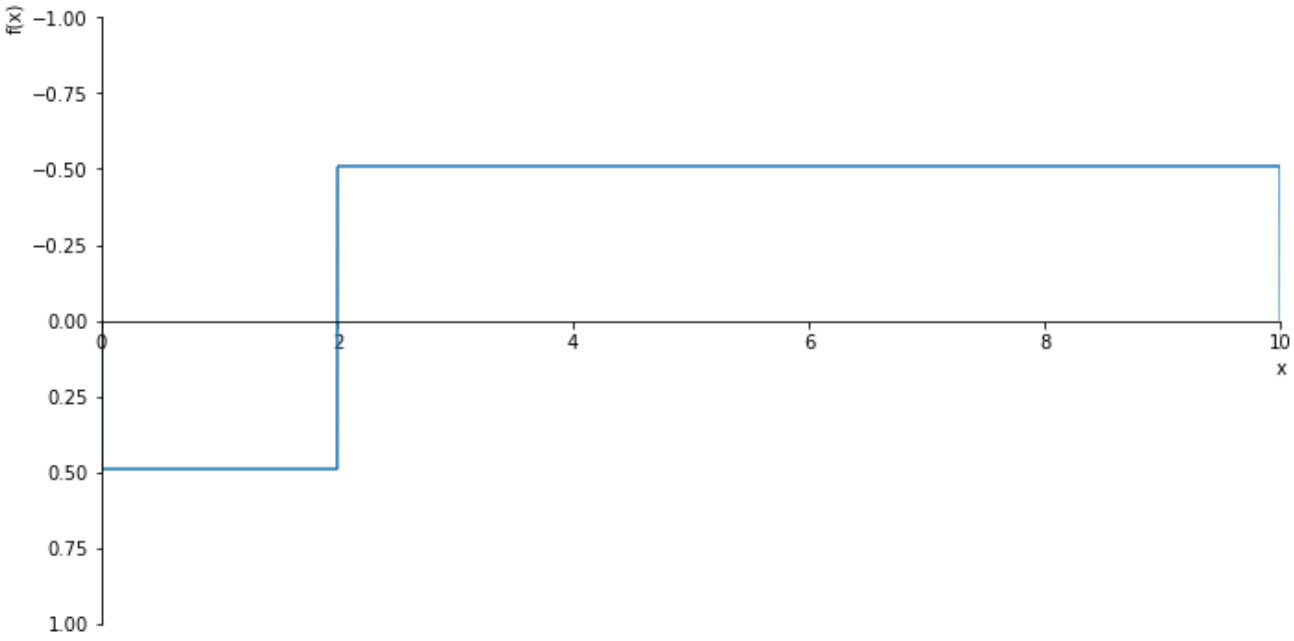

```
<sympy.plotting.plot.Plot at 0x1f3c0e861c0>
```

```
#Voorbeeld 4
w = -89/1086*sf(x, 0, 3) + 1/6*sf(x, 2, 3) - 6/181*sf(x, 4, 1) - 558/362*sf(x, 6, 2) - 92
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(M)
sp.plot(w, ylim=(10,0), xlim=(0, 10), size=(10,5))
sp.plot(phi, ylim=(3,-3), xlim=(0, 10), size=(10,5))
sp.plot(M, ylim=(3,-3), xlim=(0, 10), size=(10,5))
sp.plot(V, ylim=(1,-1), xlim=(0, 10), size=(10,5))
```


f(x)
0
2
4
6
8
10
x
0
2
4
6
8
10

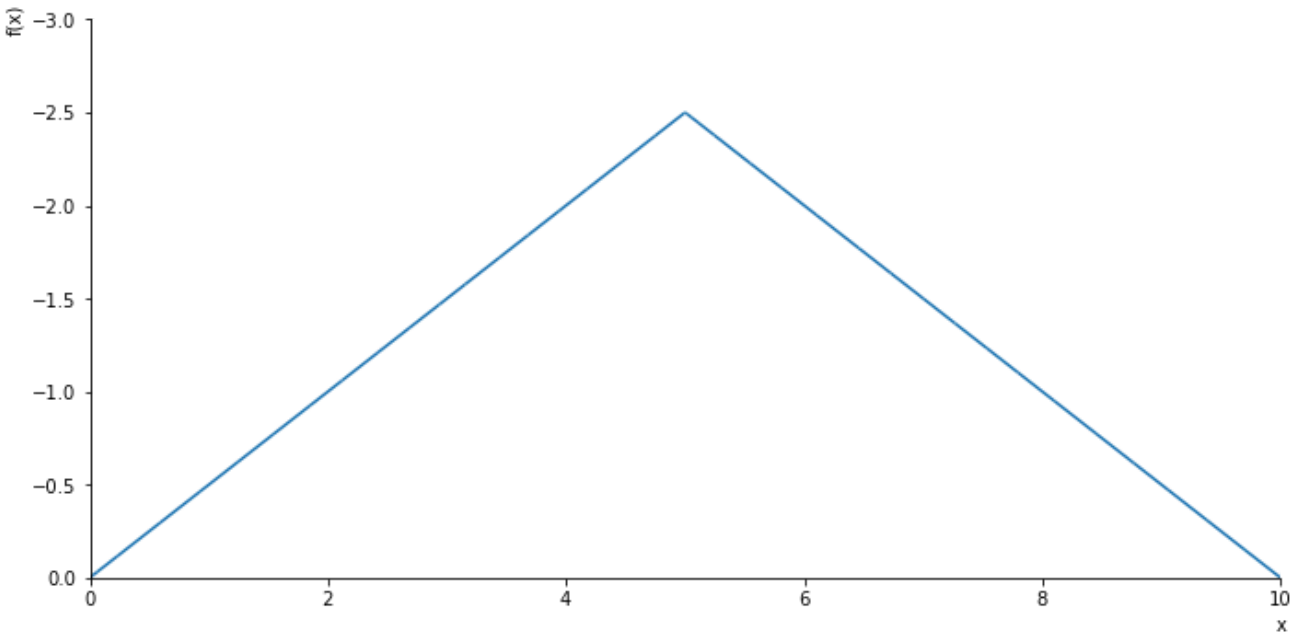

f(x)
−3
−2
−1
0
1
2
3
0
2
4
6
8
10
x

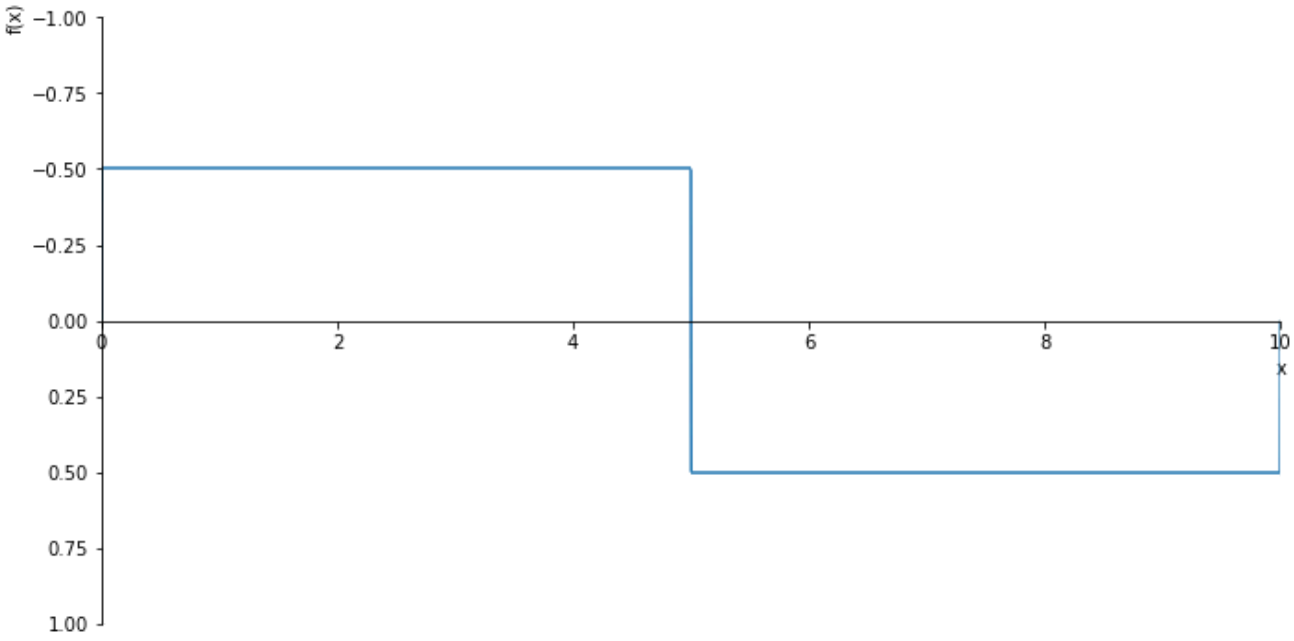





```
<sympy.plotting.plot.Plot at 0x1f3c216d460>
```

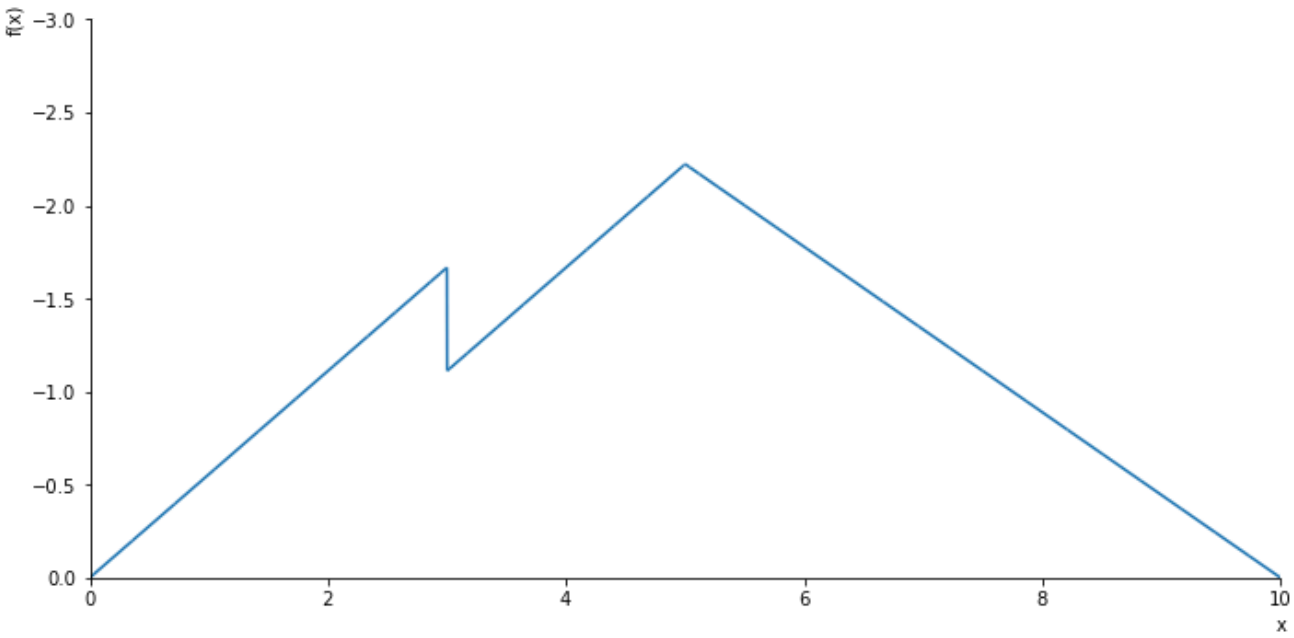
```
#Voorbeeld 5
u = -1/2*sf(x, 0, 1) + 1*sf(x, 5, 1) - 1/2*sf(x, 10, 1)
N = sp.diff(u)
sp.plot(u, ylim=(0, -3), xlim=(0, 10), size=(10,5))
sp.plot(N, ylim=(1,-1), xlim=(0, 10), size=(10,5))
```

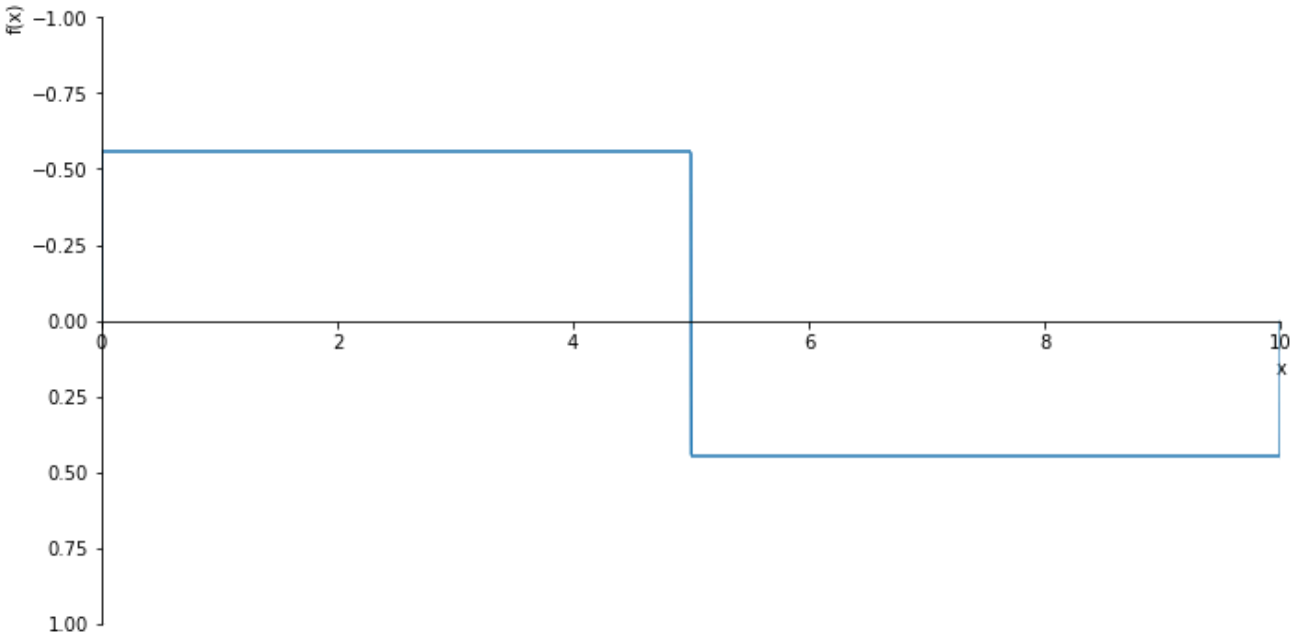





```
<sympy.plotting.plot.Plot at 0x1f3c1079130>
```

```
#Voorbeeld 6
u = -5/9*sf(x, 0, 1) + 5/9*sf(x, 3, 0) + 1*sf(x, 5, 1) - 4/9*sf(x, 10, 1)
N = sp.diff(u)
sp.plot(u, ylim=(0, -3), xlim=(0, 10), size=(10,5))
sp.plot(N, ylim=(1,-1), xlim=(0, 10), size=(10,5))
```





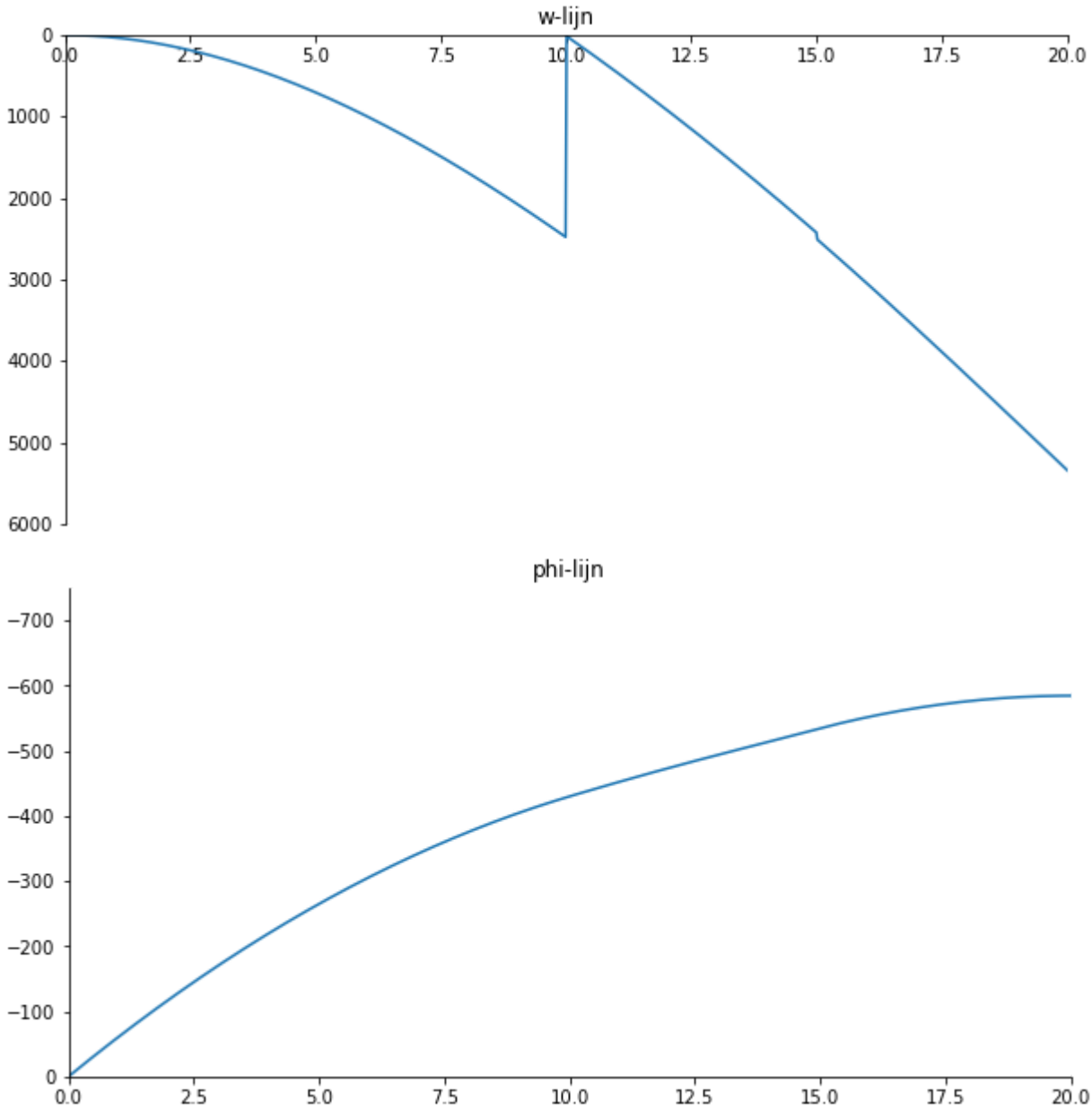

```
<sympy.plotting.plot.Plot at 0x1f3c10c91c0>
```

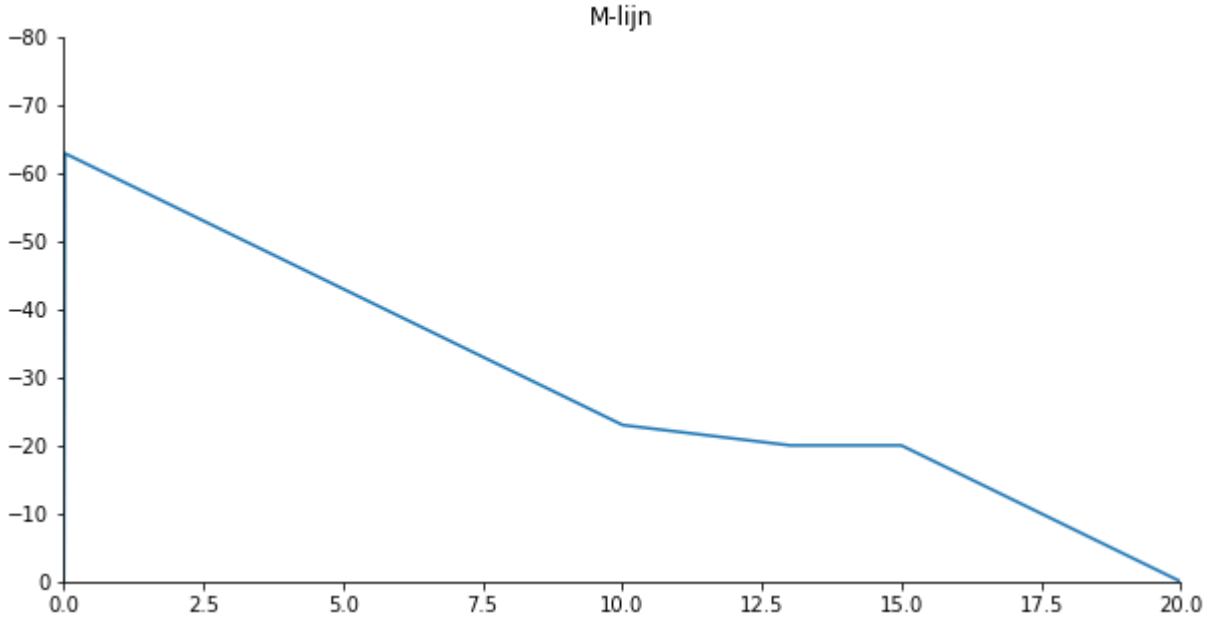
```
#Voorbeeld 7
x = symbols('x')
w = 63/2*sf(x, 0, 2) - 4/6*sf(x, 0, 3) + 3/6*sf(x, 10, 3) - 7420/3*sf(x, 10, 0) + 1/6*sf(
wV = 63/2*sf(x, 0, 2) - 4/6*sf(x, 0, 3) + 3/6*sf(x, 10, 3) + 1/6*sf(x, 13, 3) - 4/6*sf(x,
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(wV, x, 3)

w_numpy = sp.lambdify(x, w.rewrite(sp.Piecewise))
phi_numpy = sp.lambdify(x, phi.rewrite(sp.Piecewise))
M_numpy = sp.lambdify(x, M.rewrite(sp.Piecewise))
V_numpy = sp.lambdify(x, V.rewrite(sp.Piecewise))

u = 1*sf(x, 0, 1) - 5*sf(x, 10, 1) - 7480/3*sf(x, 10, 0) + 4*sf(x, 15, 1) + 14794/3*sf(x,
N = sp.diff(u)

u_numpy = sp.lambdify(x, u.rewrite(sp.Piecewise))
N_numpy = sp.lambdify(x, N.rewrite(sp.Piecewise))
```

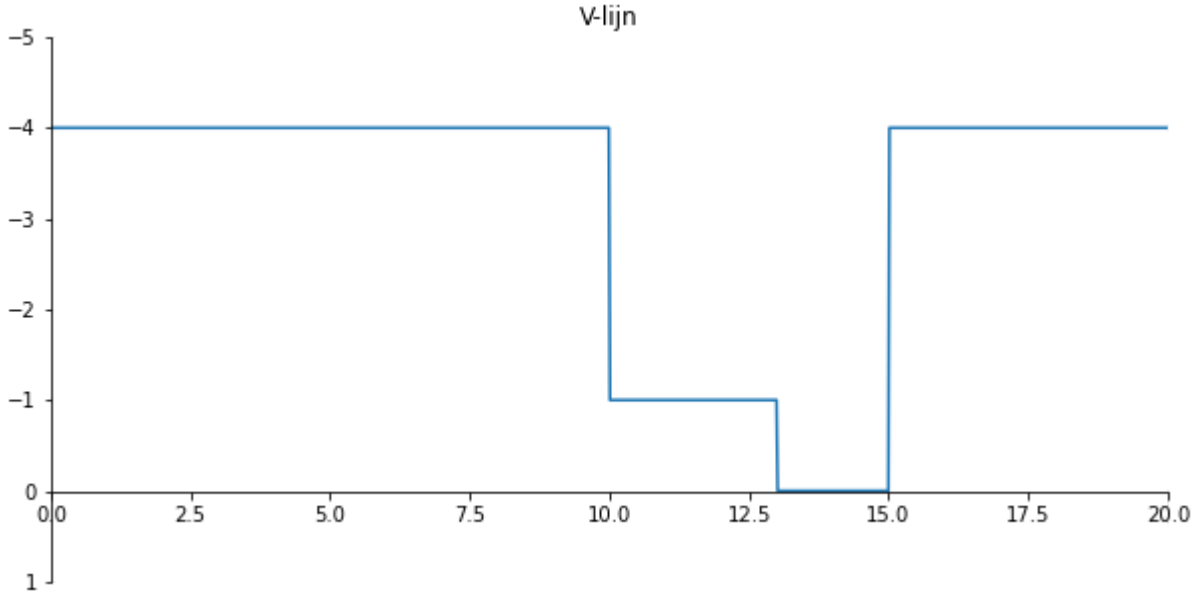
```
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), w_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([6000, 0])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), phi_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([0, -750])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), M_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([0, -80])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), V_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1, -5])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('V-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), u_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([4000, -4000])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), N_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
```

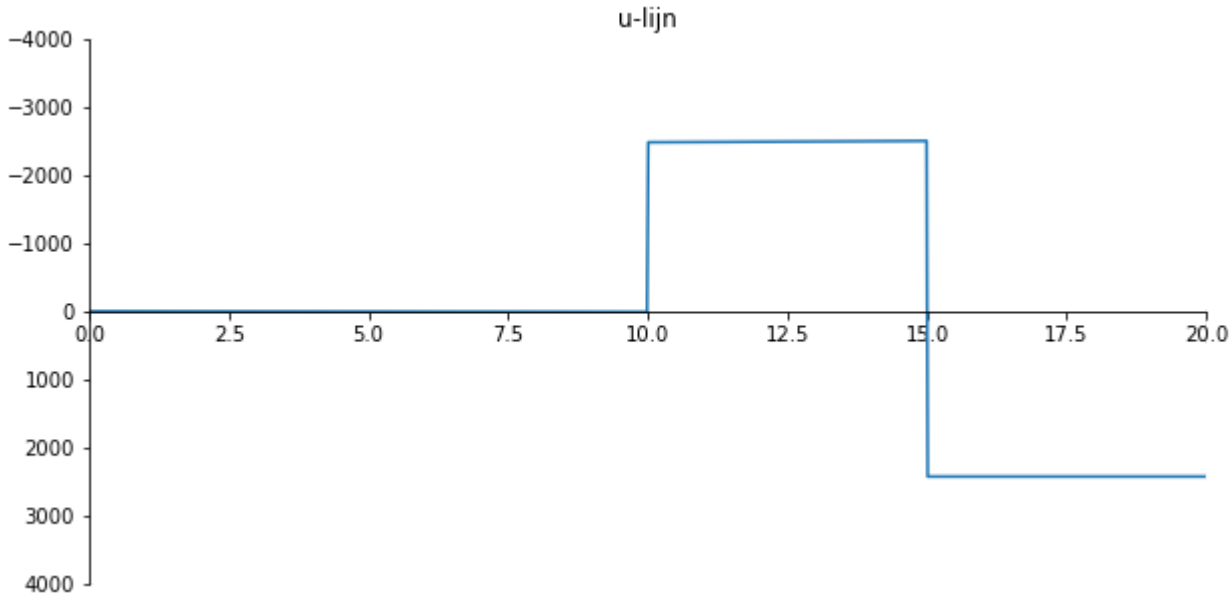
```
plt.ylim([2, -6])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')
```

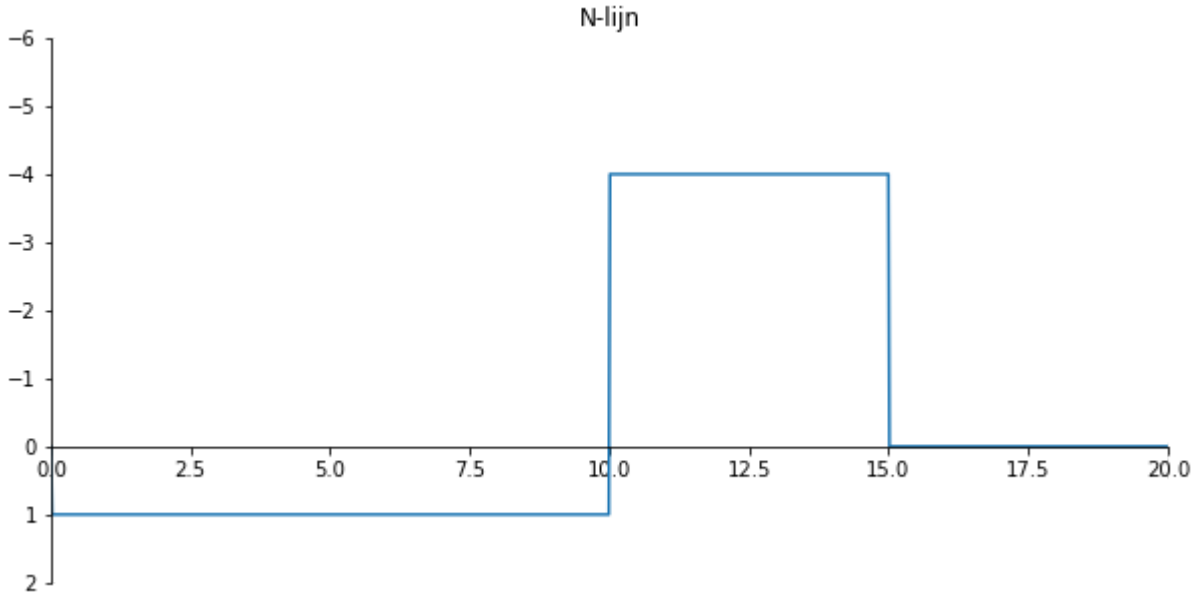
```
Text(0.5, 1.0, 'N-lijn')
```




M-lijn
−80
−70
−60
−50
−40
−30
−20
−10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


V-lijn
−5
−4
−3
−2
−1
0
1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



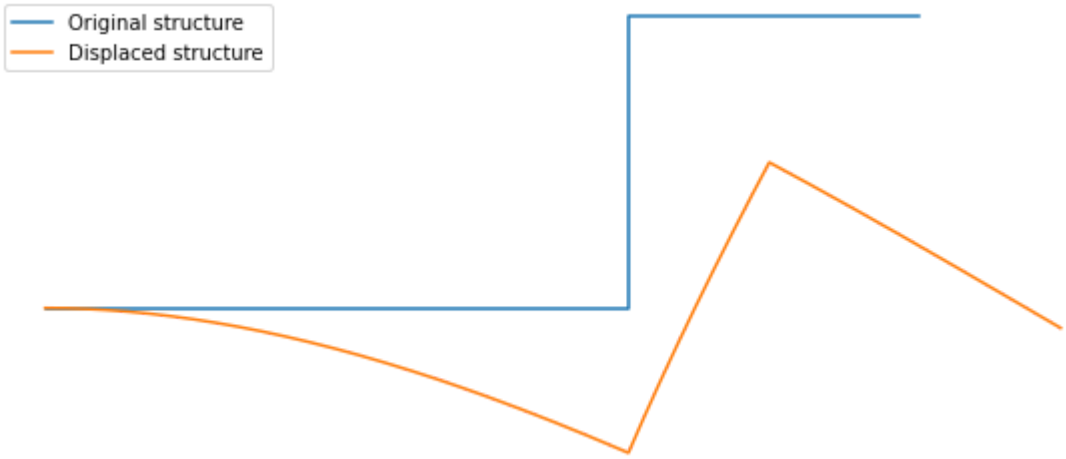



```
factor = sp.symbols('factor')
w = (63/2*sf(x, 0, 2) - 4/6*sf(x, 0, 3) + 3/6*sf(x, 10, 3) - 7420/3*sf(x, 10, 0) + 1/6*sf
u = (1*sf(x, 0, 1) - 5*sf(x, 10, 1) - 7480/3*sf(x, 10, 0) + 4*sf(x, 15, 1) + 14794/3*sf(x

u_test = u.subs(factor, 1000)
w_test = w.subs(factor, 1000)

x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * x
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * 10
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * (-x+
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (-5
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)
display(w_test.subs({x:20}))
display(u_test.subs({x:20}))
```

$$5.3425$$

```
x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * u_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * w_te
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * u_t
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * w_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * -u_t
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * w_t
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)
```

```
plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)),z_structure_numpy(np.linspace(0,20,101)
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend();
```

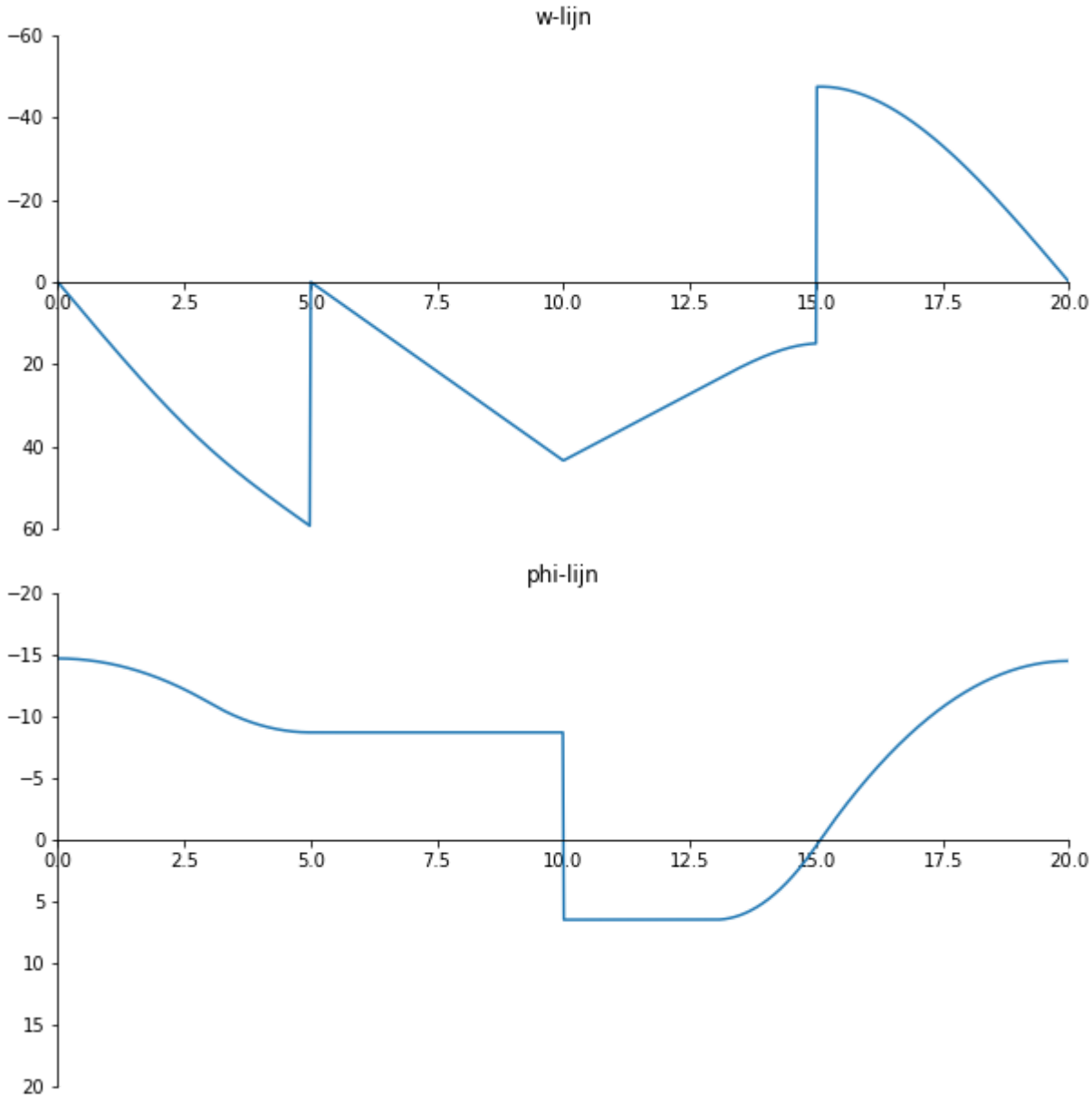


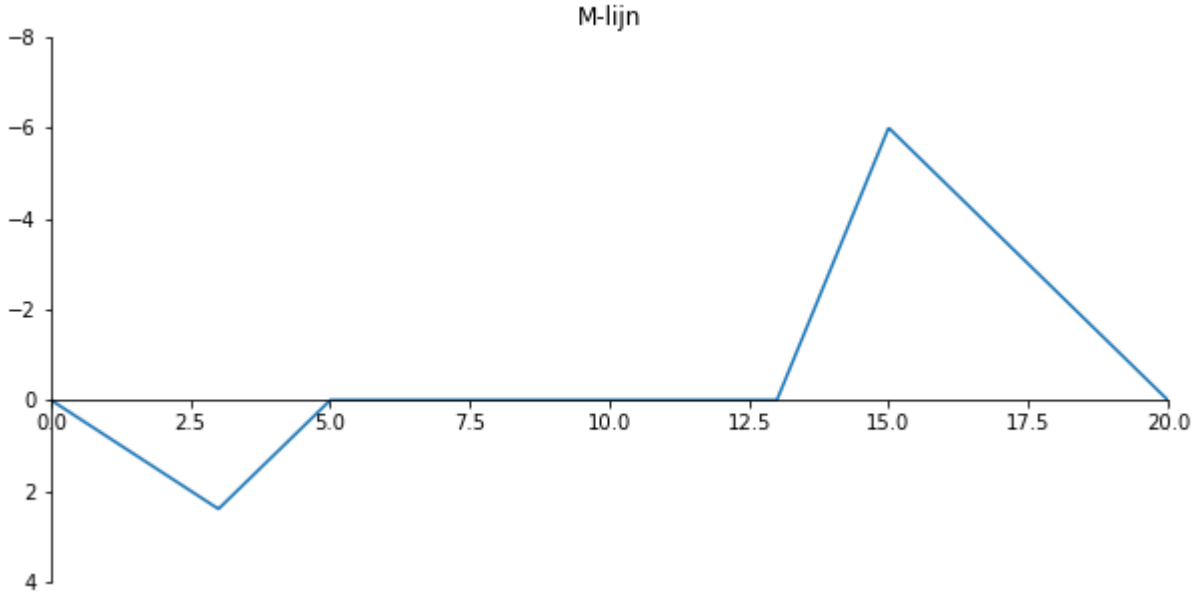

```
#Voorbeeld 8
x = symbols('x')
w = -2/15*sf(x, 0, 3) + 1/3*sf(x, 3, 3) - 1/5*sf(x, 5, 3) - 119/2*sf(x, 5, 0) - 76/5*sf(x
wV = -2/15*sf(x, 0, 3) + 1/3*sf(x, 3, 3) - 1/5*sf(x, 5, 3) - 76/5*sf(x, 10, 1) + 3/6*sf(x
phi = sp.diff(w)*-1
M = sp.diff(phi)
V = sp.diff(wV, x, 3)

w_numpy = sp.lambdify(x, w.rewrite(sp.Piecewise))
phi_numpy = sp.lambdify(x, phi.rewrite(sp.Piecewise))
M_numpy = sp.lambdify(x, M.rewrite(sp.Piecewise))
V_numpy = sp.lambdify(x, V.rewrite(sp.Piecewise))

u = -6/5*sf(x, 5, 1) + 119/2*sf(x, 5, 0) - 9/5*sf(x, 15, 1) - 65/2*sf(x, 15, 0) + 3*sf(x,
N = sp.diff(u)

u_numpy = sp.lambdify(x, u.rewrite(sp.Piecewise))
N_numpy = sp.lambdify(x, N.rewrite(sp.Piecewise))
```

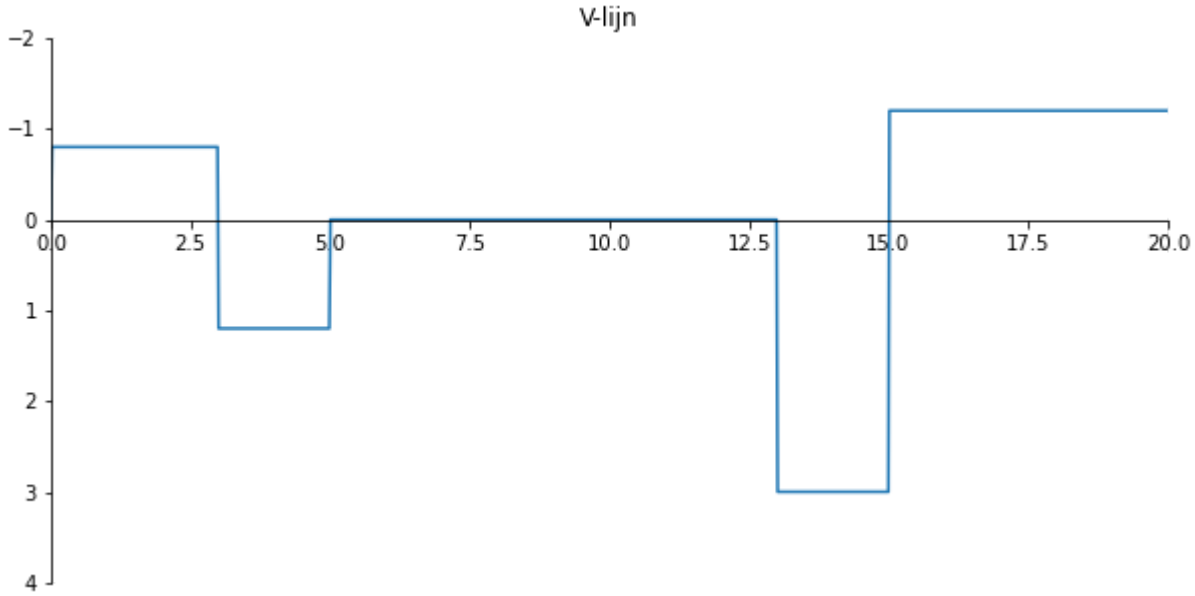
```
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), w_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([60, -60])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), phi_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([20, -20])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), M_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([4, -8])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), V_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([4, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('V-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), u_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([100, -100])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), N_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
```

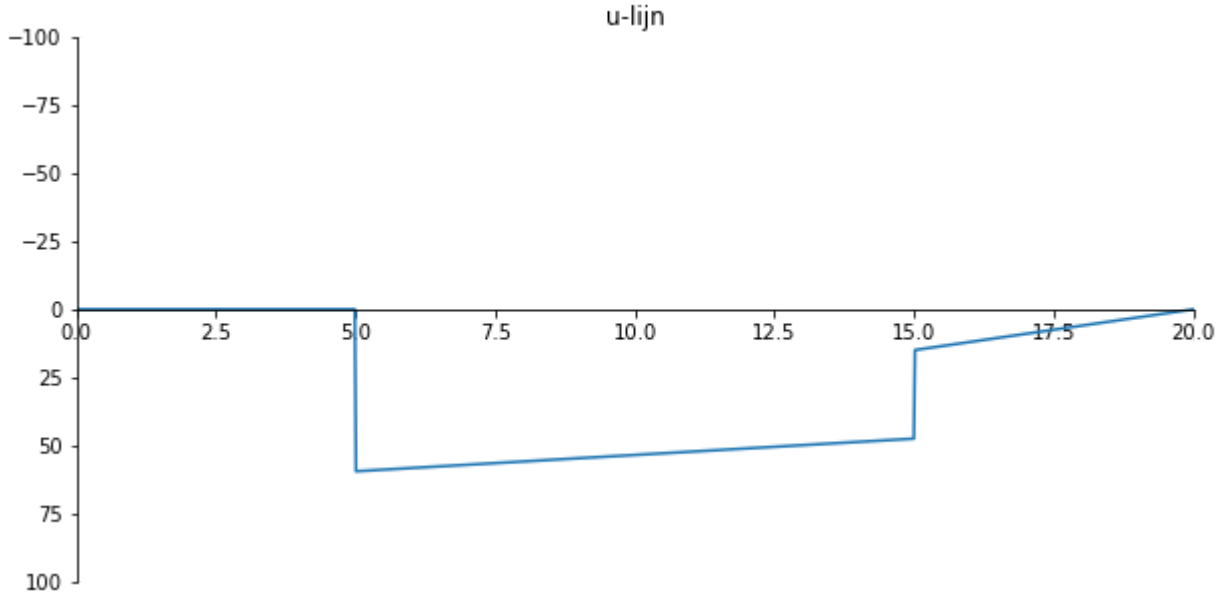
```
plt.ylim([2, -6])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')
```

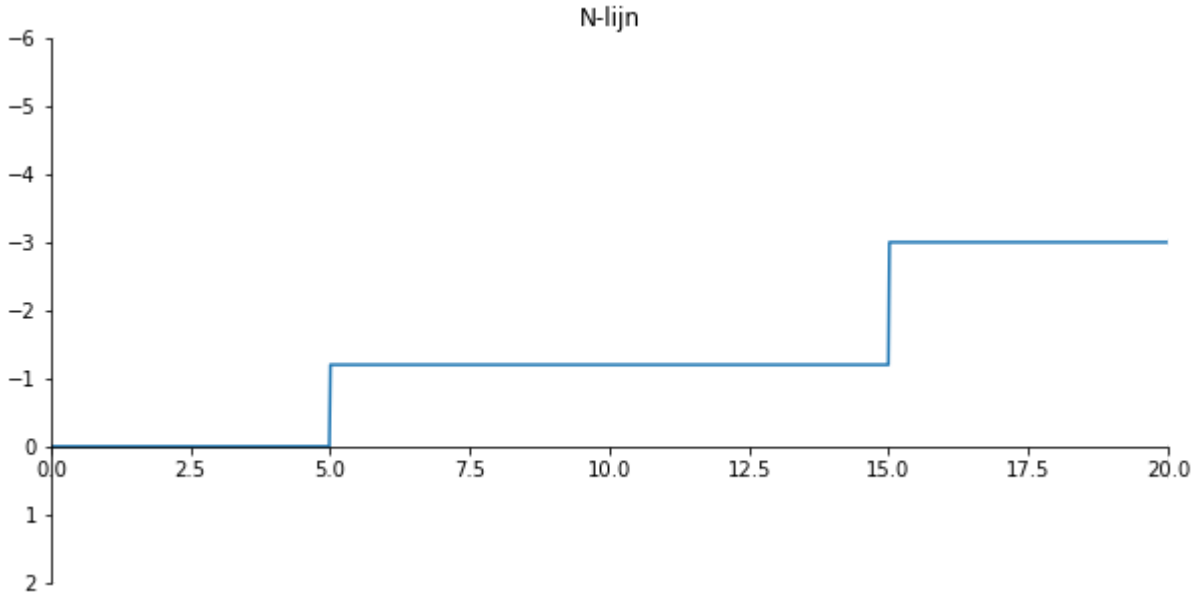
```
Text(0.5, 1.0, 'N-lijn')
```




M-lijn
−8
−6
−4
−2
0
2
4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


V-lijn
−2
−1
0
1
2
3
4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

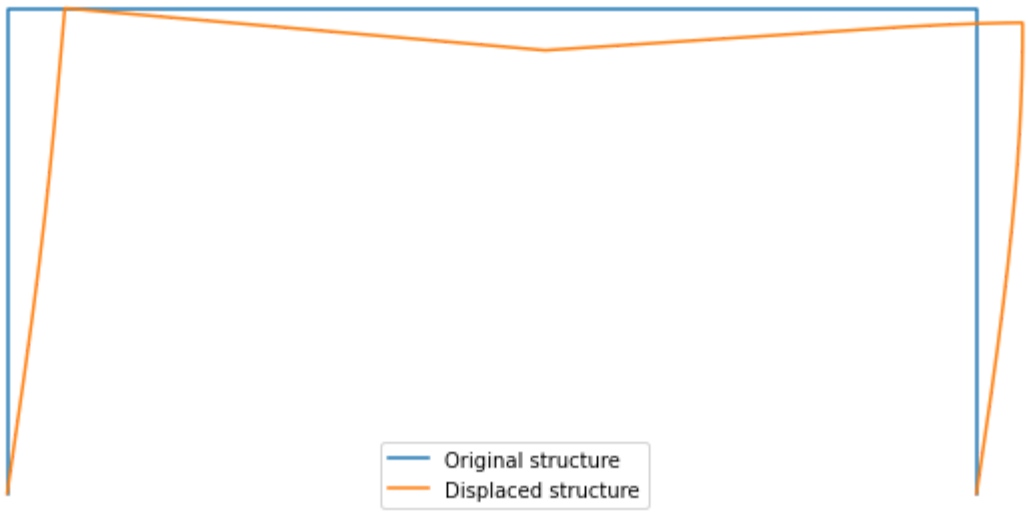





```
factor = sp.symbols('factor')
w = (-2/15*sf(x, 0, 3) + 1/3*sf(x, 3, 3) - 1/5*sf(x, 5, 3) - 119/2*sf(x, 5, 0) - 76/5*sf(
u = (-6/5*sf(x, 5, 1) + 119/2*sf(x, 5, 0) - 9/5*sf(x, 15, 1) - 65/2*sf(x, 15, 0) + 3*sf(x

u_test = u.subs(factor, 100)
w_test = w.subs(factor, 100)

x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * 0
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * (x-5)
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (10
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * -x
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * -5
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)
display(w_test.subs({x:20}))
display(u_test.subs({x:20}))
```

```
x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * w_test
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * u_tes
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * -w_
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * -u_test
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * w_tes
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * u_t
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)
```

```
plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)),z_structure_numpy(np.linspace(0,20,101)
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend();
```



Julia: Influence lines

Contents

De methode van Macaulay is een methode waarmee interne krachten voor een constructie met belastingen kan worden bepaald. Macaulay maakt gebruik van singulariteitsfuncties om discontinue belastingen op constructies overzichtelijk te beschrijven. De methode van Macaulay is zowel voor eendimensionale als tweedimensionale constructies toepasbaar. Invloedslijnen worden toegepast in de mechanica om de meest ongunstige posities van krachten te vinden. Echter is er nog geen onderzoek verricht naar de toepassing van Macaulay's methode met invloedslijnen. Daarom is er gekeken naar hoe deze invloedslijnen kunnen worden gecombineerd met de methode van Macaulay om het bepalen van invloedslijnen overzichtelijker te maken. De onderzoeksvraag van dit rapport luidt: *Hoe kan Macaulay's methode worden toegepast met invloedslijnen voor zowel eendimensionale als tweedimensionale constructies?*

Voor het toepassen van de methode van Macaulay met invloedslijnen op eendimensionale constructies, wordt er gekeken in het bekende xzassenstelsel. De bewegende puntlast wordt hierbij aangenomen op een onbekende positie op $x = a$ zoals weergegeven in Fig. 24. De krachtsverdeling wordt hierbij omschreven aan de hand van singulariteits functies met de desbetreffende orde. Door middel van integreren worden vergelijkingen verkregen die zowel afhangen van x als a. Daarom kan met deze methode in één vergelijking op een overzichtelijke manier zowel alle interne krachten (waarbij a een gespecificeerde waarde heeft) als alle invloedsfactoren (waarbij x een gespecificeerde waarde heeft) worden bepaald.

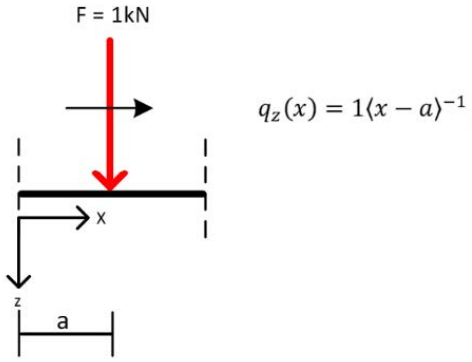


Fig. 24 Methode voor eendimensionaal

De krachtsverdeling bij tweedimensionale constructies wordt beschreven aan de hand van de differentiaalvergelijkingen voor extensie en buiging. Er wordt hierbij gekeken in een nieuw geïntroduceerd st-assenstelsel. De bewegende eenheidslast wordt beschreven als een kracht die altijd verticaal naar beneden gericht is en bevindt zich op een onbekende positie $s = a$. De kracht wordt hierbij in beide richtingen beschreven door het implementeren van goniometrische verhoudingen. Deze vergelijkingen zijn afhankelijk van de hoek α die de constructie met de horizontale lijn maakt zoals geschematiseerd in Fig. 25.

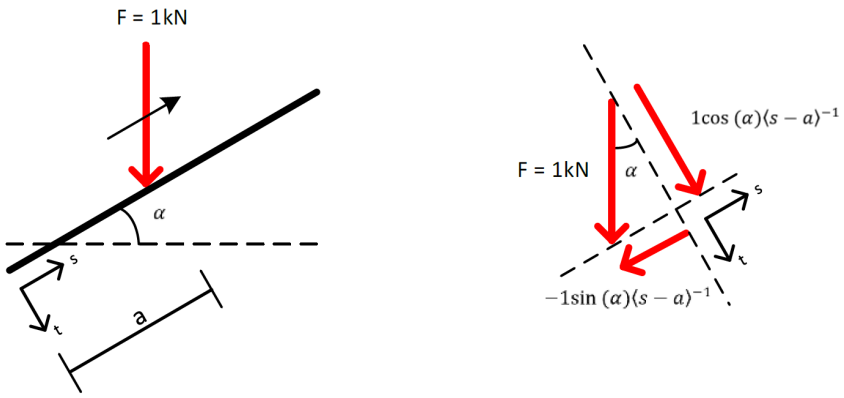


Fig. 25 Methode voor tweedimensionaal

Met deze methode worden vergelijkingen verkregen die zowel afhangen van s als a. Hierdoor kunnenook voor tweedimensionale constructies alle interne krachten en invloedsfactoren overzichtelijk worden bepaald.

Hieruit valt te concluderen dat de bedachte methode toepasbaar is voor het bepalen van invloedslijnen met de methode voor Macaulay. Deze methode is zowel voor eendimensionale als tweedimensionale constructies toepasbaar. De bedachte methode is overzichtelijk aangezien er met één vergelijking alle mogelijke scenario's kunnen worden bekeken voor zowel interne krachten als invloedsfactoren. Deze methode kan

> vervolgens worden toegepast bij het bepalen van de meest ongunstige posities van krachten in constructies wat van belang is bij het ontwerpen van civiele projecten.

Jankie (2023)

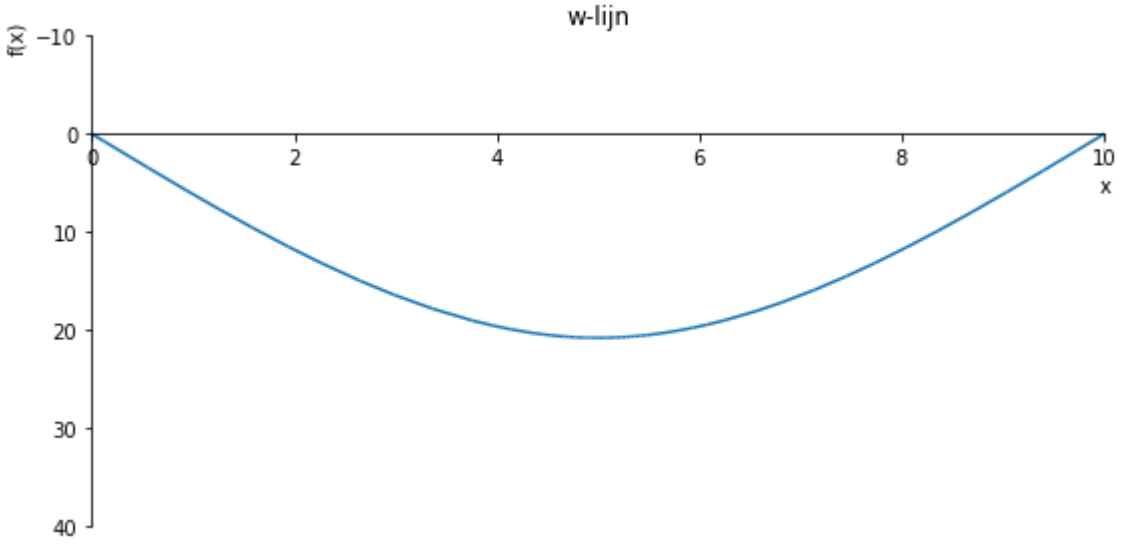
Documenten

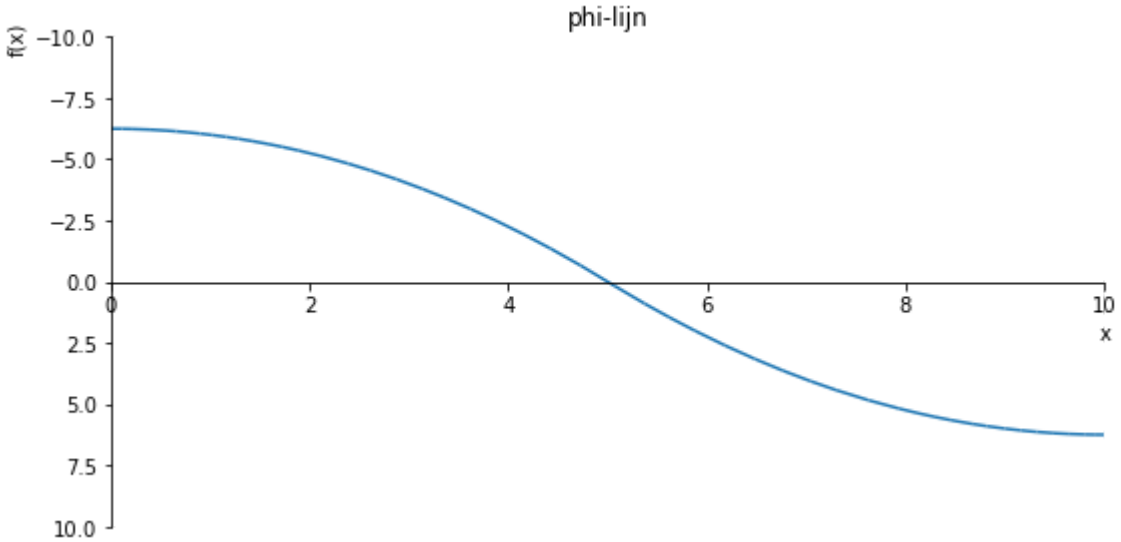
- TU Delft Education repository
- GitHub repository, examples also shown in this book: example 1 and example 2

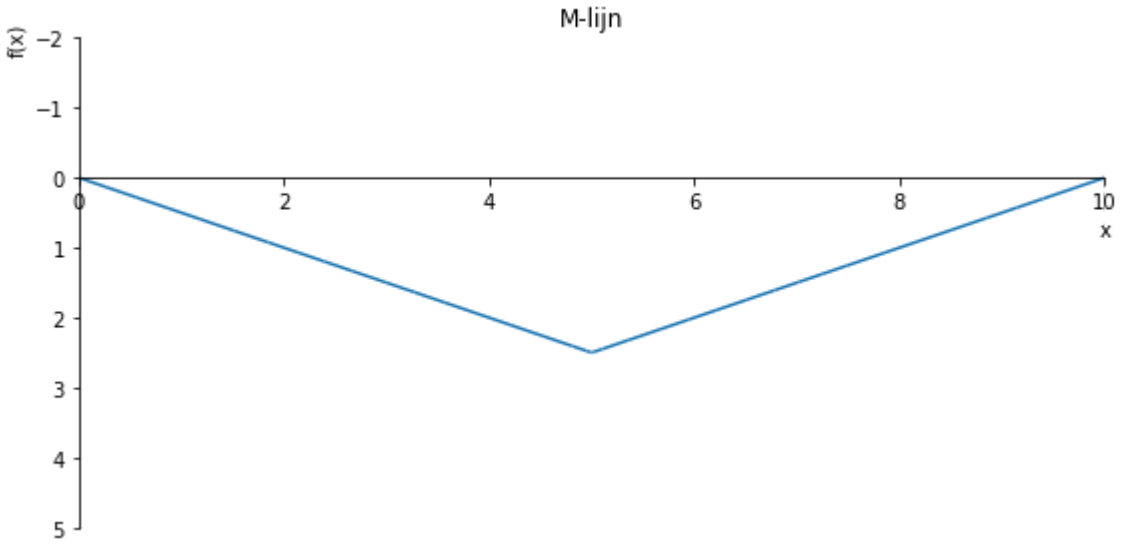
Eendimensionale voorbeelden

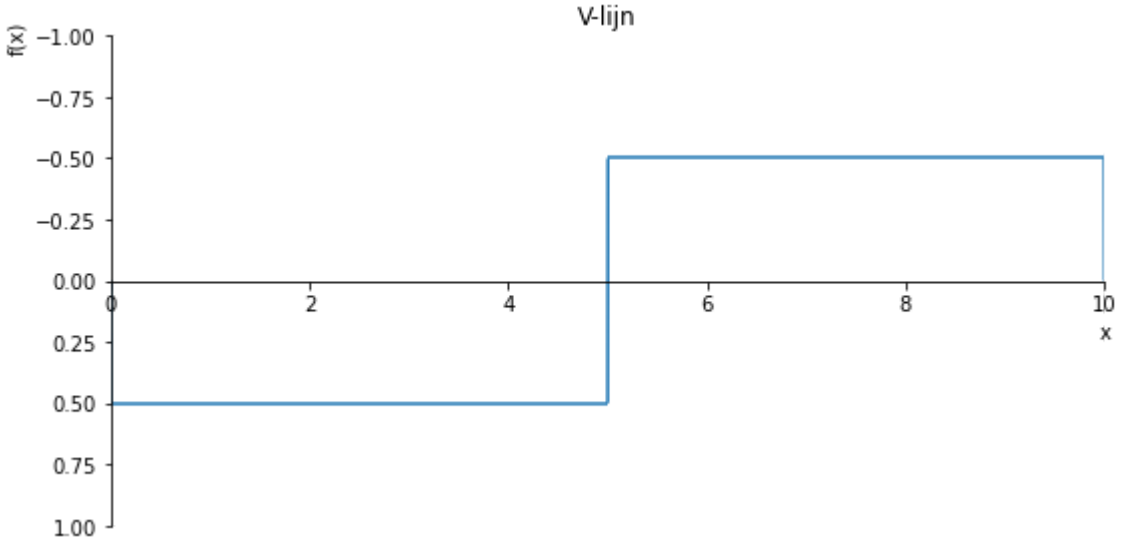
```
import sympy as sp
import numpy as np
from sympy import symbols
E, I = symbols('E, I')
x = symbols('x')
a = symbols('a')
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

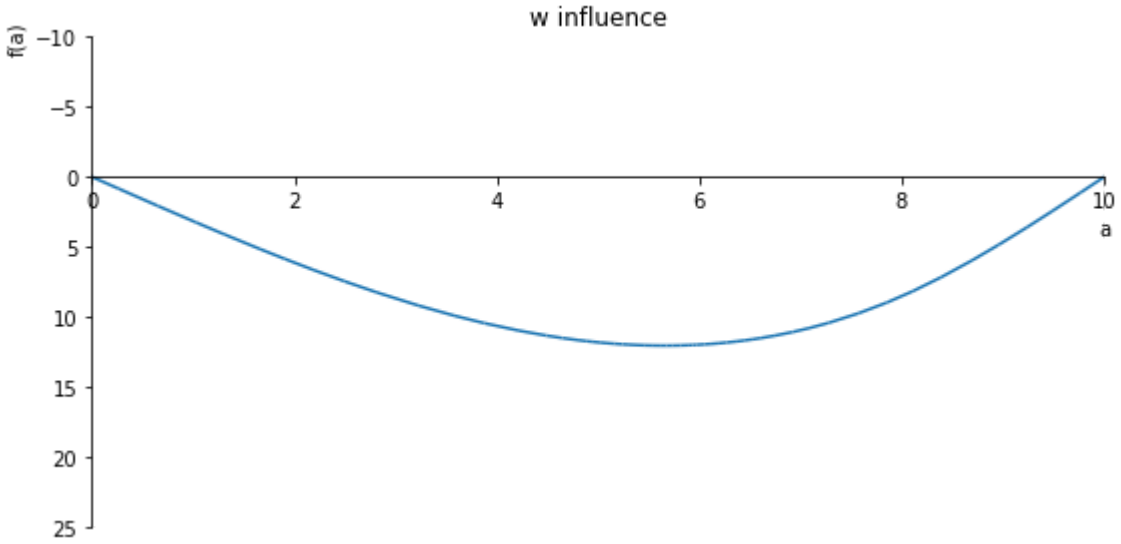
```
# Voorbeeld 1
w_influence = (-1/6)*(1/10)*sf(10, a, 1)*sf(x, 0, 3) + 1/6*sf(x, a, 3) - (1/6)*(1 - (1/10
phi = sp.diff(w_influence, x)*-1
M = sp.diff(phi, x)
V = sp.diff(M, x)

x_value = 8
a_value = 5

sp.plot(w_influence.subs(a, a_value), ylim=(40, -10), xlim=(0, 10), size=(8, 4), title='w
sp.plot(phi.subs(a, a_value), ylim=(10, -10), xlim=(0, 10), size=(8, 4), title='phi-lijn'
sp.plot(M.subs(a, a_value), ylim=(5, -2), xlim=(0, 10), size=(8, 4), title='M-lijn')
sp.plot(V.subs(a, a_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V-lijn')

sp.plot(w_influence.subs(x, x_value), ylim=(25, -10), xlim=(0, 10), size=(8, 4), title='w
sp.plot(phi.subs(x, x_value), ylim=(10, -10), xlim=(0, 10), size=(8, 4), title='Phi influ
sp.plot(M.subs(x, x_value), ylim=(5, -1), xlim=(0, 10), size=(8, 4), title='M influence')
sp.plot(V.subs(x, x_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V influence')

Av_inlfuence = (1/10)*sf(10, a, 1)
sp.plot(Av_inlfuence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title= 'Av-influence')
Bv_inlfuence = (1 - (1/10)*sf(10, a, 1))
sp.plot(Bv_inlfuence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='Bv-influence')


plt.show()
```

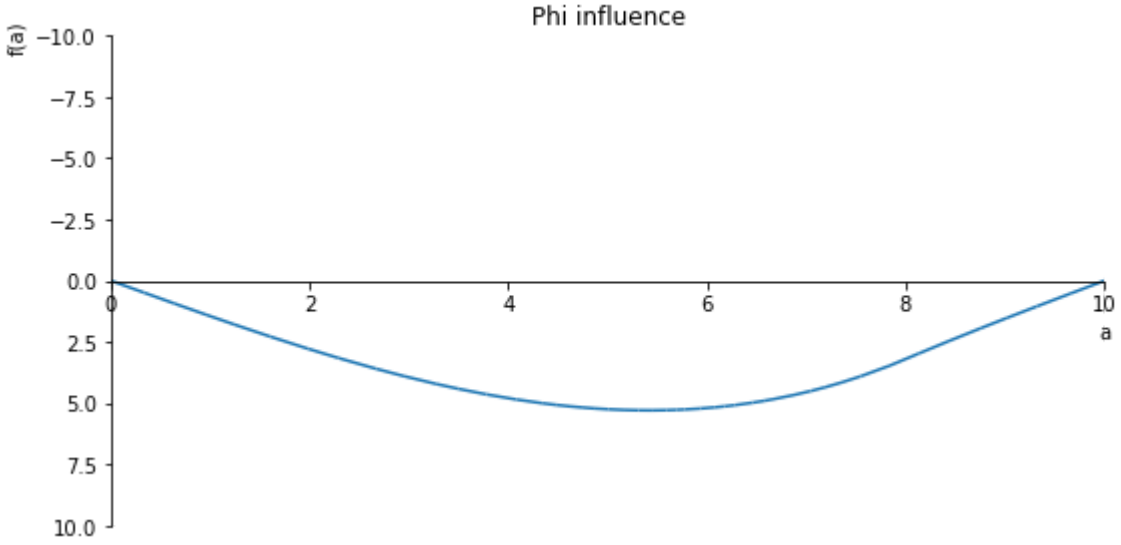

w-lijn
f(x)
−10
0
10
20
30
40
0
2
4
6
8
10
x

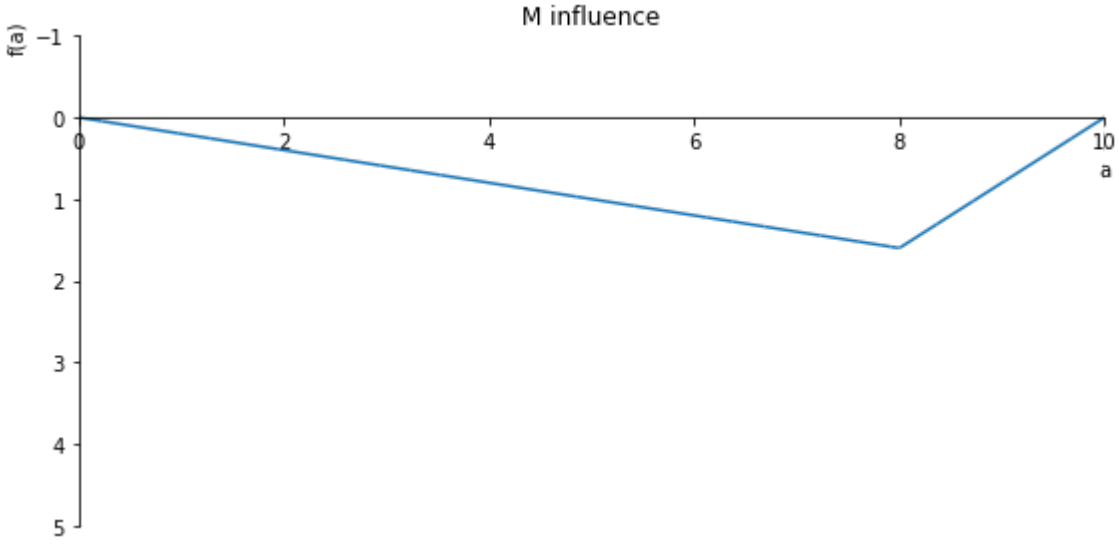

phi-lijn
f(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10
x

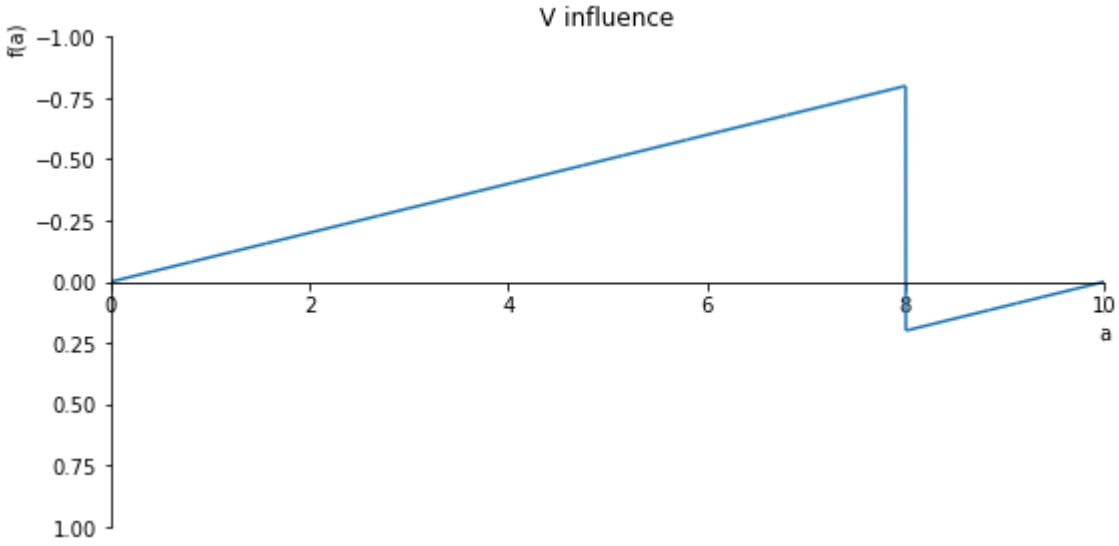

M-lijn
f(x)
−2
−1
0
1
2
3
4
5
0
2
4
6
8
10
x

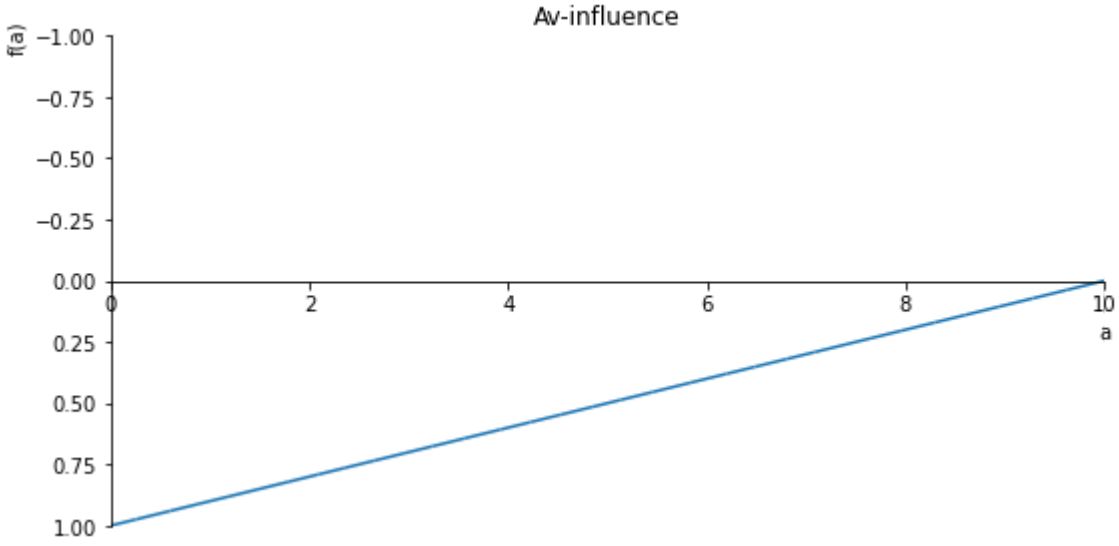

V-lijn
f(x)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
x

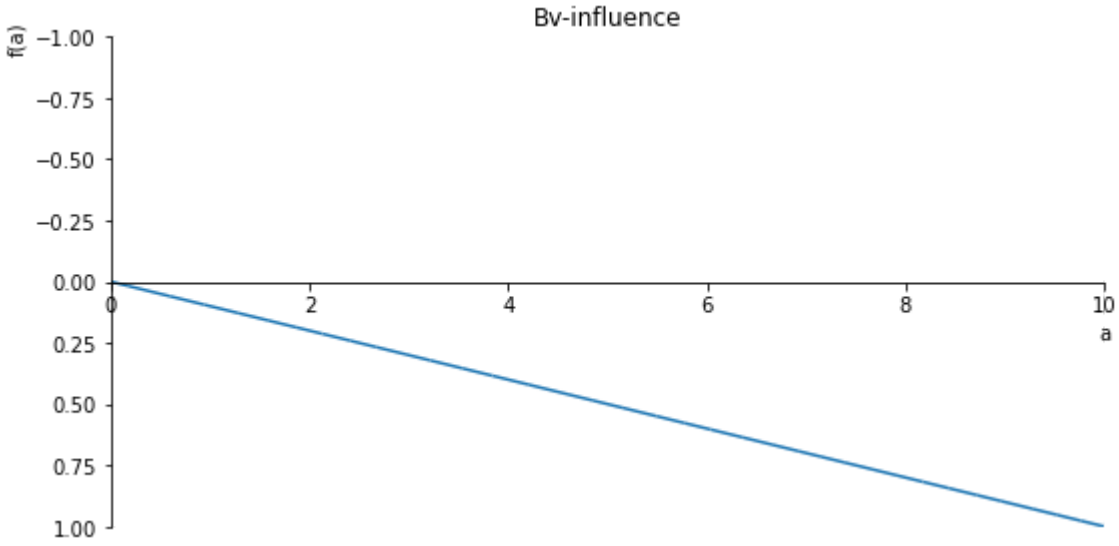

w influence
f(a)
−10
−5
0
5
10
15
20
25
0
2
4
6
8
10
a


Phi influence
f(a)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10
a

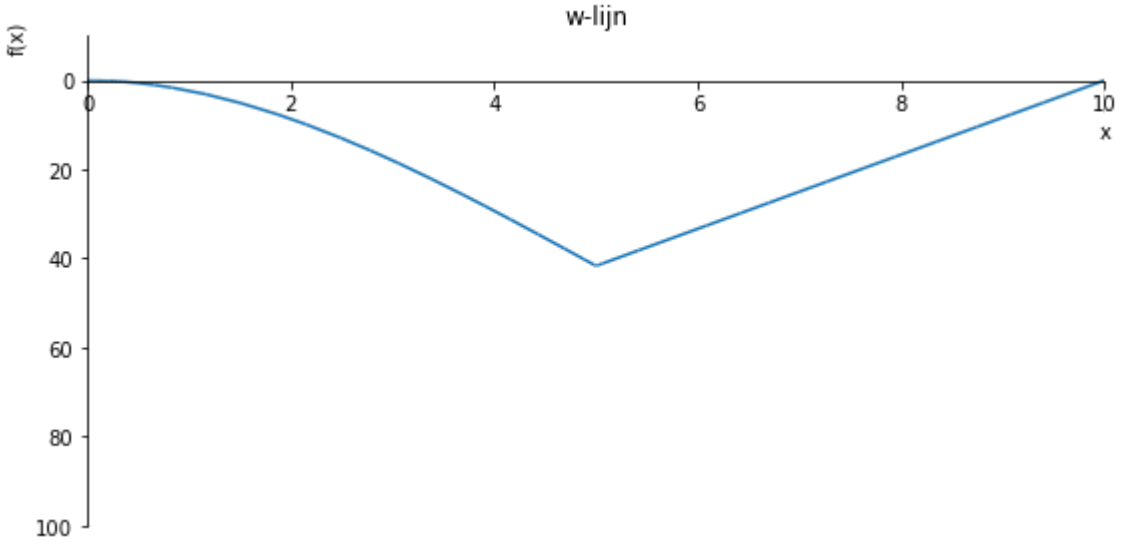

M influence
f(a)
−1
0
1
2
3
4
5
0
2
4
6
8
10
a

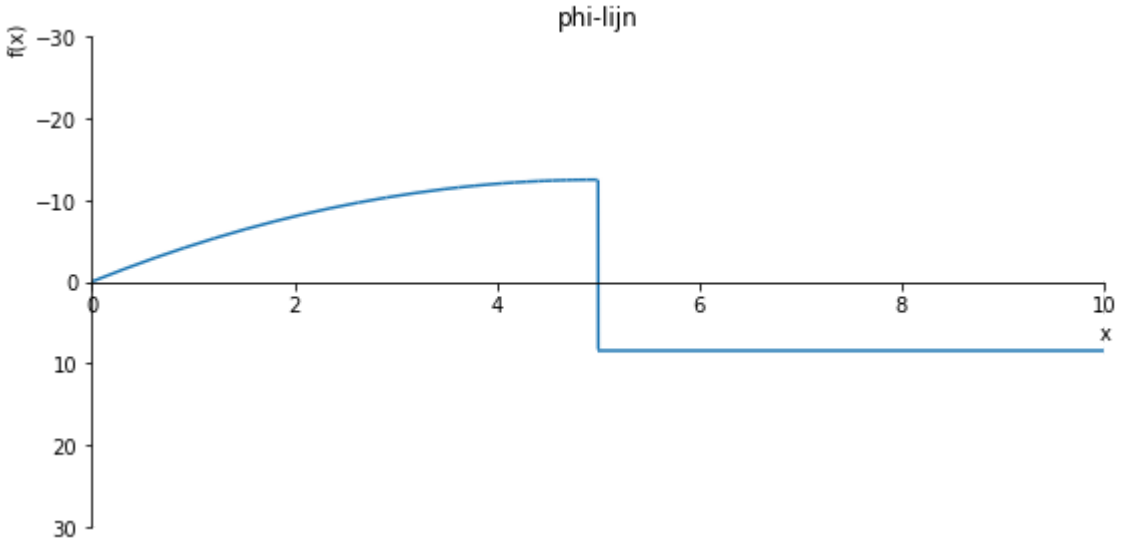

V influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

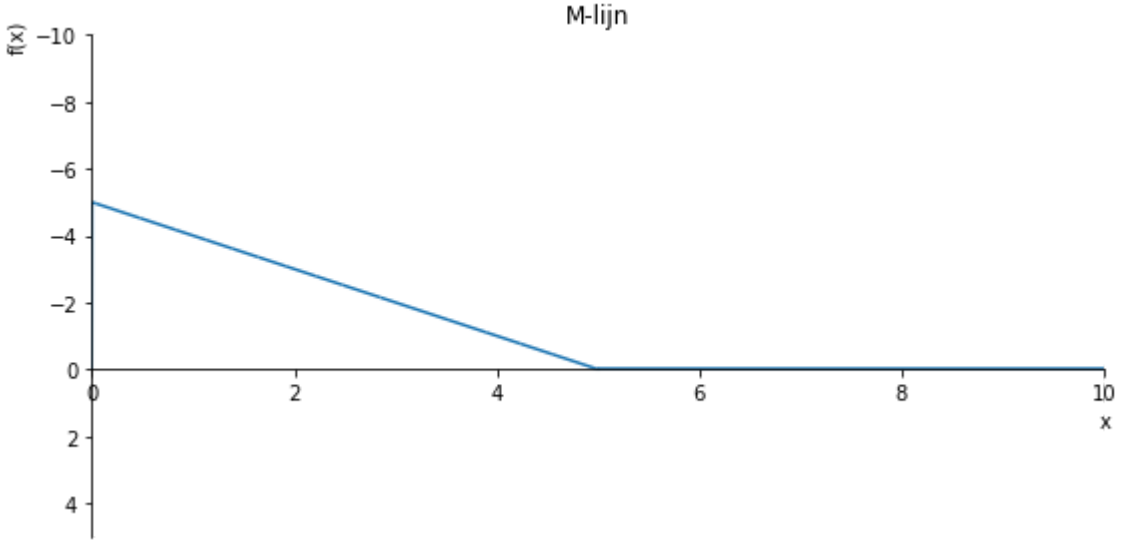

Av-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

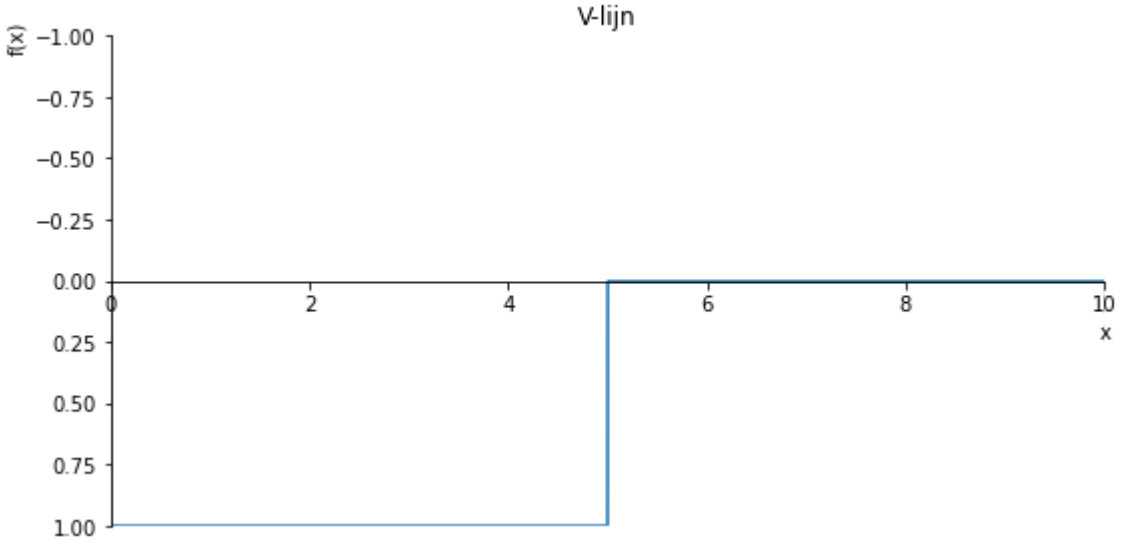

Bv-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

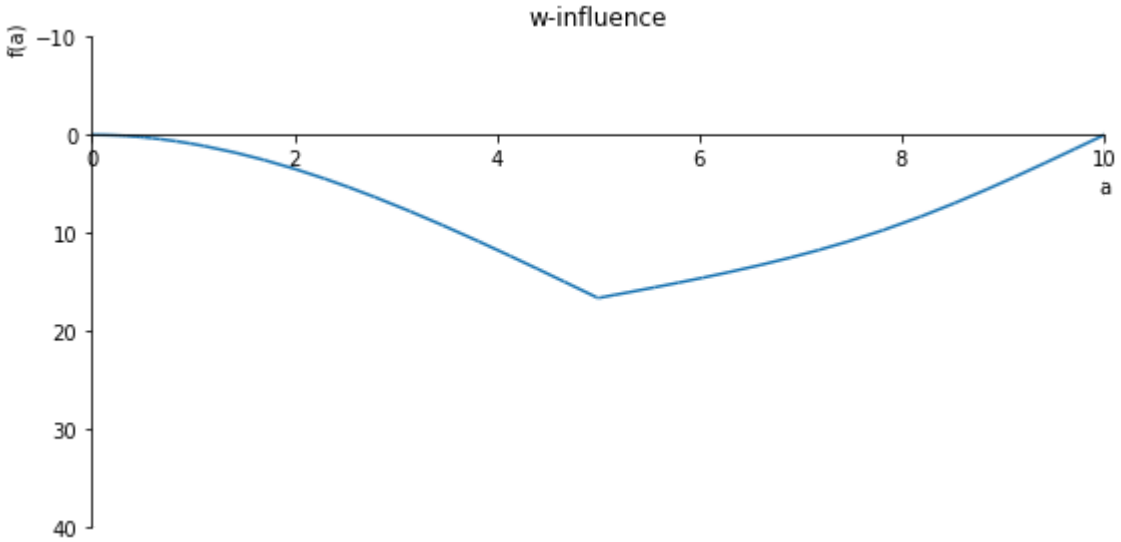
```
# voorbeeld 2.
w_influence = (-1/6)*((-sf(5, a, 1) + sf(10, a, 1))/5)*sf(x, 0, 3) + (1/2)*(-2*sf(5, a, 1
phi = sp.diff(w_influence, x)*-1
M = sp.diff(phi, x)
V = sp.diff(M, x)

x_value = 8
a_value = 5

sp.plot(w_influence.subs(a, a_value), ylim=(100, -10), xlim=(0, 10), size=(8, 4), title='
sp.plot(phi.subs(a, a_value), ylim=(30, -30), xlim=(0, 10), size=(8, 4), title='phi-lijn'
sp.plot(M.subs(a, a_value), ylim=(5, -10), xlim=(0, 10), size=(8, 4), title='M-lijn')
sp.plot(V.subs(a, a_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V-lijn')


sp.plot(w_influence.subs(x, x_value), ylim=(40, -10), xlim=(0, 10), size=(8, 4), title='w
sp.plot(phi.subs(x, x_value), ylim=(15, -10), xlim=(0, 10), size=(8, 4), title='phi-influ
sp.plot(M.subs(x, x_value), ylim=(3, -3), xlim=(0, 10), size=(8, 4), title='M-influence')
sp.plot(V.subs(x, x_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V-influence')


Av_inlfuence = (-sf(5, a, 1) + sf(10, a, 1))/5
sp.plot(Av_inlfuence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title= 'Av-influence')
Bv_inlfuence = 1- (-sf(5, a, 1) + sf(10, a, 1))/5
sp.plot(Bv_inlfuence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='Bv-influence')
Am = -2*sf(5, a, 1) + sf(10, a, 1)
sp.plot(Am, ylim=(6, -1), xlim=(0, 10), size=(8, 4), title='Am-influence')



plt.show()
```

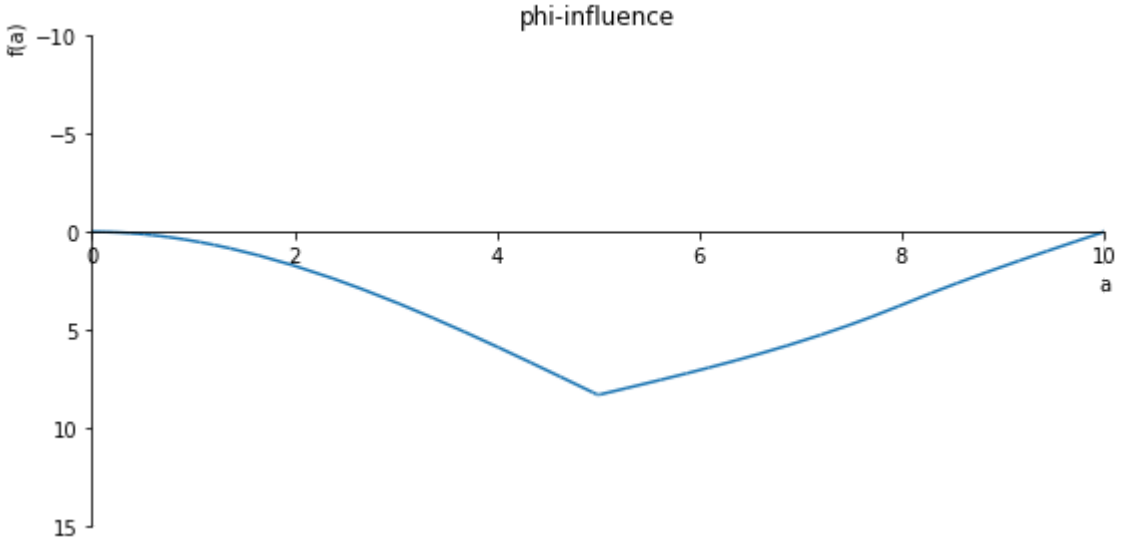

w-lijn
f(x)
0
20
40
60
80
100
0
2
4
6
8
10
x

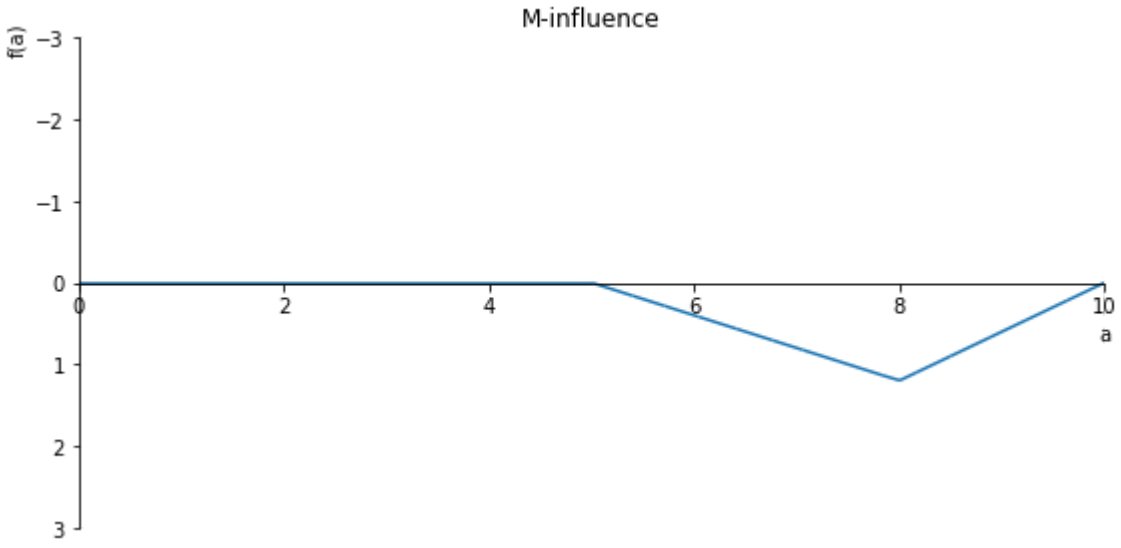

phi-lijn
f(x)
−30
−20
−10
0
10
20
30
0
2
4
6
8
10
x

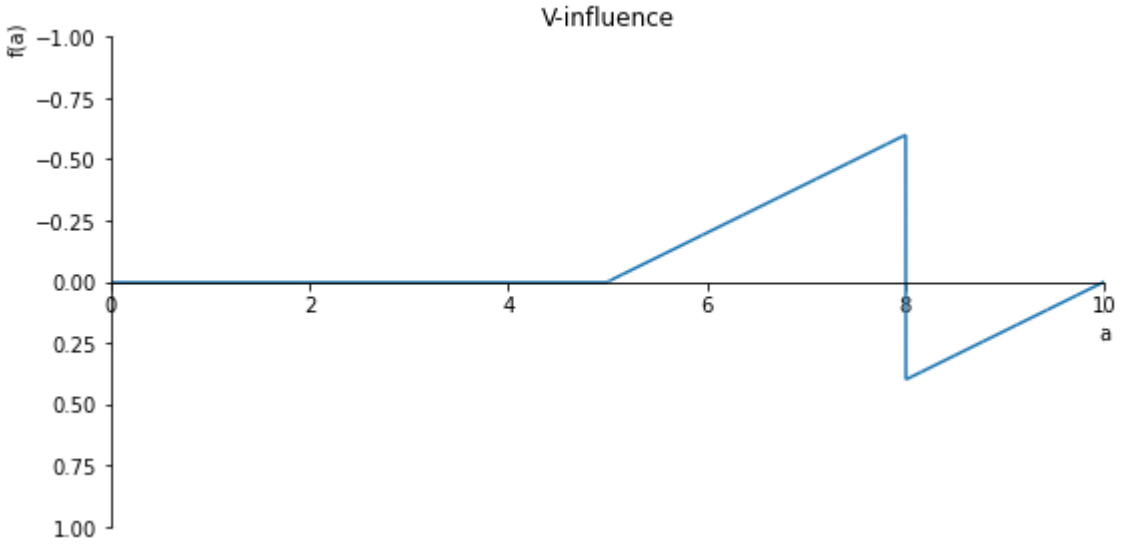

M-lijn
f(x)
−10
−8
−6
−4
−2
0
2
4
0
2
4
6
8
10
x

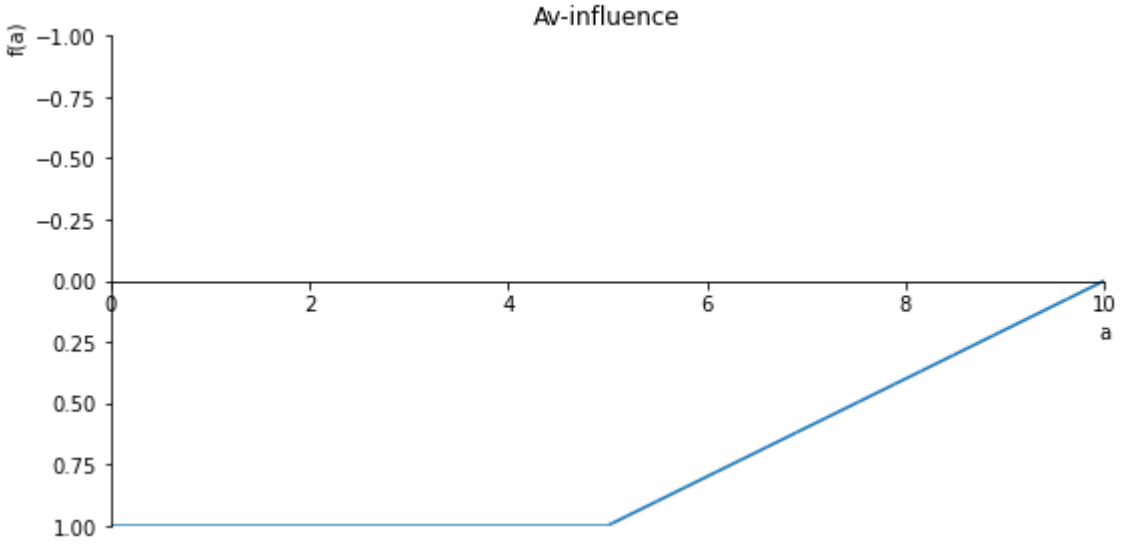

V-lijn
f(x)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
x

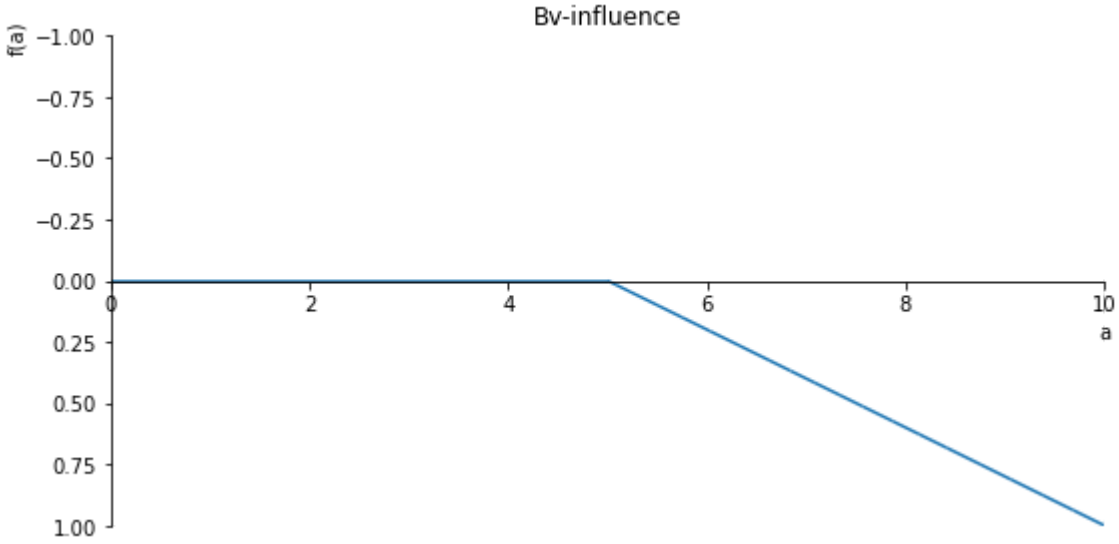

w-influence
f(a)
−10
0
10
20
30
40
0
2
4
6
8
10
a

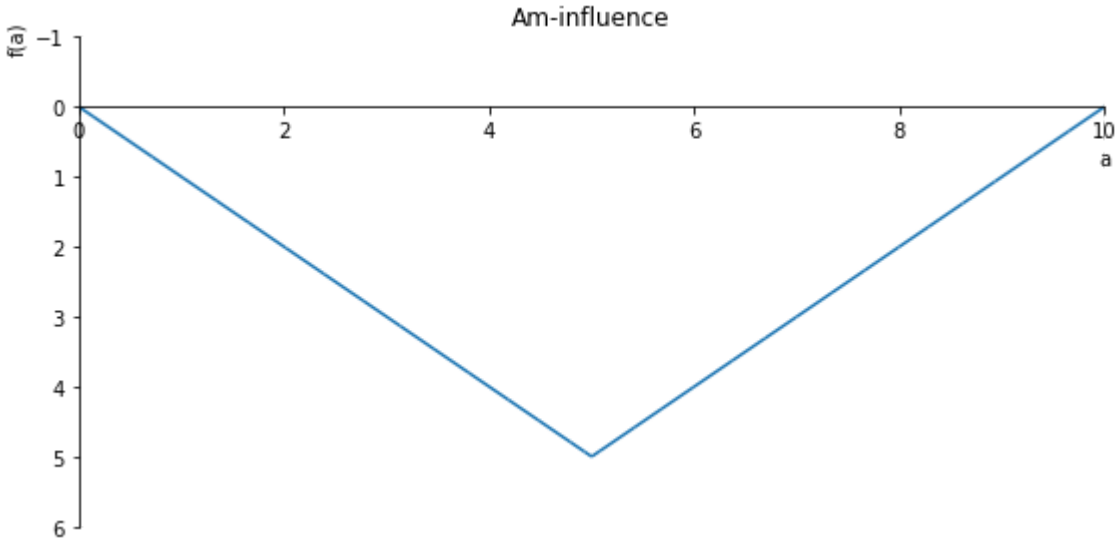

phi-influence
f(a)
−10
−5
0
5
10
15
0
2
4
6
8
10
a


M-influence
f(a)
−3
−2
−1
0
1
2
3
0
2
4
6
8
10
a

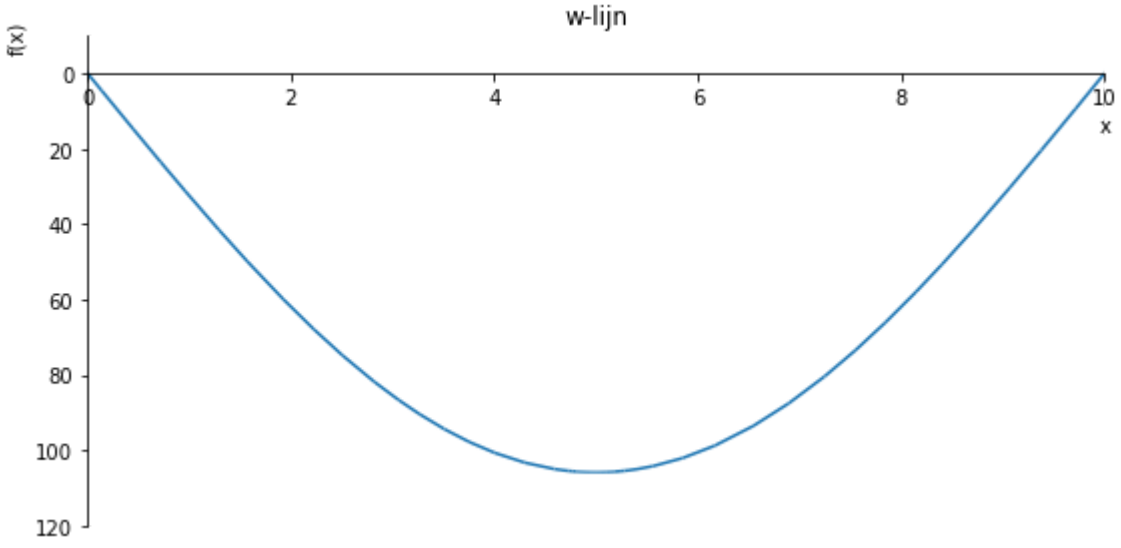

V-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

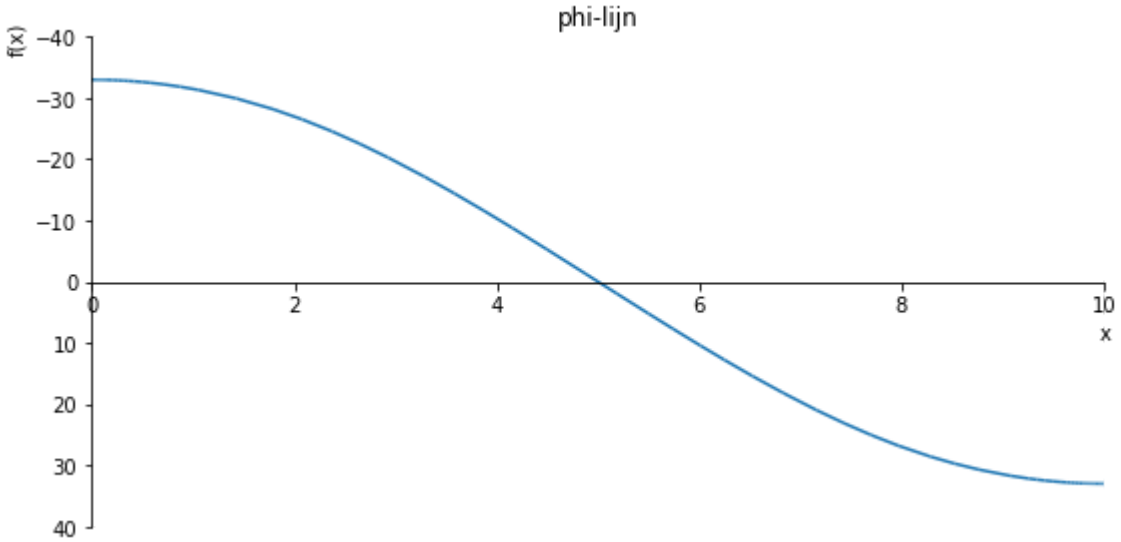

Av-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

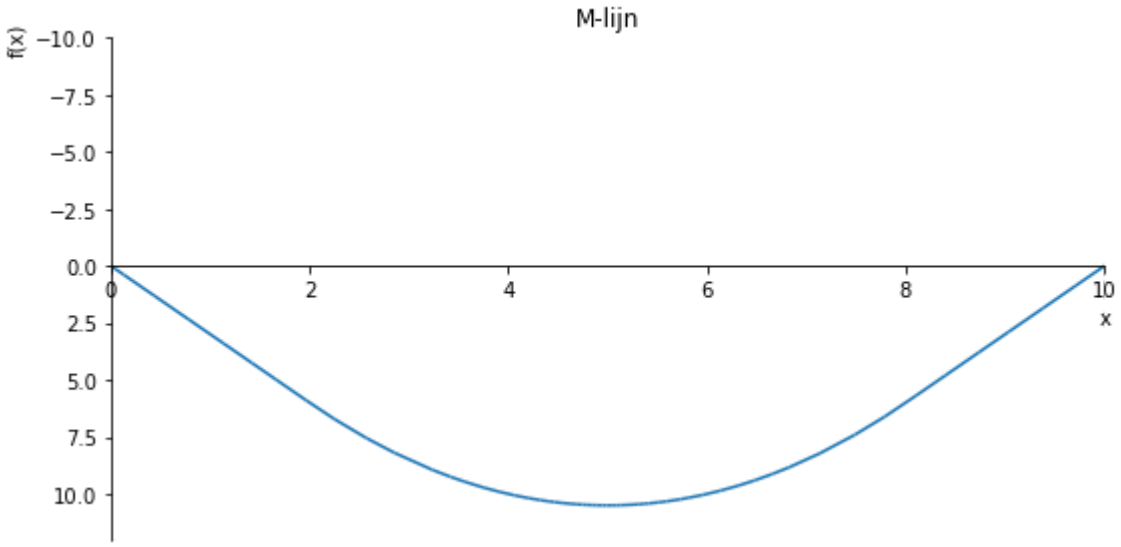

Bv-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

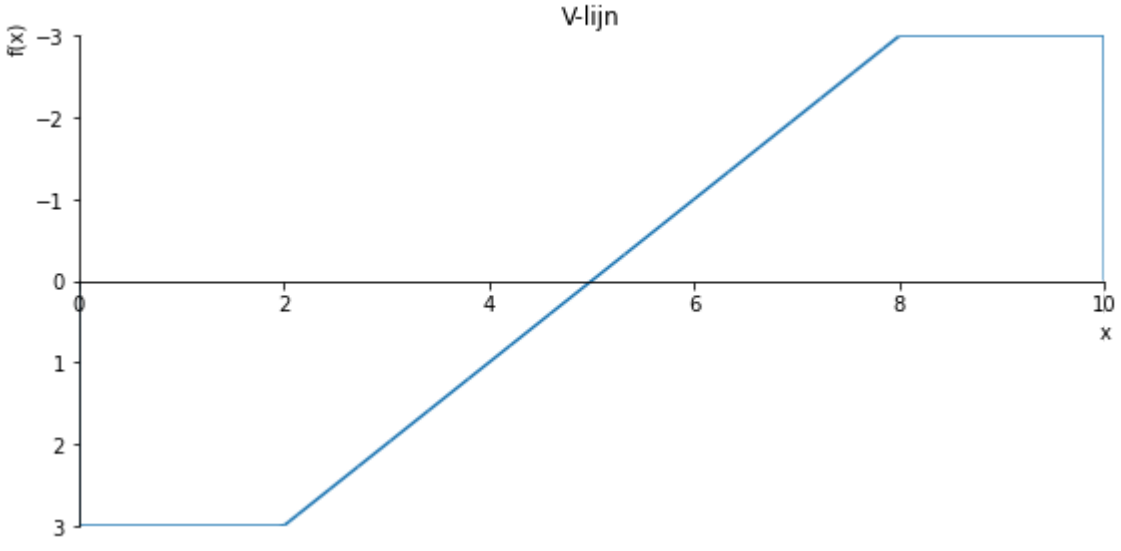

Am-influence
f(a)
−1
0
1
2
3
4
5
6
0
2
4
6
8
10
a

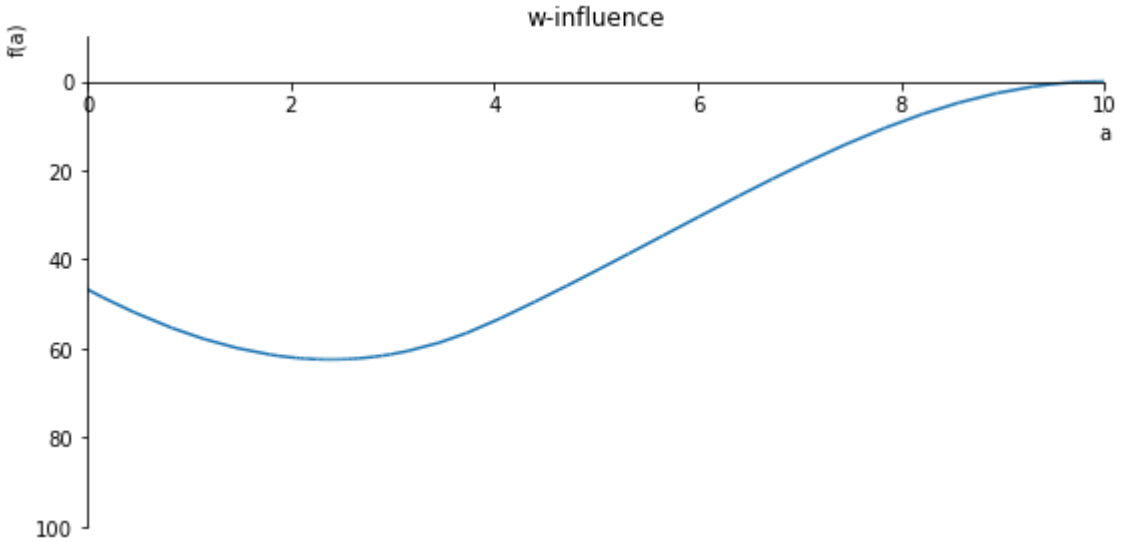
```
# voorbeeld 3
Av_influence = (sf(10, a, 2) - sf(4, a, 2))/20
Bv_influence = sf(10, a, 1) - sf(4, a, 1) - ((sf(10, a, 2) - sf(4, a, 2))/20)
C_phi = -(5/6)*(sf(10, a, 2) - sf(4, a, 2)) + (1/240)*sf(10, a, 4) - (1/240)*sf(4, a, 4)


w_influence = (-1/6)*((sf(10, a, 2) - sf(4, a, 2))/20)*sf(x, 0, 3) + 1/24*sf(x, a, 4) - 1
phi = sp.diff(w_influence, x)*-1
M = sp.diff(phi, x)
V = sp.diff(M, x)

x_value = 8
a_value = 2

sp.plot(w_influence.subs(a, a_value), ylim=(120, -10), xlim=(0, 10), size=(8, 4), title='
sp.plot(phi.subs(a, a_value), ylim=(40, -40), xlim=(0, 10), size=(8, 4), title='phi-lijn'
sp.plot(M.subs(a, a_value), ylim=(12, -10), xlim=(0, 10), size=(8, 4), title='M-lijn')
sp.plot(V.subs(a, a_value), ylim=(3, -3), xlim=(0, 10), size=(8, 4), title='V-lijn')


sp.plot(w_influence.subs(x, x_value), ylim=(100, -10), xlim=(0, 10), size=(8, 4), title='
sp.plot(phi.subs(x, x_value), ylim=(40, -10), xlim=(0, 10), size=(8, 4), title='phi-influ
sp.plot(M.subs(x, x_value), ylim=(10, -1), xlim=(0, 10), size=(8, 4), title='M-influence'
sp.plot(V.subs(x, x_value), ylim=(1, -5), xlim=(0, 10), size=(8, 4), title='V-influence')

sp.plot(Av_influence, ylim=(5, -1), xlim=(0, 10), size=(8, 4), title='Av-influence')
sp.plot(Bv_influence, ylim=(5, -1), xlim=(0, 10), size=(8, 4), title='Bv-influence')

plt.show()
```

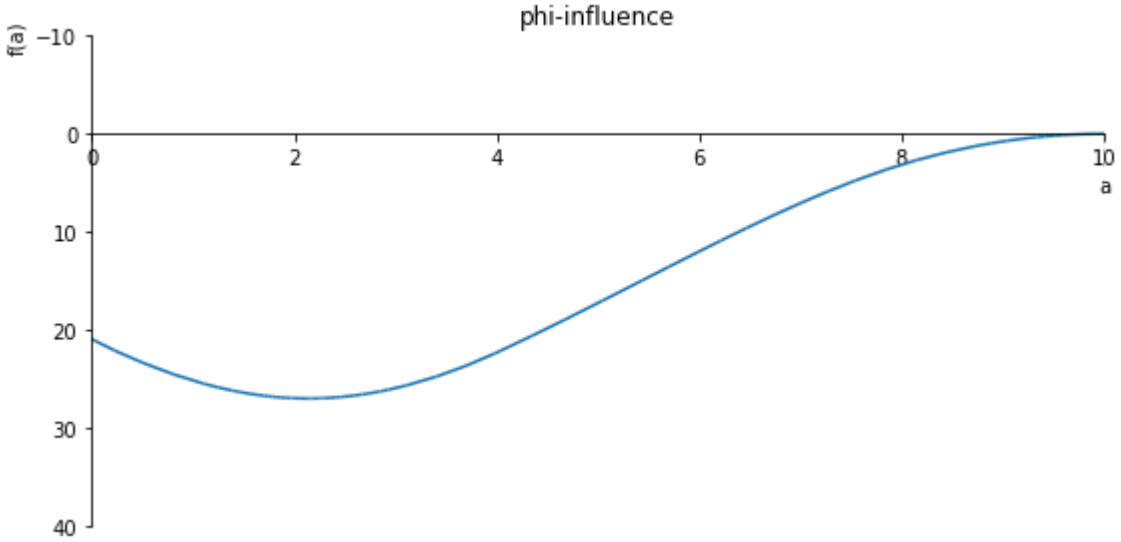

w-lijn
f(x)
0
20
40
60
80
100
120
0
2
4
6
8
10
x

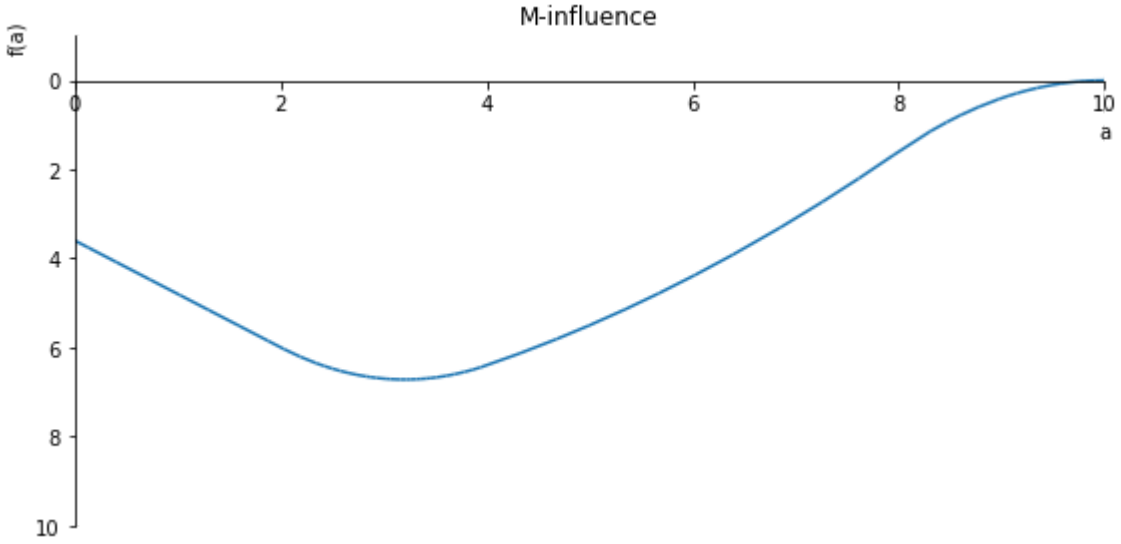

phi-lijn
f(x)
−40
−30
−20
−10
0
10
20
30
40
0
2
4
6
8
10
x

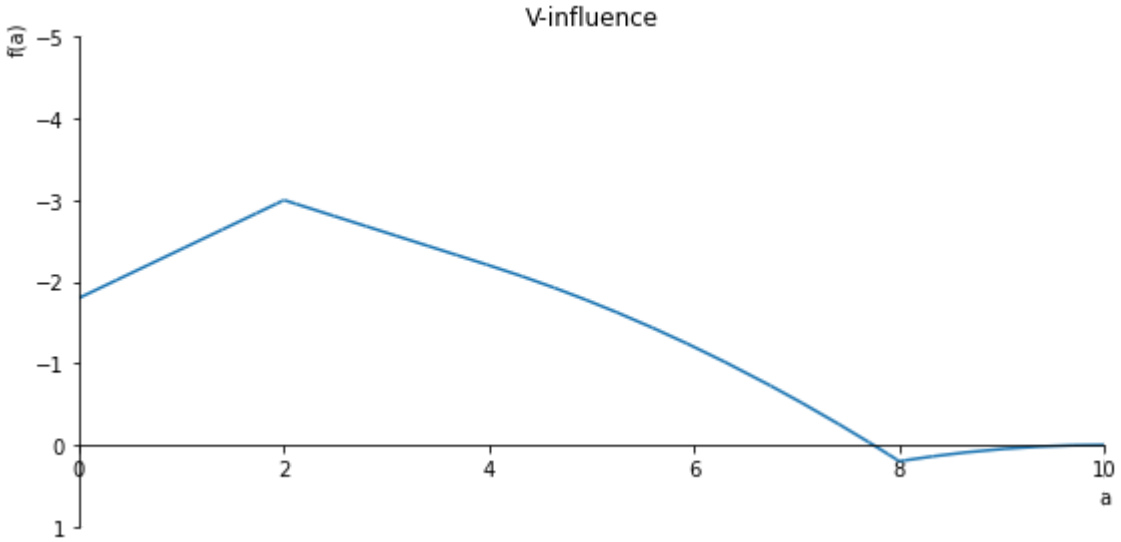

M-lijn
f(x)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
2
4
6
8
10
x

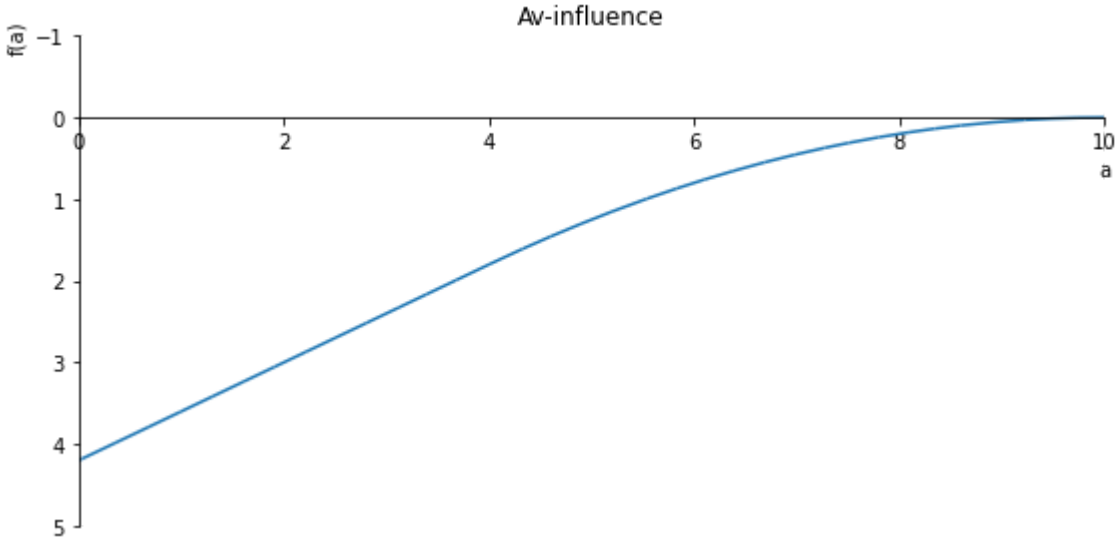

V-lijn
f(x)
−3
−2
−1
0
1
2
3
0
2
4
6
8
10
x

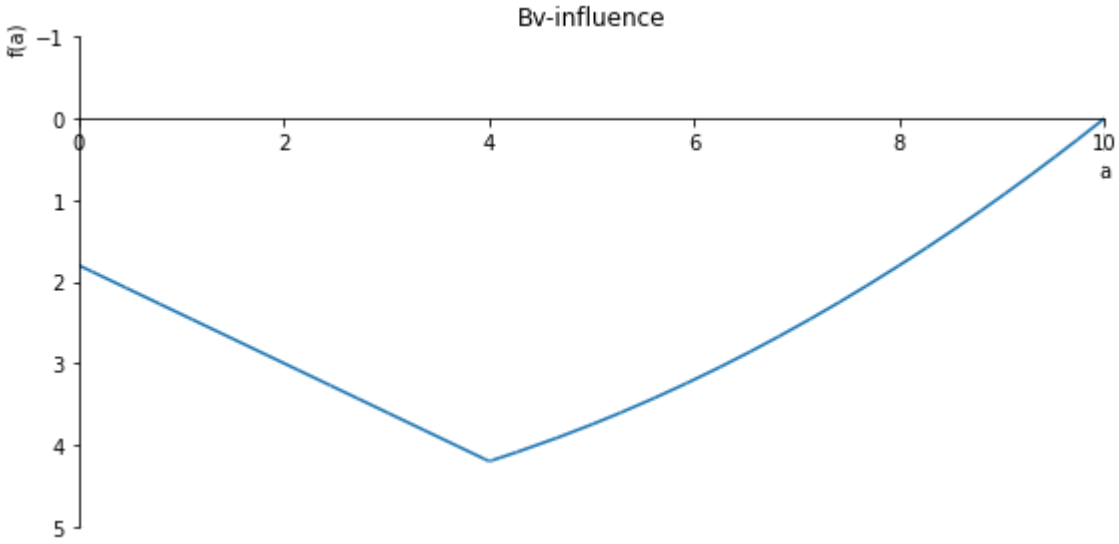

w-influence
f(a)
0
20
40
60
80
100
0
2
4
6
8
10
a


phi-influence
f(a)
−10
0
10
20
30
40
0
2
4
6
8
10
a

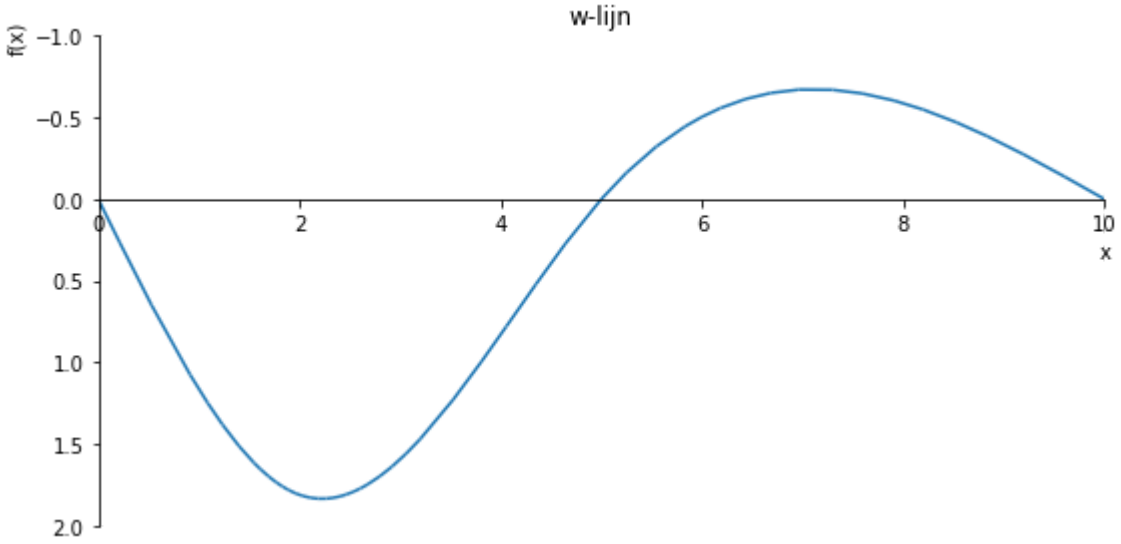

M-influence
f(a)
0
2
4
6
8
10
a

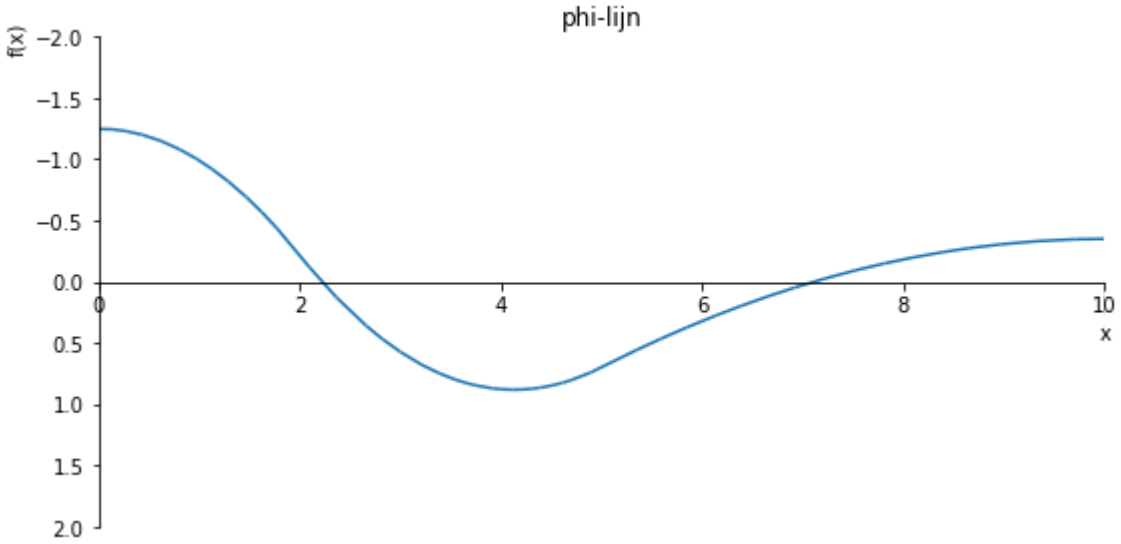

V-influence
f(a)
−5
−4
−3
−2
−1
0
1
0
2
4
6
8
10
a

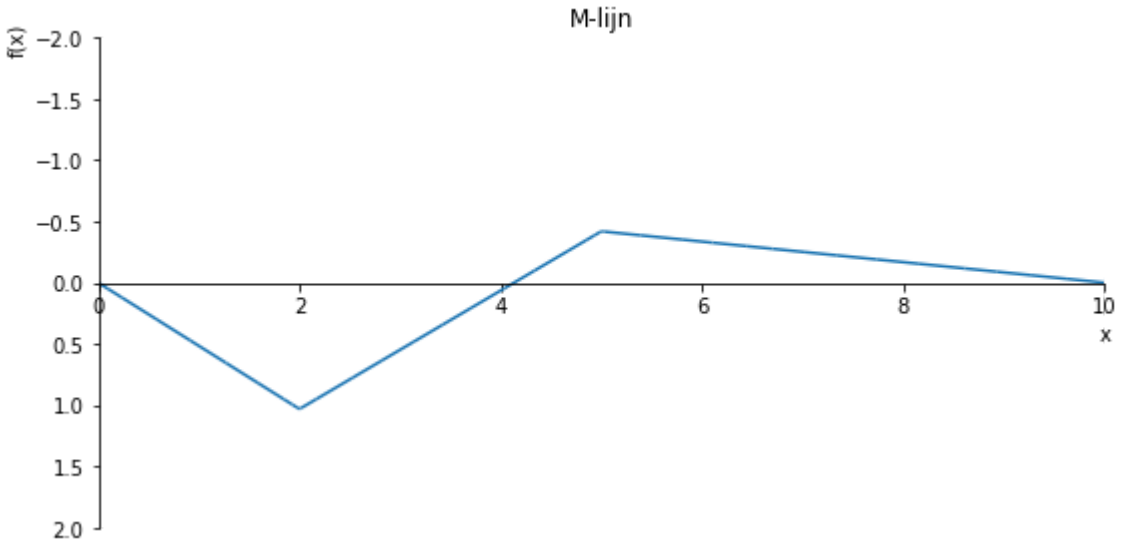

Av-influence
f(a)
−1
0
1
2
3
4
5
0
2
4
6
8
10
a

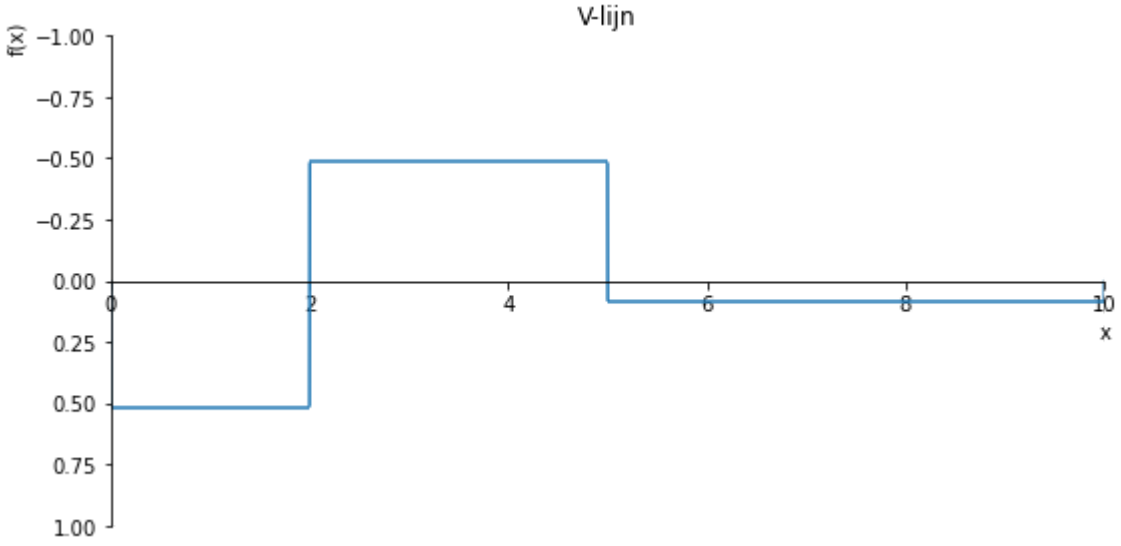

Bv-influence
f(a)
−1
0
1
2
3
4
5
0
2
4
6
8
10
a

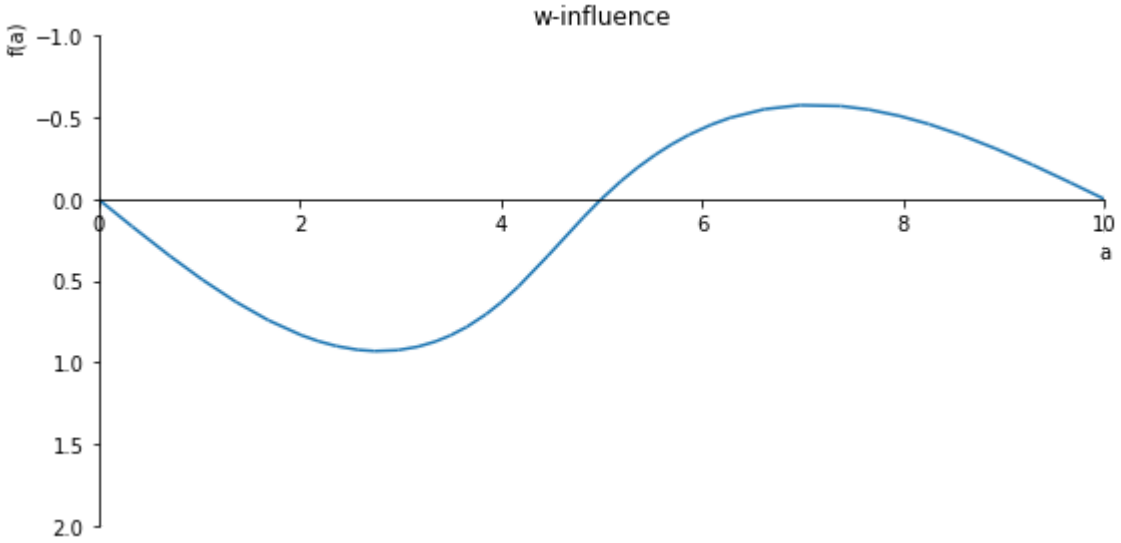
```
# voorbeeld 4
Av_influence = (1/500)*sf(10, a, 3) - (1/20)*sf(10, a, 1) - (1/250)*sf(5, a, 3)
Cv_influence = -2*(Av_influence) + (1/5)*sf(10, a, 1)
Bv_influence = 1 - Av_influence - Cv_influence
Cphi = -(25/6)*Av_influence +(1/30)*sf(5, a, 3)

w_influence = (-1/6)*(Av_influence)*sf(x, 0, 3) + (1/6)*sf(x, a, 3) - (1/6)*Cv_influence*
phi = sp.diff(w_influence, x)*-1
M = sp.diff(phi, x)
V = sp.diff(M, x)

x_value = 4
a_value = 2

sp.plot(w_influence.subs(a, a_value), ylim=(2, -1), xlim=(0, 10), size=(8, 4), title='w-l
sp.plot(phi.subs(a, a_value), ylim=(2, -2), xlim=(0, 10), size=(8, 4), title='phi-lijn')
sp.plot(M.subs(a, a_value), ylim=(2, -2), xlim=(0, 10), size=(8, 4), title='M-lijn')
sp.plot(V.subs(a, a_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V-lijn')


sp.plot(w_influence.subs(x, x_value), ylim=(2, -1), xlim=(0, 10), size=(8, 4), title='w-i
sp.plot(phi.subs(x, x_value), ylim=(1, -(1/2)), xlim=(0, 10), size=(8, 4), title='phi-inf
sp.plot(M.subs(x, x_value), ylim=(0.75, -0.5), xlim=(0, 10), size=(8, 4), title='M-influe
sp.plot(V.subs(x, x_value), ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='V-influence')


sp.plot(Av_influence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='Av-influence')
sp.plot(Cv_influence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='Cv-influence')
sp.plot(Bv_influence, ylim=(1, -1), xlim=(0, 10), size=(8, 4), title='Bv-influence')



plt.show()
```

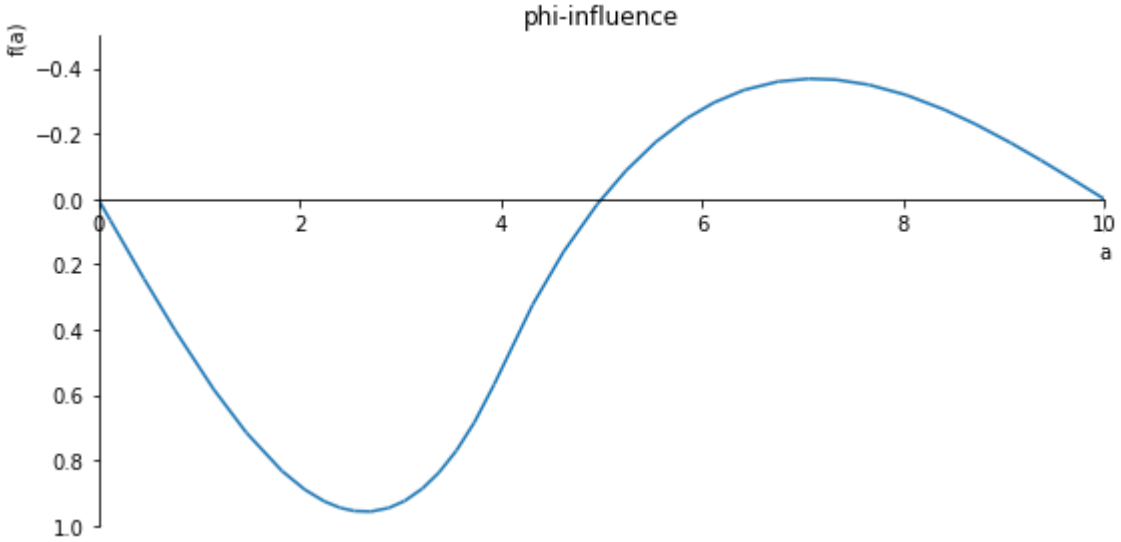

w-lijn
f(x)
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
10
x

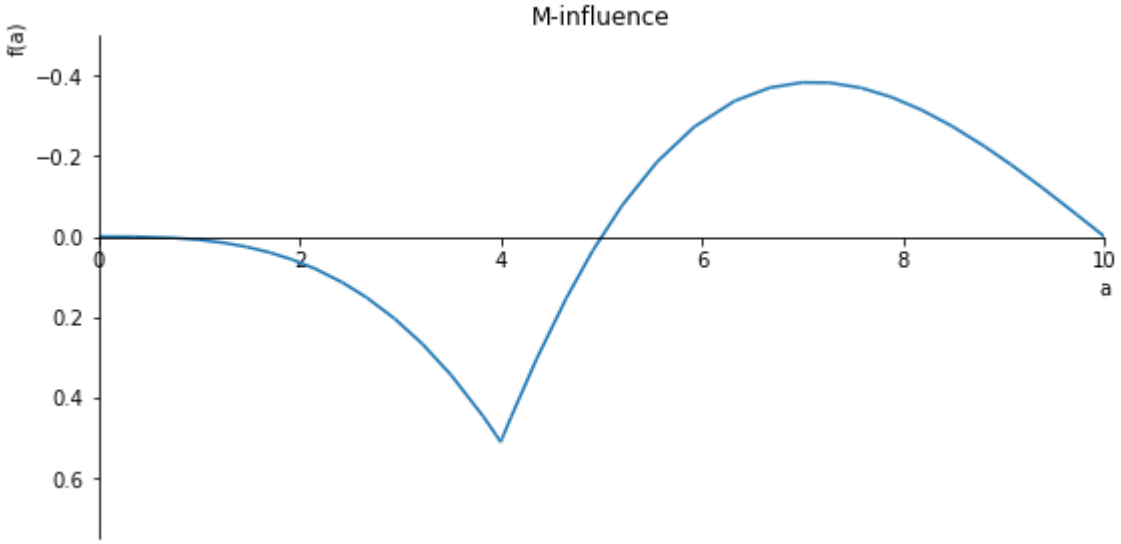

phi-lijn
f(x)
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
10
x

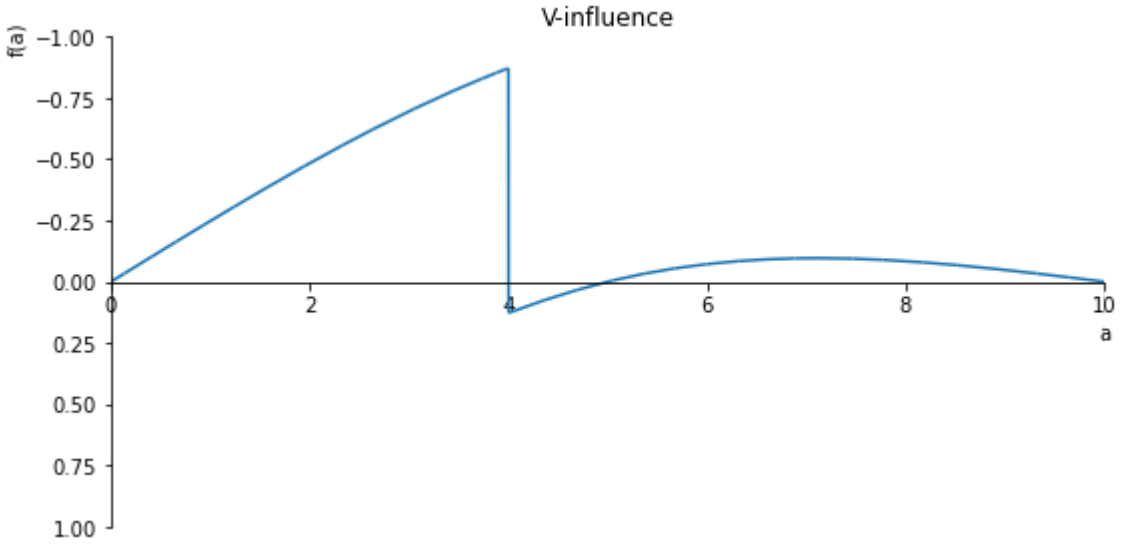

M-lijn
f(x)
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
10
x

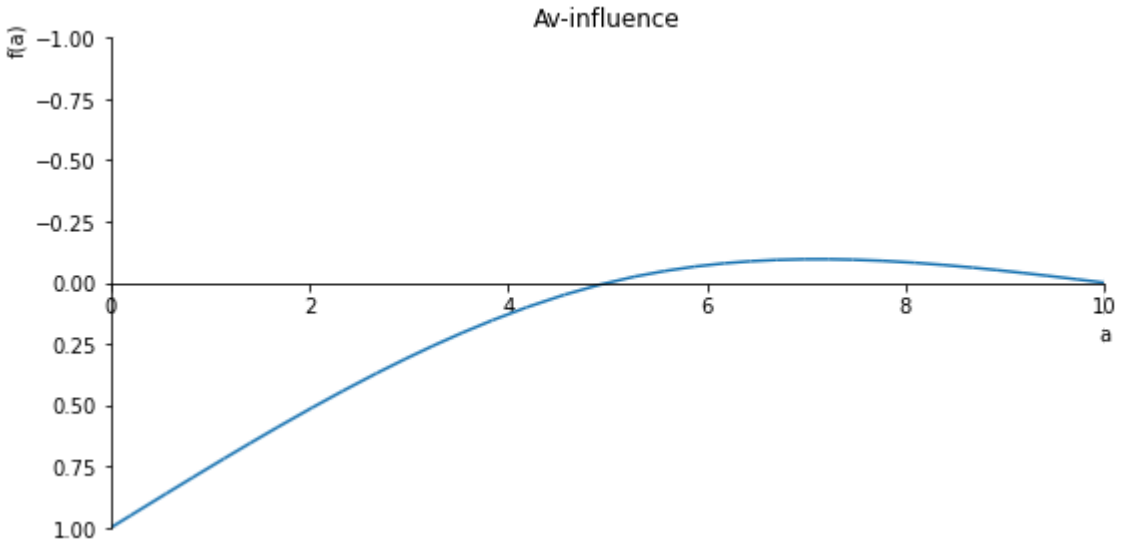

V-lijn
f(x)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
x

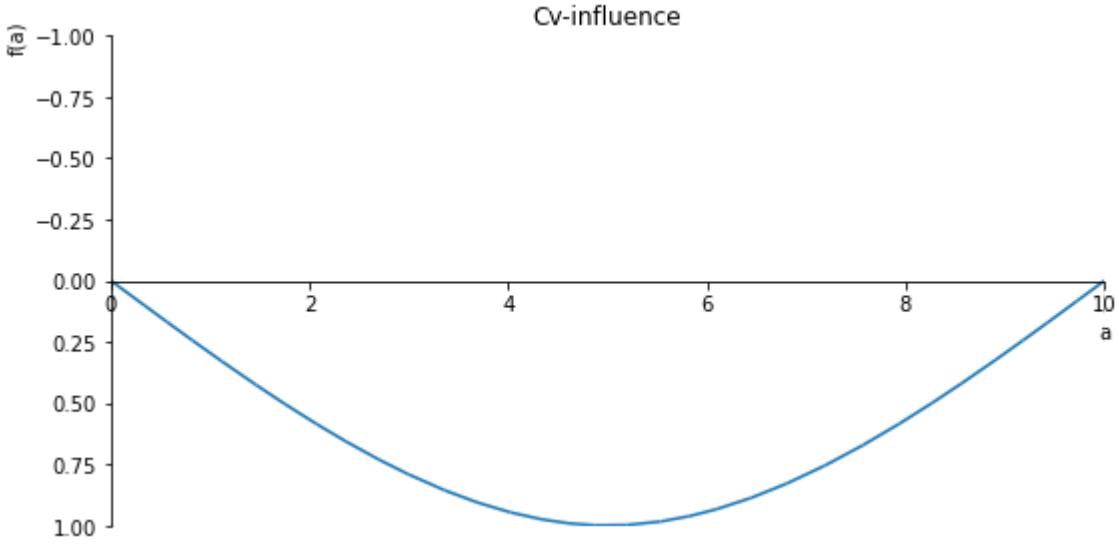

w-influence
f(a)
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
10
a

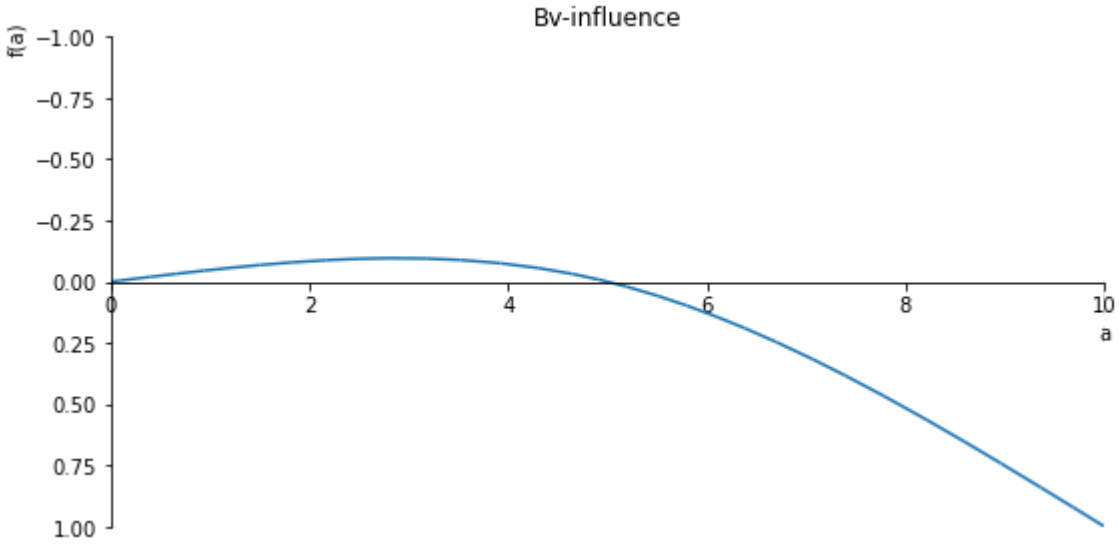

phi-influence
f(a)
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
a


M-influence
f(a)
−0.4
−0.2
0.0
0.2
0.4
0.6
0
2
4
6
8
10
a


V-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

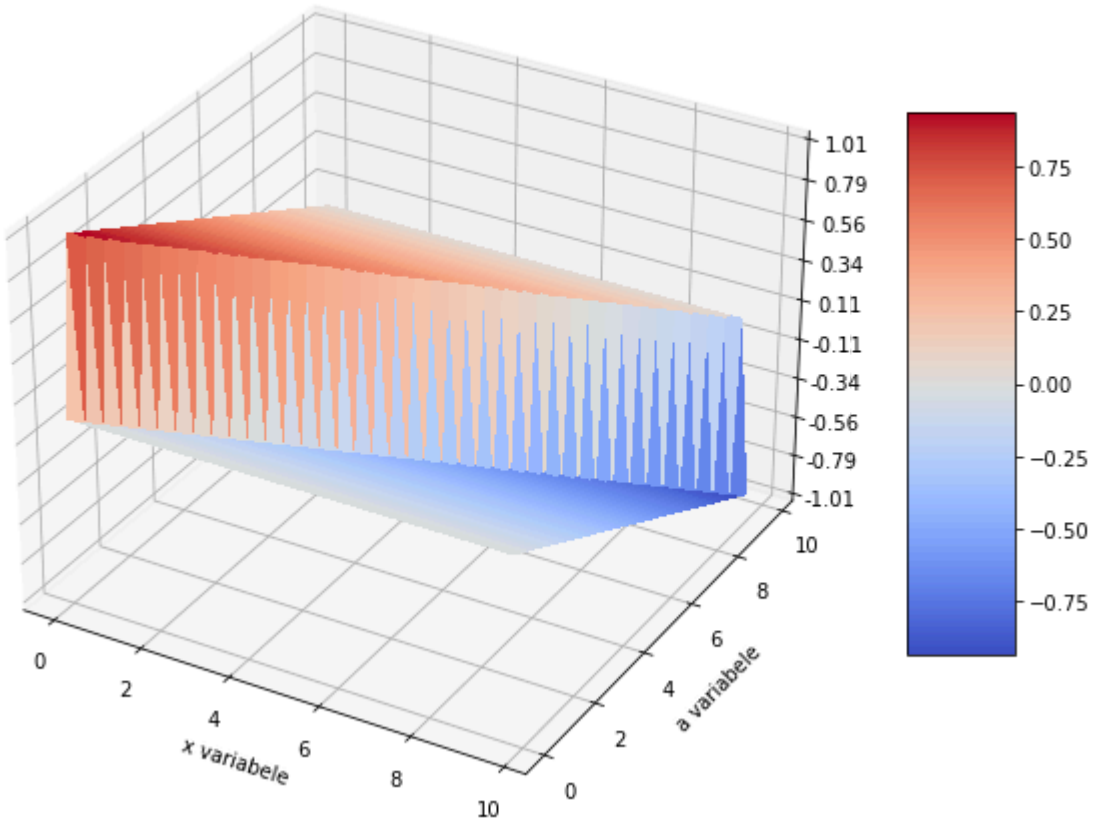

Av-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a


Cv-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a


Bv-influence
f(a)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0
2
4
6
8
10
a

```python
# 3D plotje

import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import sympy as sp
sf = sp.SingularityFunction

fig = plt.figure(figsize=(10, 10))
ax = fig.gca(projection='3d')

x = np.arange(0, 10, 0.25)
a = np.arange(0, 10, 0.25)
X, A = np.meshgrid(x, a)

def V(x, a):
    sf_x = np.where(x >= 0, 1, 0)
    sf_xa = np.where(x - a >= 0, 1, 0)
    sf_x10 = np.where(x - 10 >= 0, 1, 0)
    return ((10-a)/10)*1.0*sf_x - 1.0*sf_xa + (a/10)*1.0*sf_x10

Z = V(X, A)

surf = ax.plot_surface(X, A, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter('{x:.02f}')

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.xlabel('x variabele')
plt.ylabel('a variabele')
plt.title('3D weergave dwarskracht voor variabele x en a')
plt.show()
```

3D weergave dwarskracht voor variabele x en a



Tweedimensionale voorbeelden

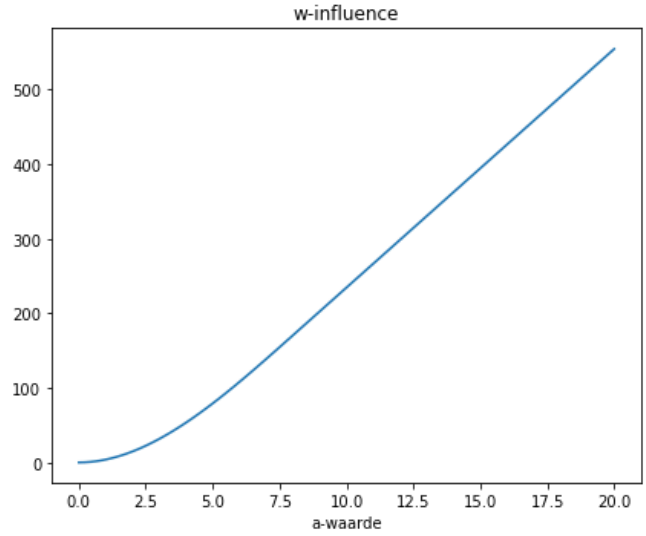
```
import sympy as sp
import numpy as np
from sympy import symbols, And, Or
E, I = symbols('E, I')
x = symbols('x')
a = symbols('a')
s = symbols('s')
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
from sympy import Piecewise
import pandas as pd
```

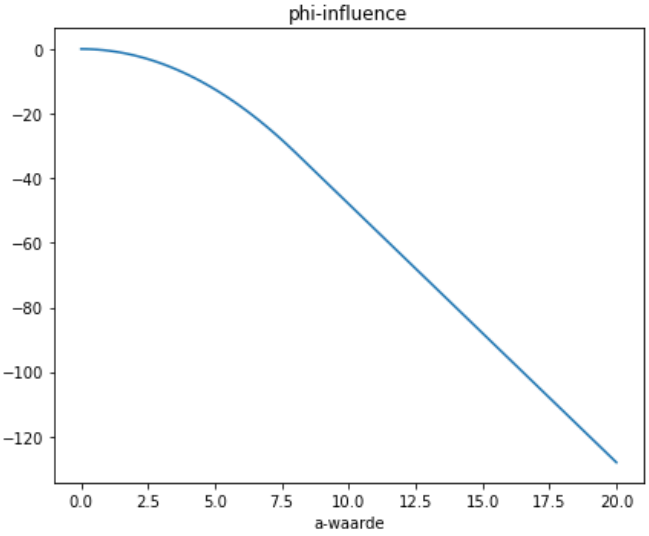
```
#Voorbeeld 5, Berekening van de onbekendes
Am, Av, Cv, Bv, Ah, Ch, Bh, uc, wc, wb, ub, alpha, a = sp.symbols('Am, Av, Cv, Bv, Ah, Ch

hoek_rad = sp.rad(alpha)
# hoek_rad = sp.rad(0*sp.Function('sf')(s, 0, 0) - 0*sp.Function('sf')(s, 10, 0) + 90*sp.
# display(hoek_rad)

V20 = -Av - 1*sp.cos(hoek_rad) - Cv - Bv
M20 = -Am - 20*Av - 10*Cv - 1*sp.cos(hoek_rad)*sp.Function('sf')(20, a, 1) - 10*Bv - 5*Cv
N20 = -Ah - Bh - Ch + 1*sp.sin(hoek_rad)
N10 = -Ah - Bh + 1*sp.sin(hoek_rad) - Av - 1*sp.cos(hoek_rad)
V10 = -Av - Bv - 1*sp.cos(hoek_rad) + Ah - 1*sp.sin(hoek_rad)
N15 = -Ah - Bh - Ch + 1*sp.sin(hoek_rad) + Av + 1*sp.cos(hoek_rad) + Bv
V15 = -Av - 1*sp.cos(hoek_rad) - Bv - Cv - Ah - Bh + 1*sp.sin(hoek_rad)

u10 = -10*Ah -ub + 1*sp.sin(hoek_rad)*sp.Function('sf')(10, a, 1) + 50*Am + (1000/6)*Av +
w10 = 50*Am + (1000/6)*Av + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(10, a, 3) + wb + 10*
u15 = -15*Ah - 5*Bh -ub - uc + 1*sp.sin(hoek_rad)*sp.Function('sf')(15, a, 1) - (225/2)*A
w15 = (225/2)*Am + (3375/6)*Av + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(15, a, 3) + (12

oplossing = sp.solve((V20, M20, N20, N20, N10, V10, N15, V15, u10, w10, u15, w15), (Am, A
oplossing_in_breuken = {variabele: sp.nsimplify(waarde) for variabele, waarde in oplossin
oplossing_df = pd.DataFrame([(str(variabele), waarde) for variabele, waarde in oplossing_

pd.set_option('display.latex.repr', True)
pd.set_option('display.max_colwidth', None)

display(oplossing_df)
```

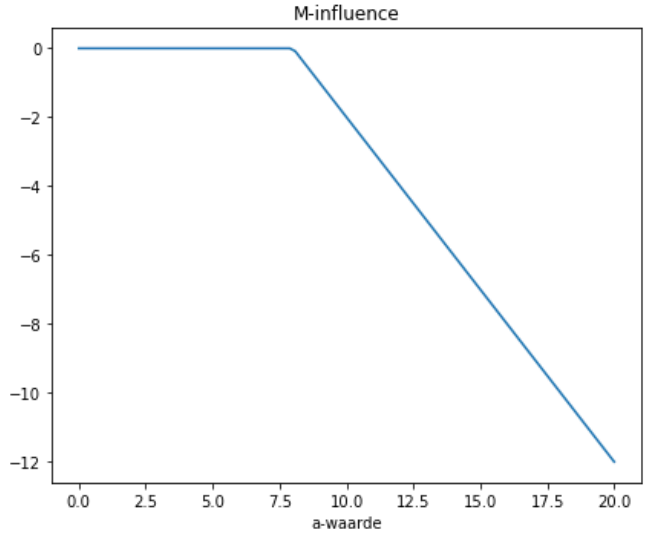
	Variabele	Waarde
0	Am	-sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000) + 20*cos(174532925199433*alpha/10000000000000000)
1	Av	-cos(174532925199433*alpha/10000000000000000)
2	Cv	0
3	Bv	0
4	Ah	sin(174532925199433*alpha/10000000000000000)
5	Ch	0
6	Bh	0
7	uc	-2*sf(10, a, 1)*sin(174532925199433*alpha/10000000000000000) + sf(15, a, 1)*sin(174532925199433*alpha/10000000000000000) - sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + 225*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/2 + 5*sin(174532925199433*alpha/10000000000000000) - 3375*cos(174532925199433*alpha/10000000000000000)/2
8	wc	sf(10, a, 3)*cos(174532925199433*alpha/10000000000000000)/3 - sf(15, a, 1)*sin(174532925199433*alpha/10000000000000000) - sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + 25*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/2 + 15*sin(174532925199433*alpha/10000000000000000) - 125*cos(174532925199433*alpha/10000000000000000)/6
9	wb	sf(10, a, 1)*sin(174532925199433*alpha/10000000000000000) - sf(10, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + 50*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000) - 10*sin(174532925199433*alpha/10000000000000000) - 2500*cos(174532925199433*alpha/10000000000000000)/3
10	ub	sf(10, a, 1)*sin(174532925199433*alpha/10000000000000000) + sf(10, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 - 50*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000) - 10*sin(174532925199433*alpha/10000000000000000) + 2500*cos(174532925199433*alpha/10000000000000000)/3

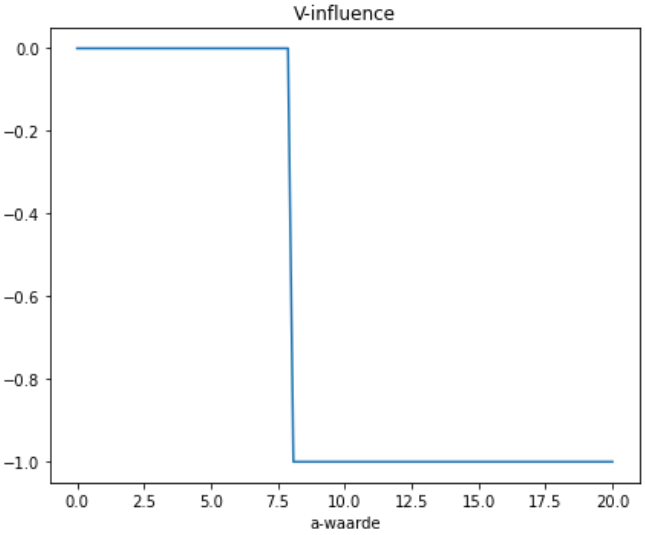
```
#Voorbeeld 5, Plotjes van de invloedslijnen

hoek_rad = sp.rad(alpha)


Am = -sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000) + 20*sp.cos(1745329251
Ah = sp.sin(174532925199433*alpha/10000000000000000)
Bh = 0
Ch = 0
Bv = 0
Av = -sp.cos(174532925199433*alpha/10000000000000000)
Cv = 0
ub = sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) + sf(10, a, 3)*sp.cos(1
wb = sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) - sf(10, a, 3)*sp.cos(1
uc = -2*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) + sf(15, a, 1)*sp.si
wc = sf(10, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/3 - sf(15, a, 1)*sp.sin

w = (1/2)*Am*sf(s, 0, 2) + (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (1
wV = (1/2)*Am*sf(s, 0, 2) + (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (
phi = sp.diff(w, s)*-1
M = sp.diff(phi, s)
V = sp.diff(wV, s, 3)

U = (-Ah*sf(s, 0, 1) - Bh*sf(s, 10, 1) - ub*sf(s, 10, 0) - Ch*sf(s, 15, 1) - uc*sf(s, 15,
N = sp.diff(U, s)

s_value = 8
a_values = np.linspace(0, 20, 100)



# alpha_value = 0*sf(s, 0, 0) - 0*sf(s, 10, 0) + 90*sf(s, 10, 0) - 90*sf(s, 15, 0) + 0*sf
# display(alpha_value)

condition1 = And(0 <= s_value, s_value <= 10)
condition2 = And(15 <= s_value, s_value <= 20)

if condition1 or condition2:
    alpha_value = 0
else:
    alpha_value = 90


w_values = [w.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
phi_values = [phi.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in
M_values = [M.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
Am_values = [Am.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
V_values = [V.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
U_values = [U.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
N_values = [N.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v

plt.figure(figsize=(12, 10))


plt.subplot(221)
plt.plot(a_values, w_values)
plt.title('w-influence')
plt.xlabel('a-waarde')
plt.subplot(222)
plt.plot(a_values, phi_values)
plt.title('phi-influence')
plt.xlabel('a-waarde')
plt.subplot(223)
```

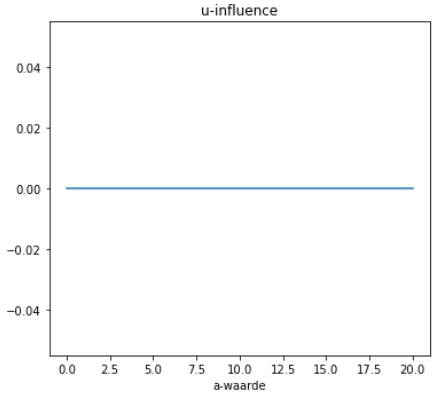
```
plt.plot(a_values, M_values)
plt.title('M-influence')
plt.xlabel('a-waarde')
plt.subplot(224)
plt.plot(a_values, V_values)
plt.title('V-influence')
plt.xlabel('a-waarde')


plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 5))

plt.subplot(131)
plt.plot(a_values, U_values)
plt.title('u-influence')
plt.xlabel('a-waarde')
plt.subplot(132)
plt.plot(a_values, N_values)
plt.title('N-influence')
plt.xlabel('a-waarde')
plt.subplot(133)
plt.plot(a_values, Am_values)
plt.title('Am-influence')
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()
```

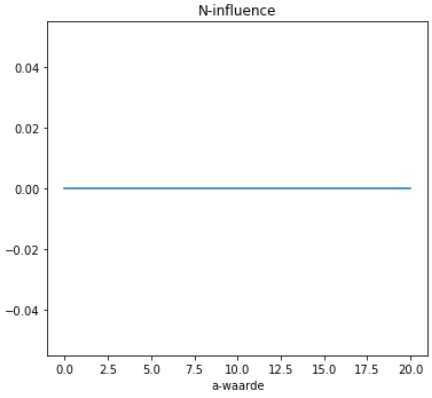

w-influence
a-waarde

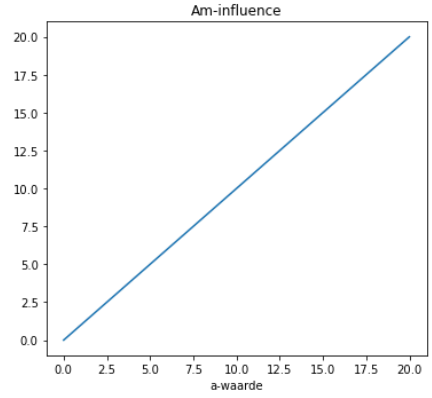

phi-influence
a-waarde


M-influence
a-waarde


V-influence
a-waarde

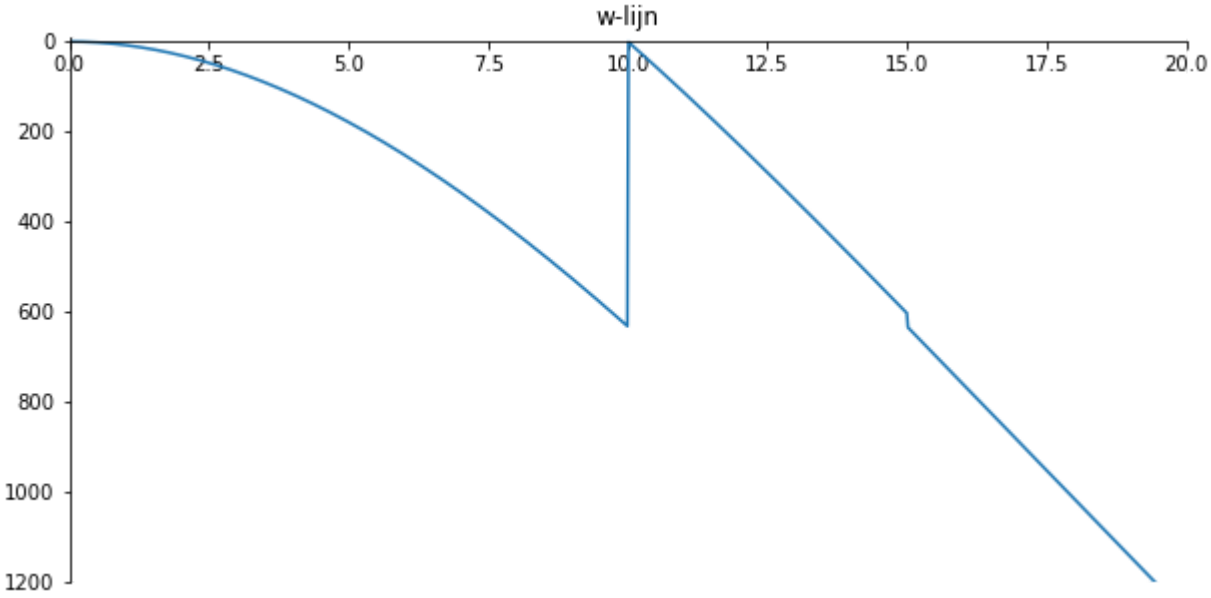

u-influence
a-waarde

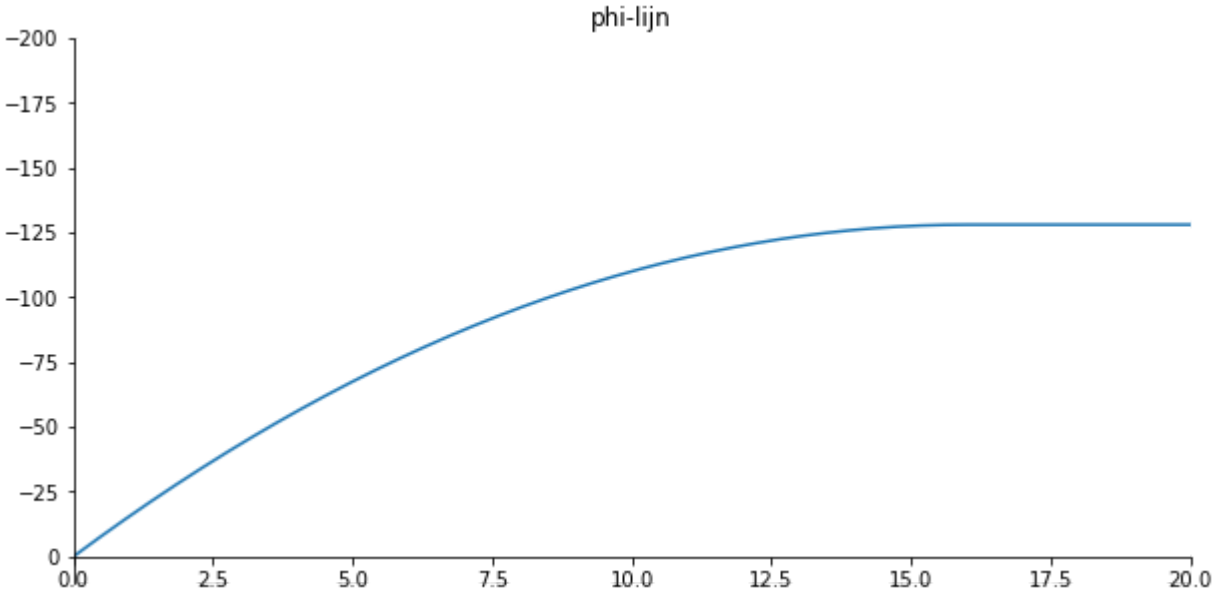

N-influence
a-waarde

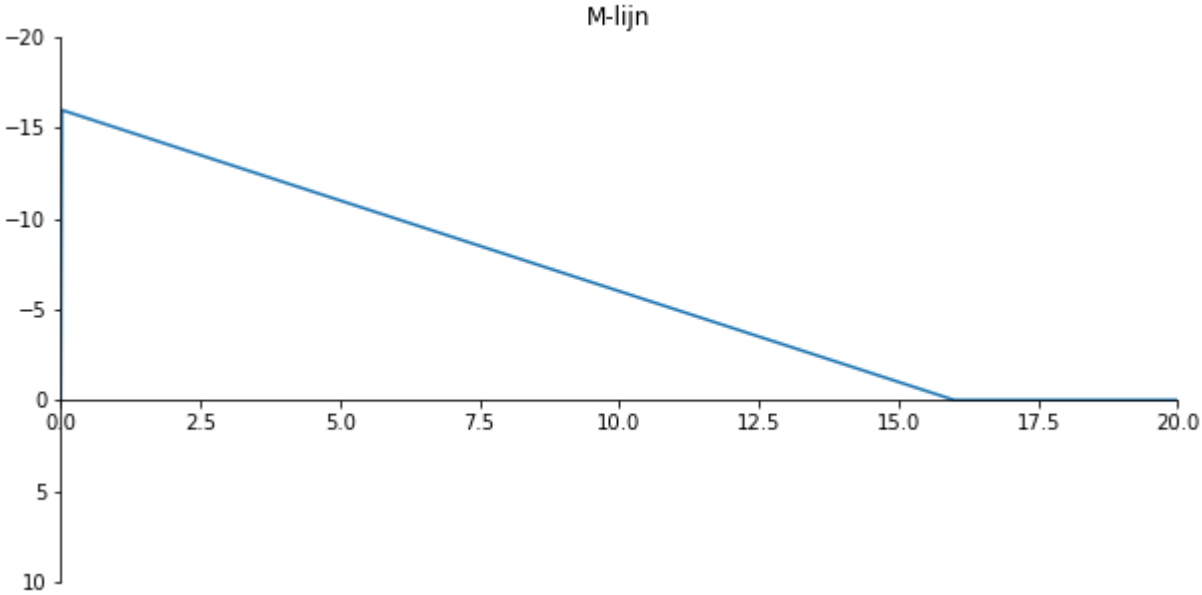

Am-influence
a-waarde

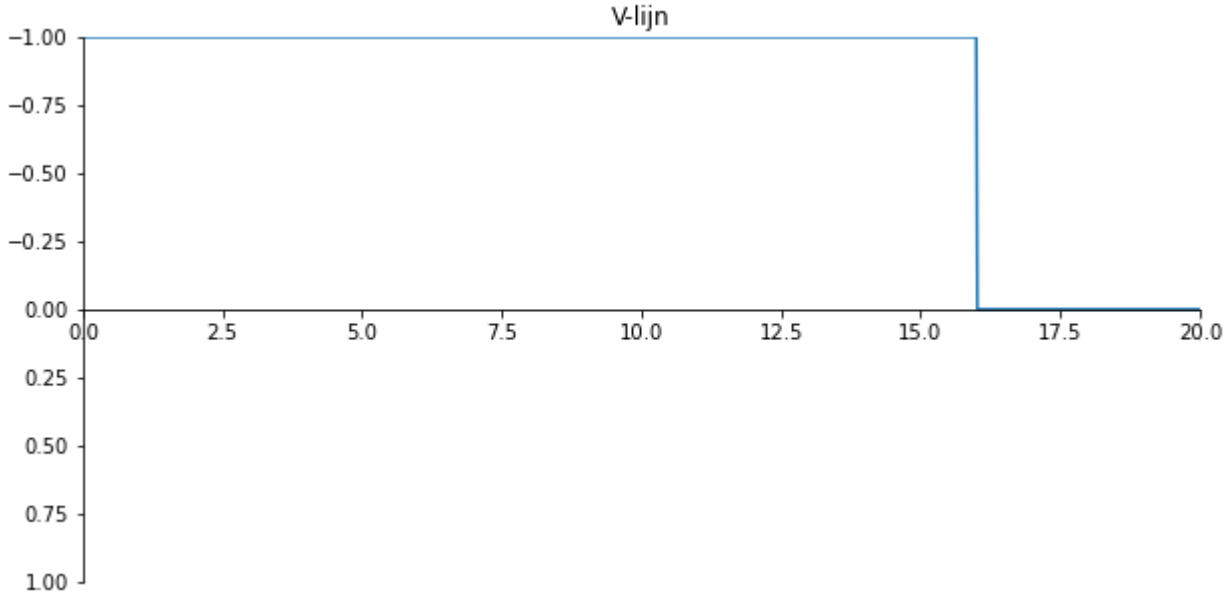
```
a_value = 16

condition1 = And(0 <= a_value, a_value <= 10)
condition2 = And(15 <= a_value, a_value <= 20)

if condition1 or condition2:
    alpha_value = 0
else:
    alpha_value = 90


w_numpy = sp.lambdify(s, w.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
phi_numpy = sp.lambdify(s, phi.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piece
M_numpy = sp.lambdify(s, M.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
V_numpy = sp.lambdify(s, V.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
u_numpy = sp.lambdify(s, U.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
N_numpy = sp.lambdify(s, N.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), w_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1200, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), phi_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([10, -200])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), M_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([10, -20])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), V_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1, -1])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
```

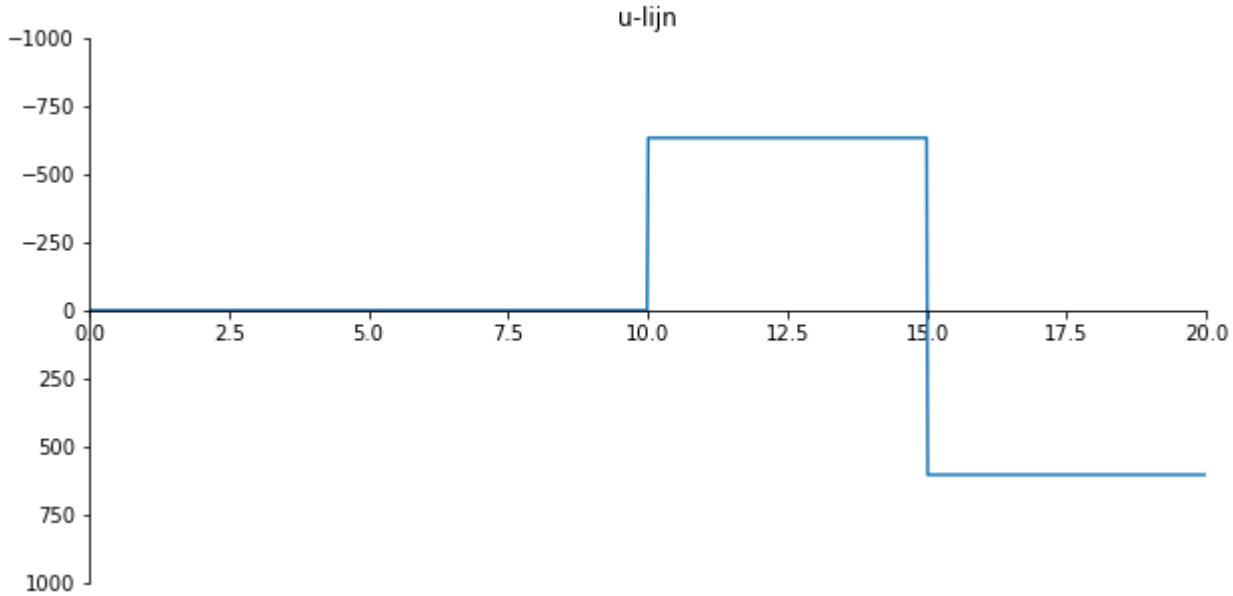
```
ax.xaxis.tick_bottom()
plt.title('V-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), u_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1000, -1000])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), N_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([10, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')

plt.show()
```

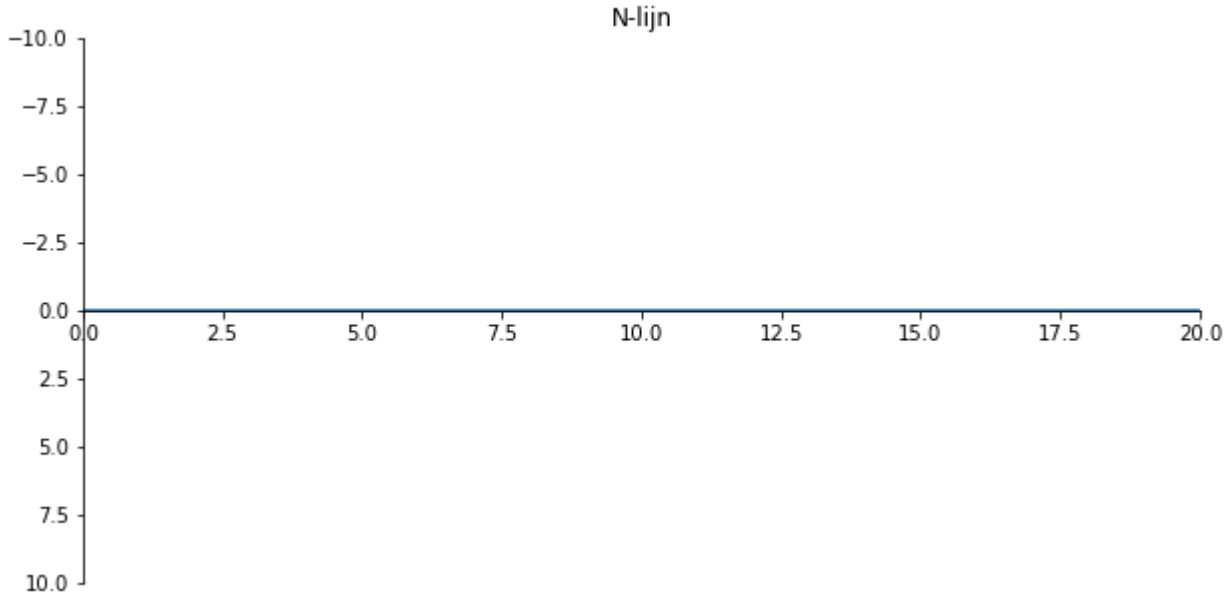

w-lijn
0
200
400
600
800
1000
1200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


phi-lijn
−200
−175
−150
−125
−100
−75
−50
−25
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

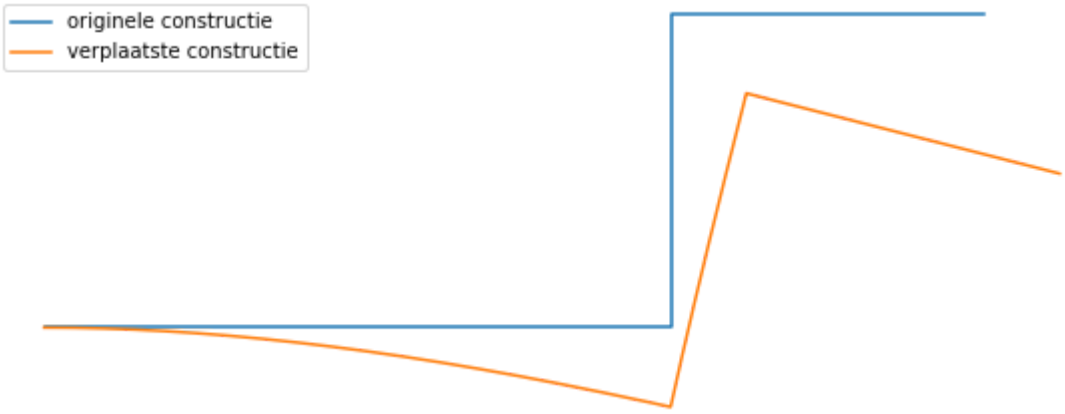

M-lijn
−20
−15
−10
−5
0
5
10
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


V-lijn
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


u-lijn
−1000
−750
−500
−250
0
250
500
750
1000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

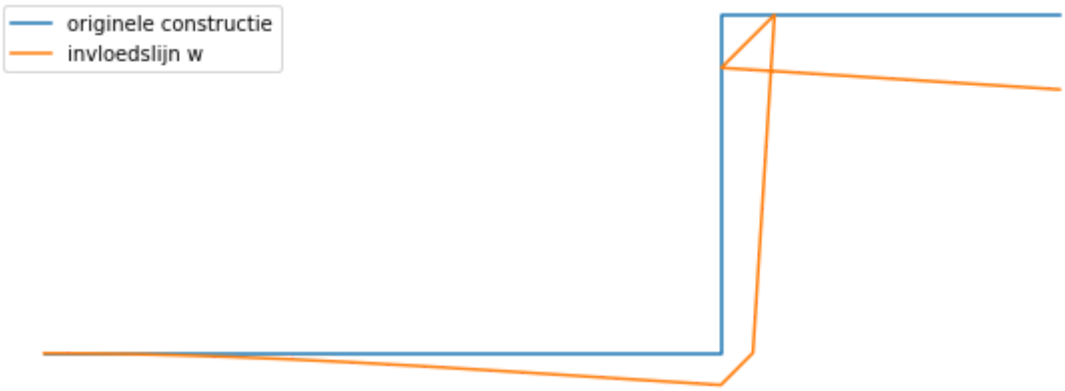

N-lijn
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

```
# voorbeeld 5, 2D plot
factor = sp.symbols('factor')

Am = -sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000) + 20*sp.cos(1745329251
Ah = sp.sin(174532925199433*alpha/10000000000000000)
Bh = 0
Ch = 0
Bv = 0
Av = -sp.cos(174532925199433*alpha/10000000000000000)
Cv = 0
ub = sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) + sf(10, a, 3)*sp.cos(1
wb = sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) - sf(10, a, 3)*sp.cos(1
uc = -2*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) + sf(15, a, 1)*sp.si
wc = sf(10, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/3 - sf(15, a, 1)*sp.sin



w = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) + (
U = ((-Ah*sf(x, 0, 1) - Bh*sf(x, 10, 1) - ub*sf(x, 10, 0) - Ch*sf(x, 15, 1) - uc*sf(x, 15


a_value = 16


u_test = U.subs(factor, 500).subs(a, a_value).subs(alpha, alpha_value)
w_test = w.subs(factor, 500).subs(a, a_value).subs(alpha, alpha_value)


x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * x
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * 10
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * (-x+
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (-5
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * u_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * w_te
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * u_t
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * w_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * -u_t
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * w_t
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)


plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)), z_structure_numpy(np.linspace(0,20,101
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```


originele constructie
verplaatste constructie

```
# Voorbeeld 5, invloedslijn 2D
x_value = 8
condition1 = And(0 <= x_value, x_value <= 10)
condition2 = And(15 <= x_value, x_value <= 20)

if condition1 or condition2:
    alpha_value = 0
else:
    alpha_value = 90


w2 = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) +
wV2 = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) +
phi2 = sp.diff(w2, x)*-1
M2 = sp.diff(phi2, x)
V2 = sp.diff(wV2, x, 3)

u2 = ((-Ah*sf(x, 0, 1) - Bh*sf(x, 10, 1) - ub*sf(x, 10, 0) - Ch*sf(x, 15, 1) - uc*sf(x, 1
N2 = sp.diff(u2, x)

Am2 = Am/factor

u_testt = u2.subs(factor, 5).subs(x, x_value).subs(alpha, alpha_value)
w_testt = w2.subs(factor, 500).subs(x, x_value).subs(alpha, alpha_value)
phi_testt = phi2.subs(factor, 100).subs(x, x_value).subs(alpha, alpha_value)
M_testt = M2.subs(factor, 5).subs(x, x_value).subs(alpha, alpha_value)
V_testt = V2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
N_testt = N2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
Am_testt = Am2.subs(factor, 20).subs(x, x_value).subs(alpha, alpha_value)

u_test = u_testt.subs(a, x)
w_test = w_testt.subs(a, x)
phi_test = phi_testt.subs(a, x)
M_test = M_testt.subs(a, x)
V_test = V_testt.subs(a, x)
N_test = N_testt.subs(a, x)
Am_test = Am_testt.subs(a, x)



x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * x
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * 10
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,15,0)) * (-x+
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (-5
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * 0
        + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, 15, 0)) * w_test
        + (sp.SingularityFunction(x, 15, 0) - sp.SingularityFunction(x, 20, 0)) * 0).expa

z_res = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * w_test
        + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, 15, 0)) * 0
        + (sp.SingularityFunction(x, 15, 0) - sp.SingularityFunction(x, 20, 0)) * w_test)

x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)

plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)), z_structure_numpy(np.linspace(0,20,101
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
```

```
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```



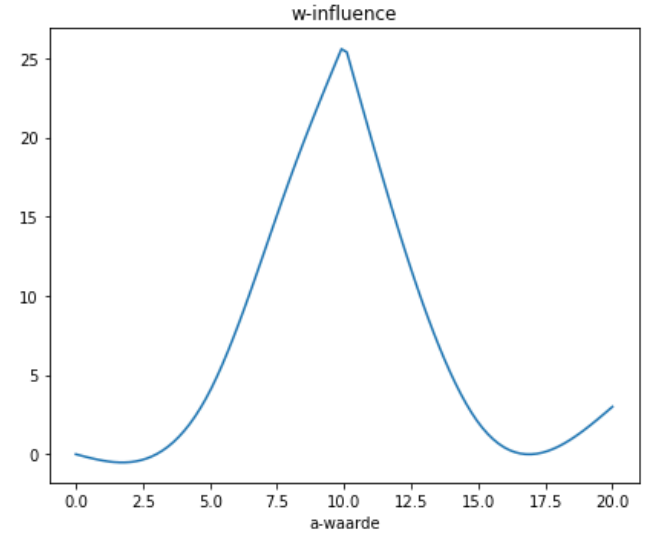

```
#Voorbeeld 6, bepalen van onbekende
Av, Cv, Dv, Bv, Ah, Ch, Dh, Bh, uc, Cphi, wc, ud, phiS, wd, alpha, a = sp.symbols('Av Cv

hoek_rad = sp.rad(alpha)

V20 = -Av - 1*sp.cos(hoek_rad) - Cv - Dv - Bv
M10 = -10*Av - 1*sp.cos(hoek_rad)*sp.Function('sf')(10, a, 1) - 5*Cv
M20 = -20*Av - 1*sp.cos(hoek_rad)*sp.Function('sf')(20, a, 1) - 15*Cv - 5*Dv
N20 = -Ah - Ch - Dh - Bh + 1*sp.sin(hoek_rad)
N5 = -Ah - Ch + 1*sp.sin(hoek_rad) + Av + 1*sp.cos(hoek_rad)
V5 = -Av - 1*sp.cos(hoek_rad) - Cv - Ah + 1*sp.sin(hoek_rad)
N15 = -Ah - Ch - Dh + 1*sp.sin(hoek_rad) + Av + 1*sp.cos(hoek_rad) + Cv
V15 = -Av - 1*sp.cos(hoek_rad) - Cv - Dv - Ah - Ch + 1*sp.sin(hoek_rad)

u5 = -5*Ah - uc + 1*sp.sin(hoek_rad)*sp.Function('sf')(5, a, 1) - (125/6)*Av - (1/6)*sp.c
w5 = (125/6)*Av + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(5, a, 3) + wc - 5*Cphi - 5*Ah
u15 = -15*Ah - 10*Ch - uc - ud + 1*sp.sin(hoek_rad)*sp.Function('sf')(15, a, 1) - (3375/6
w15 = (3375/6)*Av + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(15, a, 3) + (1000/6)*Cv + wc
w20 = (8000/6)*Av + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(20, a, 3) + (3375/6)*Cv + wc
u20 = -20*Ah - 15*Ch - uc - 5*Dh - ud + 1*sp.sin(hoek_rad)*sp.Function('sf')(20, a, 1)

oplossing = sp.solve((V20, M10, M20, N20, N5, V5, N15, V15, u5, w5, u15, w15, w20, u20),
oplossing_in_breuken = {variabele: sp.nsimplify(waarde) for variabele, waarde in oplossin
oplossing_df = pd.DataFrame([(str(variabele), waarde) for variabele, waarde in oplossing_

display(oplossing_df)
```

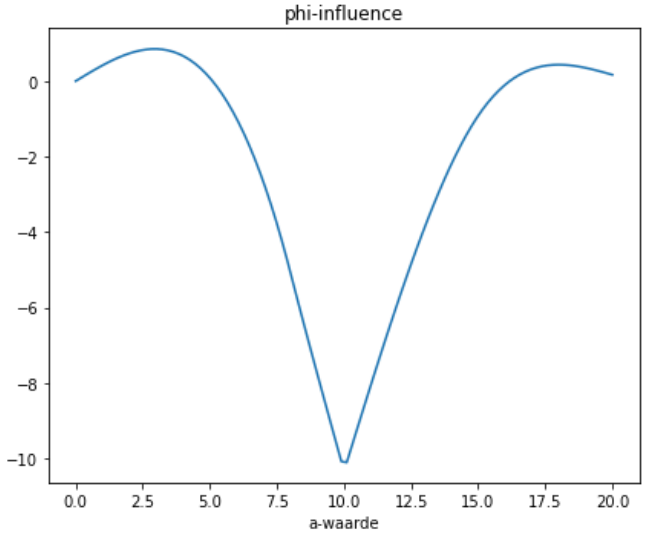
	Variabele	Waarde
0	Av	-sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/10 - cos(174532925199433*alpha/10000000000000000)
1	Cv	sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 + 2*cos(174532925199433*alpha/10000000000000000)
2	Dv	sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 - 2*cos(174532925199433*alpha/10000000000000000)
3	Bv	-sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/10
4	Ah	sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/10 + sin(174532925199433*alpha/10000000000000000) - 2*cos(174532925199433*alpha/10000000000000000)
5	Ch	-sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 + 2*cos(174532925199433*alpha/10000000000000000)
6	Dh	sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/5 + 2*cos(174532925199433*alpha/10000000000000000)
7	Bh	sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/10 - 2*cos(174532925199433*alpha/10000000000000000)
8	uc	-sf(5, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000) + sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(20, a, 1)*sin(174532925199433*alpha/10000000000000000)/2 + 119*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/12 - sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/12 - 10*sin(174532925199433*alpha/10000000000000000) - 250*cos(174532925199433*alpha/10000000000000000)/3
9	Cphi	-sf(5, a, 1)*sin(174532925199433*alpha/10000000000000000)/5 - 19*sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/30 + sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/30 + sf(20, a, 1)*sin(174532925199433*alpha/10000000000000000)/10 + 5*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/2 - sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/60 - sin(174532925199433*alpha/10000000000000000) - 137*cos(174532925199433*alpha/10000000000000000)/6
10	wc	-2*sf(5, a, 1)*sin(174532925199433*alpha/10000000000000000) - sf(5, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000) + sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(20, a, 1)*sin(174532925199433*alpha/10000000000000000)/2 + 131*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/12 - sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/12 - 310*cos(174532925199433*alpha/10000000000000000)/3

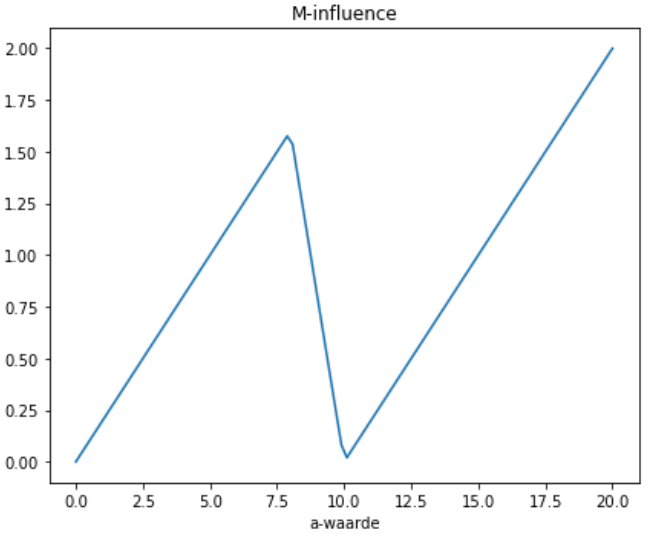
	Variabele	Waarde
11	ud	sf(5, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000) - sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + sf(20, a, 1)*sin(174532925199433*alpha/10000000000000000)/2 - 131*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/12 + sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/12 - 10*sin(174532925199433*alpha/10000000000000000) + 250*cos(174532925199433*alpha/10000000000000000)/3
12	phiS	-sf(5, a, 1)*sin(174532925199433*alpha/10000000000000000)/5 + sf(5, a, 3)*cos(174532925199433*alpha/10000000000000000)/30 + 206*sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000)/15 + sf(15, a, 1)*sin(174532925199433*alpha/10000000000000000)/5 + sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/30 + 19*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/30 - sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/30 - 2*sin(174532925199433*alpha/10000000000000000)
13	wd	-sf(5, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 - sf(10, a, 1)*cos(174532925199433*alpha/10000000000000000) - 2*sf(15, a, 1)*sin(174532925199433*alpha/10000000000000000) + sf(15, a, 3)*cos(174532925199433*alpha/10000000000000000)/6 + 3*sf(20, a, 1)*sin(174532925199433*alpha/10000000000000000)/2 + 143*sf(20, a, 1)*cos(174532925199433*alpha/10000000000000000)/12 - sf(20, a, 3)*cos(174532925199433*alpha/10000000000000000)/12 - 310*cos(174532925199433*alpha/10000000000000000)/3

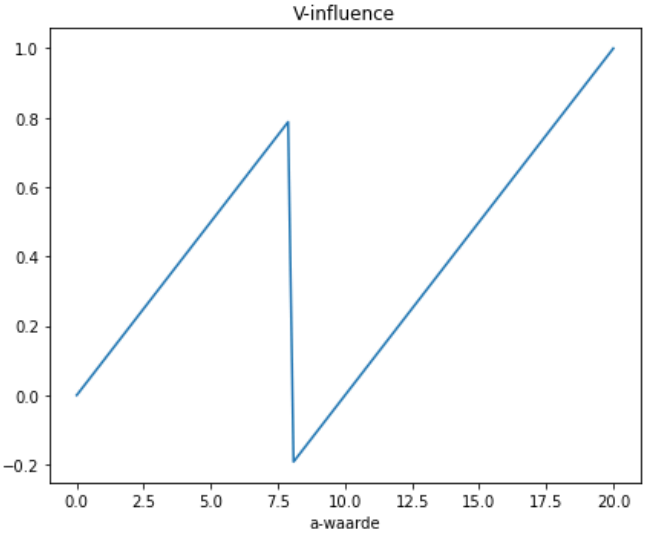
```
# voorbeeld 6, plotten van de invloedslijn

Av = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*sp.co
Cv = sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*sp.cos
Dv= sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - 2*sp.cos(17453292519
Bv = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*sp.co
Ah = sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/10 + sp.sin(17453292519
Ch = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + 2*sp.cos(174532925
Dh = sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*sp.cos
Bh = sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/10 - 2*sp.cos(174532925

uc = -sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*sp.cos
Cphi = -sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/5 - 19*sf(10, a, 1)*s
wc = -2*sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) - sf(5, a, 3)*sp.cos(
ud = sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*sp.cos(
phiS = -sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/5 + sf(5, a, 3)*sp.co
wd = -sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 - sf(10, a, 1)*sp.cos

w = (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (1/6)*Cv*sf(s, 5, 3) + wc
wV = (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (1/6)*Cv*sf(s, 5, 3) + p
phi = sp.diff(w, s)*-1
M = sp.diff(phi, s)
V = sp.diff(wV, s, 3)

U = -Ah*sf(s, 0, 1) - Ch*sf(s, 5, 1) - uc*sf(s, 5, 0) - Dh*sf(s, 15, 1) - ud*sf(s, 15, 0)
N = sp.diff(U, s)

s_value = 8
a_values = np.linspace(0, 20, 100)

condition1 = And(5 <= s_value, s_value <= 15)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90

w_values = [w.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
phi_values = [phi.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in
M_values = [M.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
V_values = [V.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
Av_values = [Av.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
Ah_values = [Ah.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
u_values = [U.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
N_values = [N.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v

plt.figure(figsize=(12, 10))

plt.subplot(221)
plt.plot(a_values, w_values)
plt.title('w-influence')
plt.xlabel('a-waarde')
plt.subplot(222)
plt.plot(a_values, phi_values)
plt.title('phi-influence')
plt.xlabel('a-waarde')
plt.subplot(223)
plt.plot(a_values, M_values)
plt.title('M-influence')
plt.xlabel('a-waarde')
plt.subplot(224)
plt.plot(a_values, V_values)
plt.title('V-influence')
```

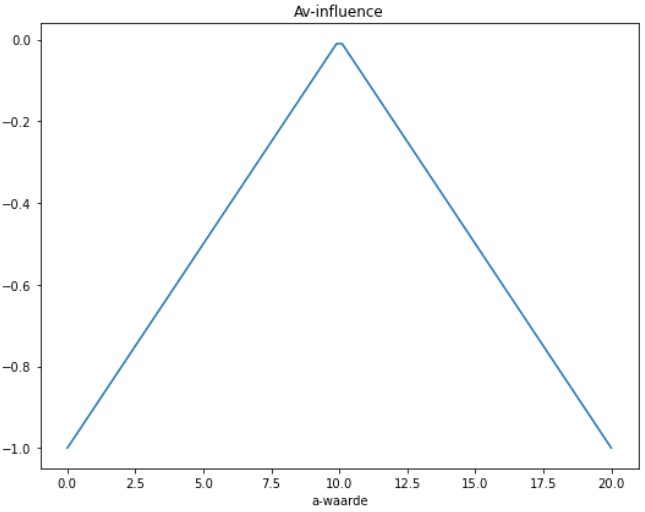
```
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 6))

plt.subplot(121)
plt.plot(a_values, Av_values)
plt.title('Av-influence')
plt.xlabel('a-waarde')
plt.subplot(122)
plt.plot(a_values, Ah_values)
plt.title('Ah-influence')
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 6))

plt.subplot(121)
plt.plot(a_values, u_values)
plt.title('u-influence')
plt.xlabel('a-waarde')
plt.subplot(122)
plt.plot(a_values, N_values)
plt.title('N-influence')
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()
```

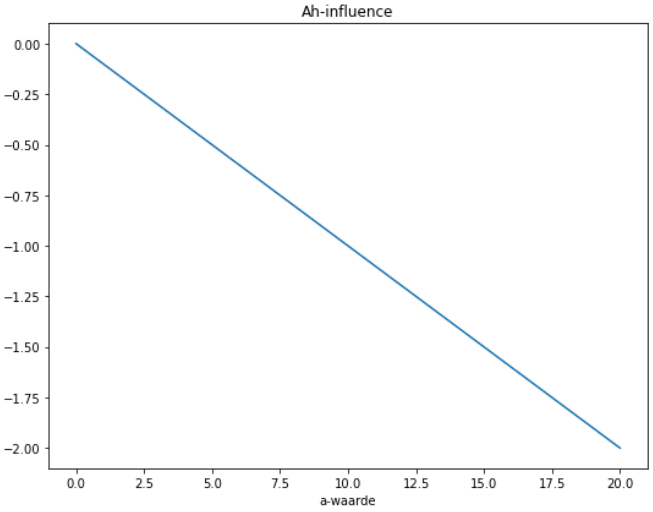

w-influence
a-waarde

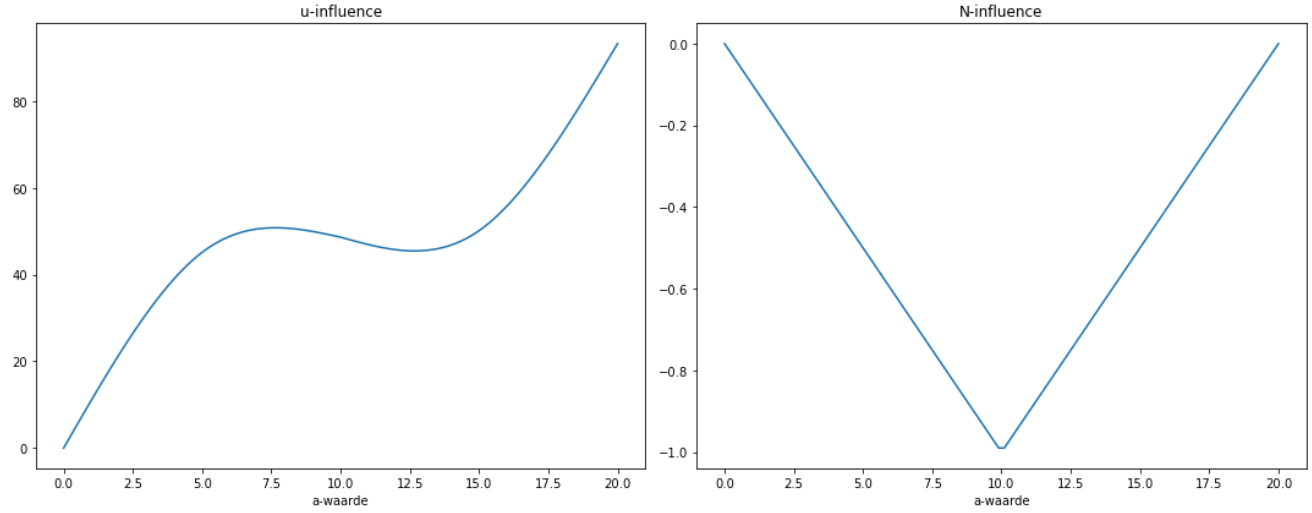

phi-influence
a-waarde


M-influence
a-waarde


V-influence
a-waarde

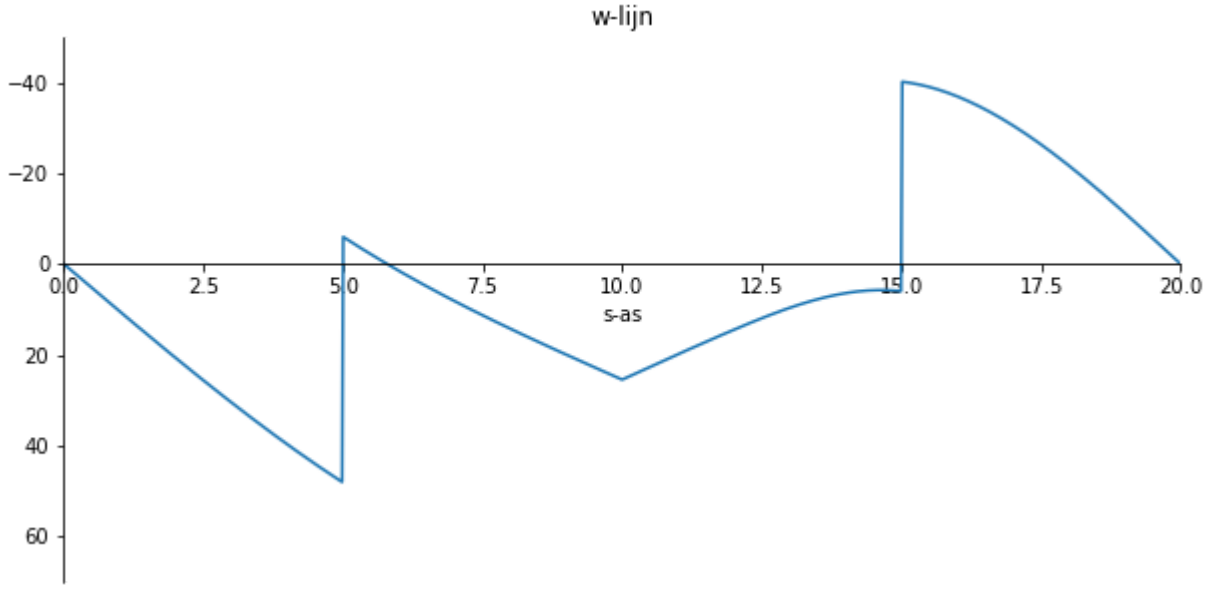

Av-influence
a-waarde

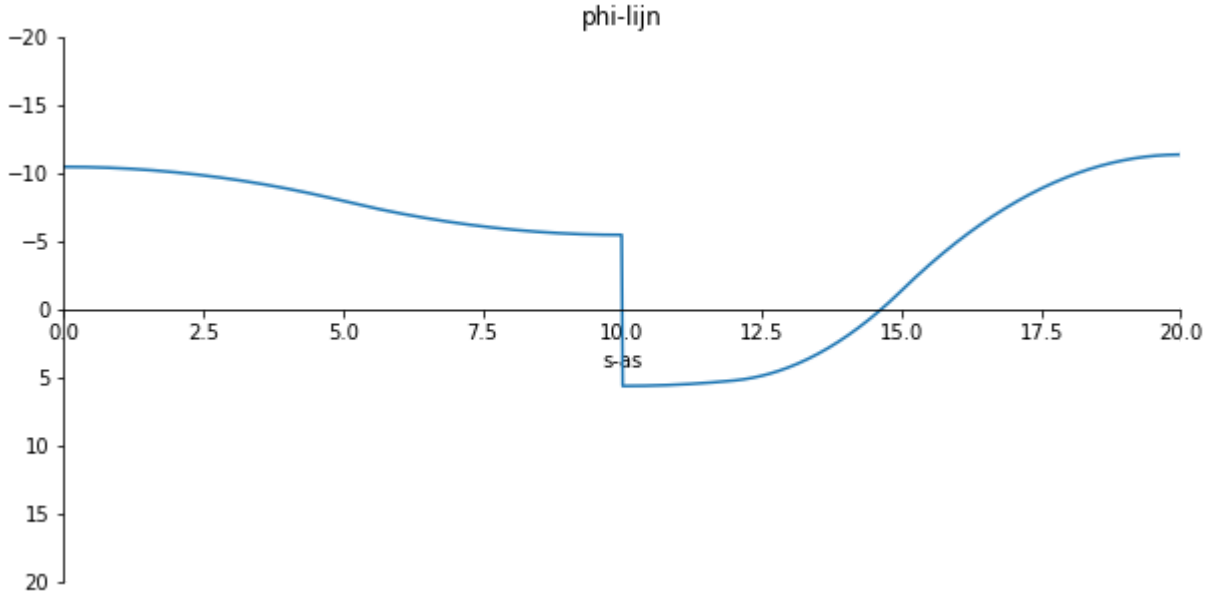

Ah-influence
a-waarde

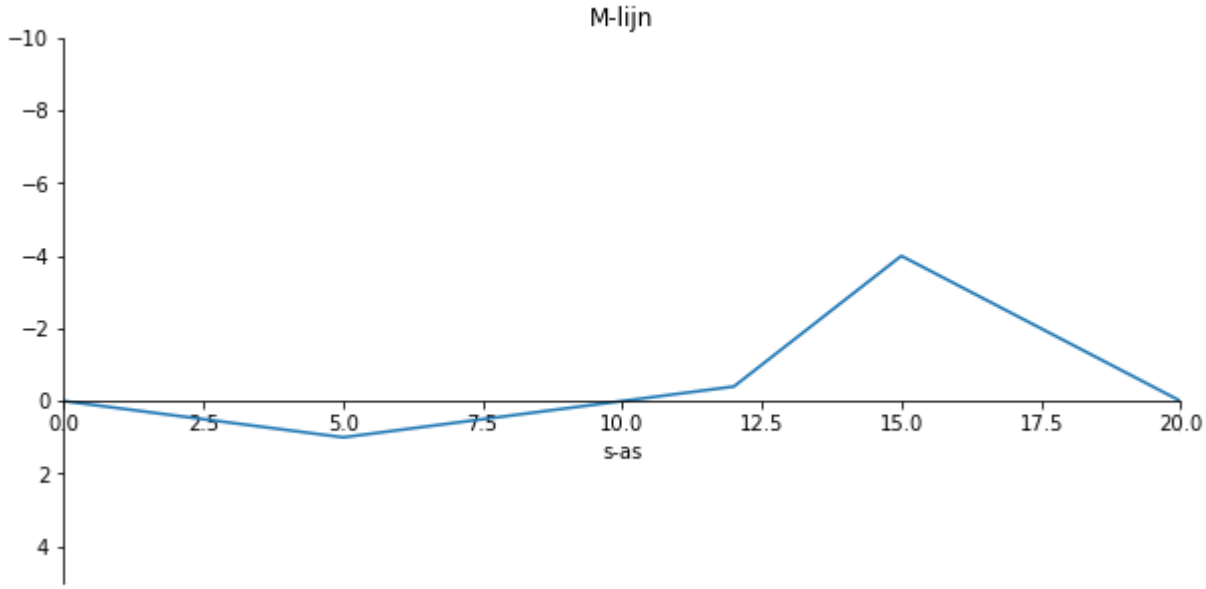

u-influence
80
60
40
20
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
a-waarde
N-influence
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
a-waarde

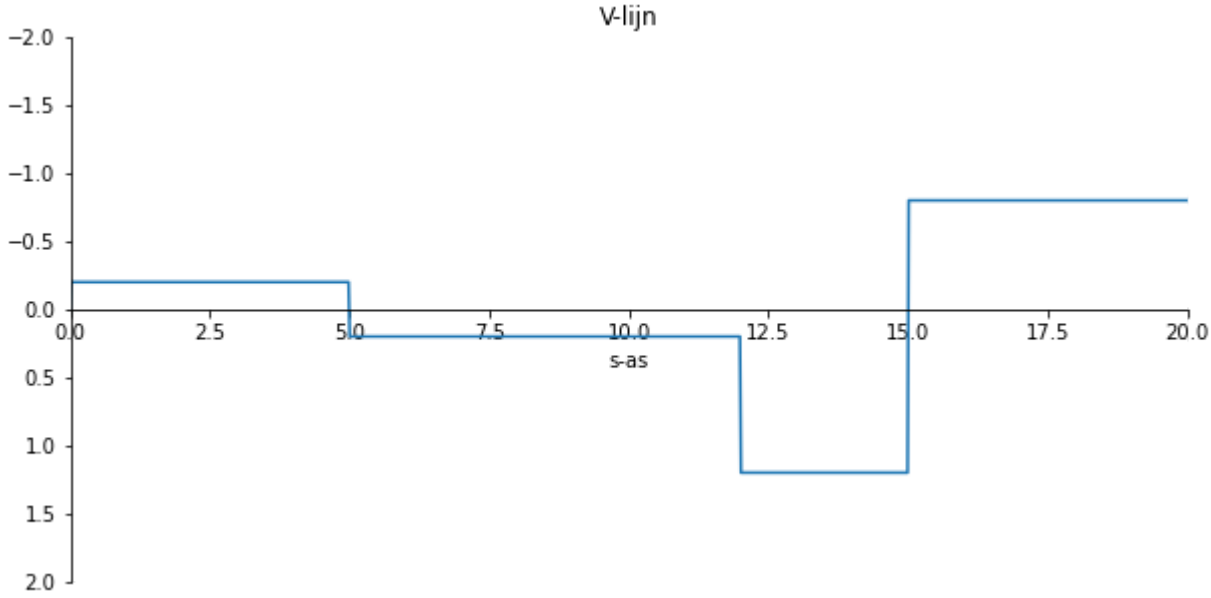
```
#Voorbeeld 6, plotten van de interne krachtslijnen
a_value = 12

condition1 = And(5 <= a_value, a_value <= 15)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90

w_numpy = sp.lambdify(s, w.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
phi_numpy = sp.lambdify(s, phi.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piece
M_numpy = sp.lambdify(s, M.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
V_numpy = sp.lambdify(s, V.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
u_numpy = sp.lambdify(s, U.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
N_numpy = sp.lambdify(s, N.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), w_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([70, -50])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), phi_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([20, -20])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), M_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([5, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), V_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([2, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
```

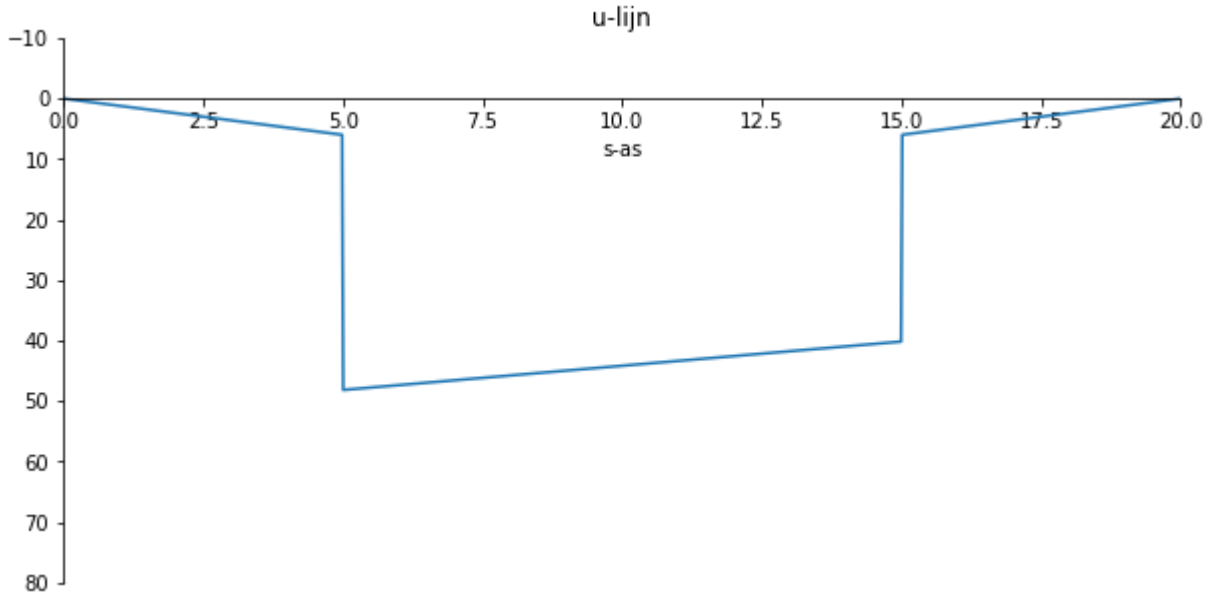
```
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('V-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), u_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([80, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), N_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([3, -3])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')
plt.xlabel('s-as')

plt.show()
```

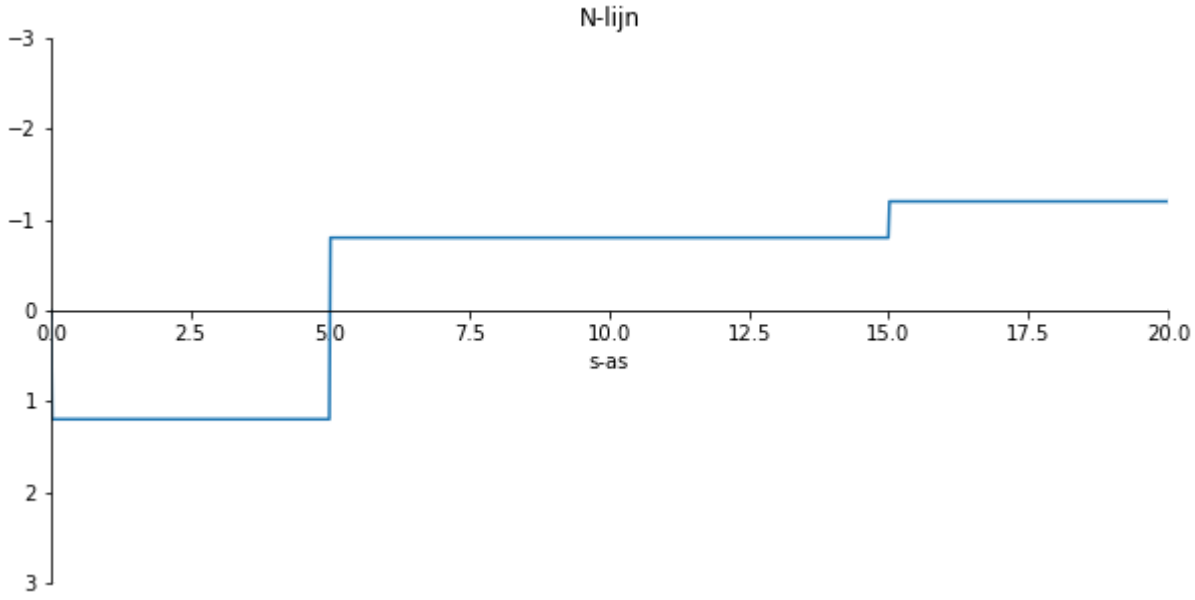

w-lijn
−40
−20
0
20
40
60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


phi-lijn
−20
−15
−10
−5
0
5
10
15
20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

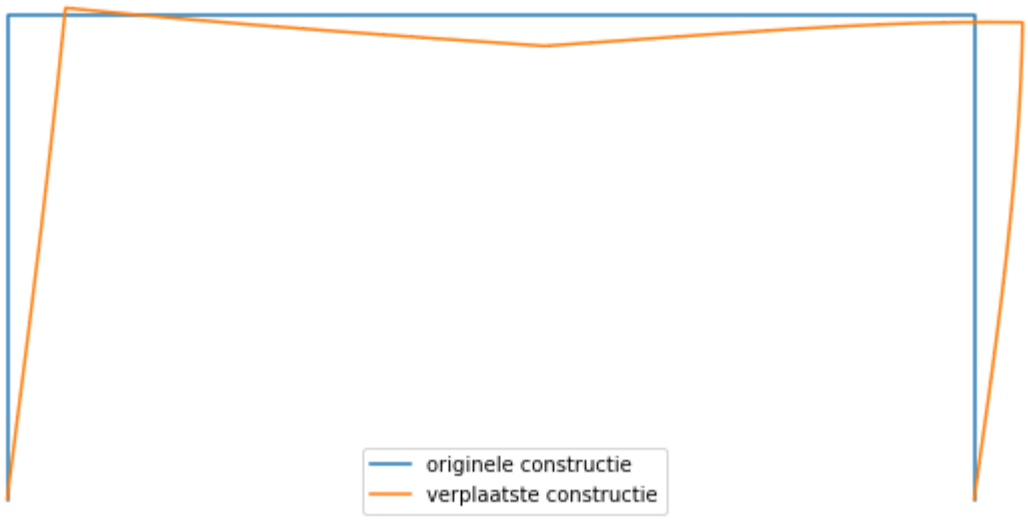

M-lijn
−10
−8
−6
−4
−2
0
2
4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


V-lijn
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


u-lijn
−10
0
10
20
30
40
50
60
70
80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

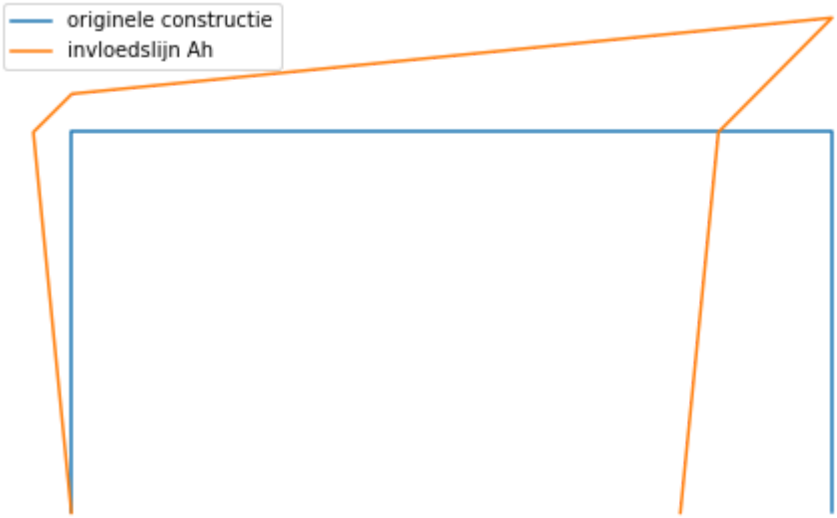

N-lijn
−3
−2
−1
0
1
2
3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

```
# Voorbeeld 6, plotten van de 2D doobuiging
Av = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*sp.co
Cv = sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*sp.cos
Dv= sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - 2*sp.cos(17453292519
Bv = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + sf(20, a, 1)*sp.co
Ah = sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/10 + sp.sin(17453292519
Ch = -sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 + 2*sp.cos(174532925
Dh = sf(10, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/5 - sf(20, a, 1)*sp.cos
Bh = sf(20, a, 1)*sp.cos(174532925199433*alpha/10000000000000000)/10 - 2*sp.cos(174532925

uc = -sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*sp.cos
Cphi = -sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/5 - 19*sf(10, a, 1)*s
wc = -2*sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000) - sf(5, a, 3)*sp.cos(
ud = sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 + sf(10, a, 1)*sp.cos(
phiS = -sf(5, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/5 + sf(5, a, 3)*sp.co
wd = -sf(5, a, 3)*sp.cos(174532925199433*alpha/10000000000000000)/6 - sf(10, a, 1)*sp.cos

w = ((1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) + (1/6)*Cv*sf(x, 5, 3) + w
U = (-Ah*sf(x, 0, 1) - Ch*sf(x, 5, 1) - uc*sf(x, 5, 0) - Dh*sf(x, 15, 1) - ud*sf(x, 15, 0

a_value2 = 12

condition1 = And(5 <= a_value2, a_value2 <= 15)

if condition1:
    alpha_value2 = 0
else:
    alpha_value2 = 90

u2_test = U.subs(a, a_value).subs(alpha, alpha_value2)
w2_test = w.subs(a, a_value).subs(alpha, alpha_value2)


x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * 0
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * (x-5)
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (10
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * -x
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * -5
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * w2_test
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * u2_te
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * -w2
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * -u2_test
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * w2_te
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * u2_
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)


plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)),z_structure_numpy(np.linspace(0,20,101)
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```


originele constructie
verplaatste constructie

```
# voorbeeld 6, 2D plot van invloedslijnen
x_value = 8

condition1 = And(5 <= x_value, a_value <= 15)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90

w2 = ((1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) + (1/6)*Cv*sf(x, 5, 3) +
wV2 = ((1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) + (1/6)*Cv*sf(x, 5, 3) +
phi2 = sp.diff(w2, x)*-1
M2 = sp.diff(phi2, x)
V2 = sp.diff(wV2, x, 3)

U2 = (-Ah*sf(x, 0, 1) - Ch*sf(x, 5, 1) - uc*sf(x, 5, 0) - Dh*sf(x, 15, 1) - ud*sf(x, 15,
N2 = sp.diff(U2, x)

Av2 = Av/factor
Ah2 = Ah/factor

u_testt = U2.subs(factor, 10).subs(x, x_value).subs(alpha, alpha_value)
w_testt = w2.subs(factor, 50).subs(x, x_value).subs(alpha, alpha_value)
phi_testt = phi2.subs(factor, 5).subs(x, x_value).subs(alpha, alpha_value)
M_testt = M2.subs(factor, 2).subs(x, x_value).subs(alpha, alpha_value)
V_testt = V2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
N_testt = N2.subs(factor, 5).subs(x, x_value).subs(alpha, alpha_value)
Av_testt = Av2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
Ah_testt = Ah2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)




u_test = u_testt.subs(a, x)
w_test = w_testt.subs(a, x)
phi_test = phi_testt.subs(a, x)
M_test = M_testt.subs(a, x)
V_test = V_testt.subs(a, x)
N_test = N_testt.subs(a, x)
Av_test = Av_testt.subs(a, x)
Ah_test = Ah_testt.subs(a, x)

x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * 0
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * (x-5)
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (10
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * -x
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * -5
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * (x-
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * Ah_test
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * 0
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * Ah_
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,5,0)) * 0
               + (sp.SingularityFunction(x,5,0) - sp.SingularityFunction(x,15,0)) * Ah_te
                + (sp.SingularityFunction(x,15,0) - sp.SingularityFunction(x,20,0)) * 0).
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)


plt.figure(figsize=(10, 5))
```

```
plt.plot(x_structure_numpy(np.linspace(0,20,101)), z_structure_numpy(np.linspace(0,20,101
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```




```
# voorbeeld 7, bepalen van de onbekenden
Am, Av, Cv, Bv, Ah, Ch, Bh, uc, wc, alpha, a = sp.symbols('Am, Av, Cv, Bv, Ah, Ch, Bh, uc

hoek_rad = sp.rad(alpha)

V20 = -Av - 1*sp.cos(hoek_rad) - Cv - Bv
M20 = -Am - 20*Av - 10*Cv - 1*sp.cos(hoek_rad)*sp.Function('sf')(20, a, 1)
N20 = -Ah - Ch - Bh + 1*sp.sin(hoek_rad)
w20 = 200*Am + (8000/6)*Av + (1000/6)*Cv + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(20, a
u20 = -20*Ah -10*Ch - uc + 1*sp.sin(hoek_rad)*sp.Function('sf')(20, a, 1)
N10 = 1*sp.sin(hoek_rad) -Ah - Ch + Av + 1*sp.cos(hoek_rad)
V10 = -Av -Cv - 1*sp.cos(hoek_rad) - Ah - 1*sp.sin(hoek_rad)
u10 = 1*sp.sin(hoek_rad)*sp.Function('sf')(10, a, 1) -10*Ah - uc - 50*Am - (1000/6)*Av -
w10 = 50*Am + (1000/6)*Av + wc + (1/6)*sp.cos(hoek_rad)*sp.Function('sf')(10, a, 3) - 10*

oplossing = sp.solve((V20, M20, N20, N20, u20, w20, N10, V10, u10, w10), (Am, Av, Cv, Bv,
oplossing_in_breuken = {variabele: sp.nsimplify(waarde) for variabele, waarde in oplossin
oplossing_df = pd.DataFrame([(str(variabele), waarde) for variabele, waarde in oplossing_

pd.set_option('display.latex.repr', True)
pd.set_option('display.max_colwidth', None)

display(oplossing_df)
```

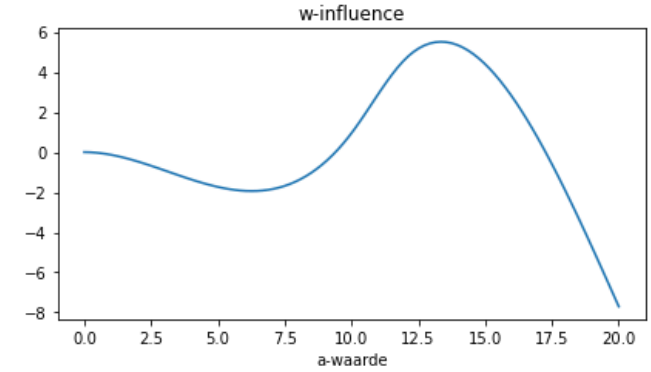
	Variabele	
0	Am	3251091 1)*sin(174532925199433*alpha/10000000000000000)/100 3945499 3)*cos(174532925199433*alpha/10000000000000000)/500 3992454 1)*sin(174532925199433*alpha/10000000000000000)/100 290520 1)*cos(174532925199433*alpha/10000000000000000)/25 618128 3)*cos(174532925199433*alpha/10000000000000000)/5000 325109158819507*sin(174532925199433*alpha/10000000000000000)/1 325109158819507*cos(174532925199433*alpha/10000000000000000)/1 -3992454 1)*sin(174532925199433*alpha/10000000000000000)/10000 1454574 3)*cos(174532925199433*alpha/10000000000000000)/10000 6360145 1)*sin(174532925199433*alpha/10000000000000000)/10000 3708769 1)*cos(174532925199433*alpha/10000000000000000)/100 3945499 3)*cos(174532925199433*alpha/10000000000000000)/10000 399284549423957*sin(174532925199433*alpha/10000000000000000)/10 399284549423957*cos(174532925199433*alpha/10000000000000000)/10 591824 1)*sin(174532925199433*alpha/10000000000000000)/125 26500 3)*cos(174532925199433*alpha/10000000000000000)/12 1745488 1)*sin(174532925199433*alpha/10000000000000000)/200 1625545 1)*cos(174532925199433*alpha/10000000000000000)/100 831842 3)*cos(174532925199433*alpha/10000000000000000)/12500 59182492503551*sin(174532925199433*alpha/10000000000000000)/125 59182492503551*cos(174532925199433*alpha/10000000000000000)/1 -1483507 1)*sin(174532925199433*alpha/10000000000000000)/20000 831842 3)*cos(174532925199433*alpha/10000000000000000)/12500 591824 1)*sin(174532925199433*alpha/10000000000000000)/2500 156833 1)*cos(174532925199433*alpha/10000000000000000)/1 541848 3)*cos(174532925199433*alpha/10000000000000000)/2000 148350781208901*sin(174532925199433*alpha/10000000000000000)/20 - 25185438476511*cos(174532925199433*alpha/10000000000000000) -1483507 1)*sin(174532925199433*alpha/10000000000000000)/20000
1	Av	
2	Cv	
3	Bv	
4	Ah	

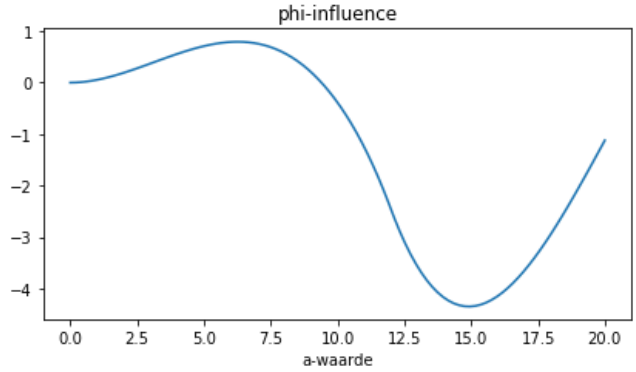
	Variabele	
5	Ch	831842 3)*cos(174532925199433*alpha/10000000000000000)/1250 591824 1)*sin(174532925199433*alpha/10000000000000000)/250 156833 1)*cos(174532925199433*alpha/10000000000000000)/1 541848 3)*cos(174532925199433*alpha/10000000000000000)/2000 25185438476511*sin(174532925199433*alpha/10000000000000000)/ 25185438476511*cos(174532925199433*alpha/10000000000000000) -3251091 1)*sin(174532925199433*alpha/10000000000000000)/1000 3945499 3)*cos(174532925199433*alpha/10000000000000000)/5000 3992845 1)*sin(174532925199433*alpha/10000000000000000)/1000 8837918 1)*cos(174532925199433*alpha/10000000000000000)/100 618128 3)*cos(174532925199433*alpha/10000000000000000)/5000 39349781682361*sin(174532925199433*alpha/10000000000000000)/2 39349781682361*cos(174532925199433*alpha/10000000000000000) 3992845 1)*sin(174532925199433*alpha/10000000000000000)/1000 1454574 3)*cos(174532925199433*alpha/10000000000000000)/1000 6360145 1)*sin(174532925199433*alpha/10000000000000000)/1000 3708769 1)*cos(174532925199433*alpha/10000000000000000)/100 3945499 3)*cos(174532925199433*alpha/10000000000000000)/10000 399284549423957*sin(174532925199433*alpha/10000000000000000)/10 - 240017886264401*cos(174532925199433*alpha/10000000000000000)/ 591824 1)*sin(174532925199433*alpha/10000000000000000)/12 26500 3)*cos(174532925199433*alpha/10000000000000000)/12 9127255 1)*sin(174532925199433*alpha/10000000000000000)/10 1625545 1)*cos(174532925199433*alpha/10000000000000000)/10 831842 3)*cos(174532925199433*alpha/10000000000000000)/1250 59182492503551*sin(174532925199433*alpha/10000000000000000)/12 59182492503551*cos(174532925199433*alpha/10000000000000000)/
6	Bh	
7	uc	
8	wc	-393497 1)*sin(174532925199433*alpha/10000000000000000)/ 3945499 3)*cos(174532925199433*alpha/10000000000000000)/5000 2400178 1)*sin(174532925199433*alpha/10000000000000000)/25

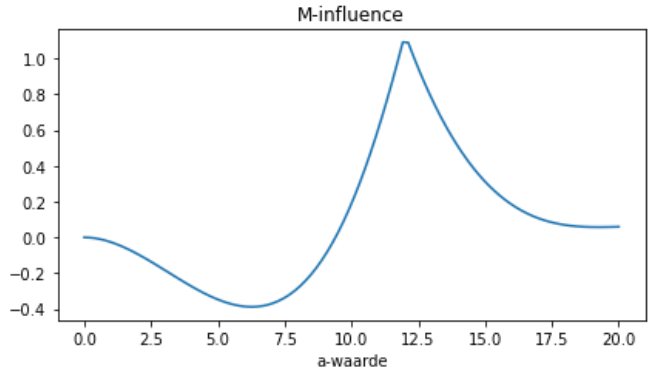
	Variabele	
		2209479
1)*cos(174532925199433*alpha/10000000000000000)/25
618128
3)*cos(174532925199433*alpha/10000000000000000)/500
39349781682361*sin(174532925199433*alpha/10000000000000000)
39349781682361*cos(174532925199433*alpha/1000000000000000

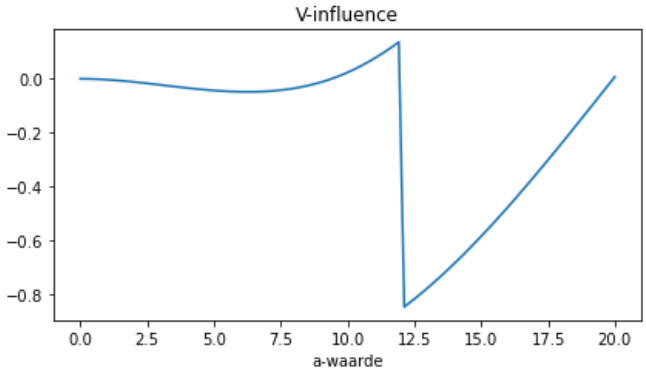
```
# Voorbeeld 7, plotten van de invloedslijnen
Am = 325109158819507*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/1000000
Av = -399284549423957*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/100000
Cv = 59182492503551*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/12500000
Bv = -148350781208901*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/200000
Ah = -148350781208901*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/200000
Ch = -325109158819507*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/100000
Bh = 399284549423957*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/1000000
uc = 59182492503551*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/12500000
wc = -39349781682361*sf(10, a, 1)*sp.sin(174532925199433*alpha/1000000000000000)/2000000


w = (1/2)*Am*sf(s, 0, 2) + (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (1
wV = (1/2)*Am*sf(s, 0, 2) + (1/6)*Av*sf(s, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(s, a, 3) + (
phi = sp.diff(w, s)*-1
M = sp.diff(phi, s)
V = sp.diff(wV, s, 3)

U = -Ah*sf(s, 0, 1) - Ch*sf(s, 10, 1) - uc*sf(s, 10, 0) - Bh*sf(s, 20, 1) + 1*sp.sin(hoek
N = sp.diff(U, s)

s_value = 12
a_values = np.linspace(0, 20, 100)

condition1 = And(10 <= s_value, s_value <= 20)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90

w_values = [w.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
phi_values = [phi.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in
M_values = [M.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
V_values = [V.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
Am_values = [Am.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
Av_values = [Av.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
Ah_values = [Ah.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a
u_values = [U.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v
N_values = [N.subs([(s, s_value), (a, a_value), (alpha, alpha_value)]) for a_value in a_v

plt.figure(figsize=(12, 7))

plt.subplot(221)
plt.plot(a_values, w_values)
plt.title('w-influence')
plt.xlabel('a-waarde')
plt.subplot(222)
plt.plot(a_values, phi_values)
plt.title('phi-influence')
plt.xlabel('a-waarde')
plt.subplot(223)
plt.plot(a_values, M_values)
plt.title('M-influence')
plt.xlabel('a-waarde')
plt.subplot(224)
plt.plot(a_values, V_values)
plt.title('V-influence')
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()
```

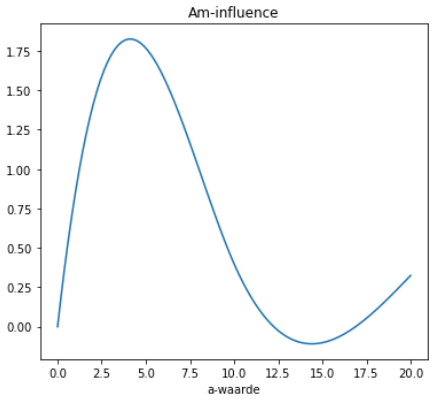
```
plt.figure(figsize=(16, 5))

plt.subplot(131)
plt.plot(a_values, Am_values)
plt.title('Am-influence')
plt.xlabel('a-waarde')
plt.subplot(132)
plt.plot(a_values, Av_values)
plt.title('Av-influence')
plt.xlabel('a-waarde')
plt.subplot(133)
plt.plot(a_values, Ah_values)
plt.title('Ah-influence')
plt.xlabel('a-waarde')

plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 6))

plt.subplot(121)
plt.plot(a_values, u_values)
plt.title('u-influence')
plt.xlabel('a-waarde')
plt.subplot(122)
plt.plot(a_values, N_values)
plt.title('N-influence')
plt.xlabel('a-waarde')


plt.tight_layout()
plt.show()
```

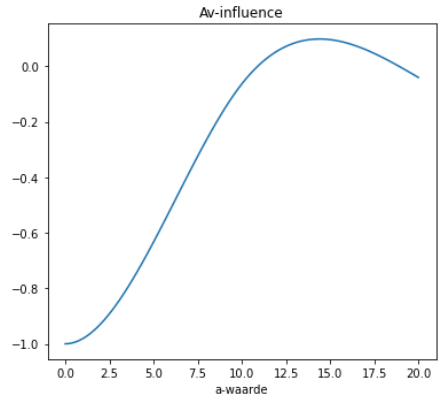

w-influence
a-waarde

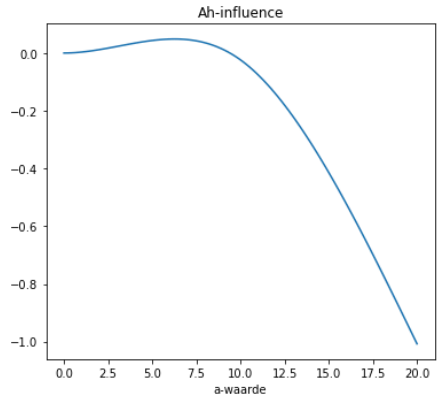

phi-influence
a-waarde

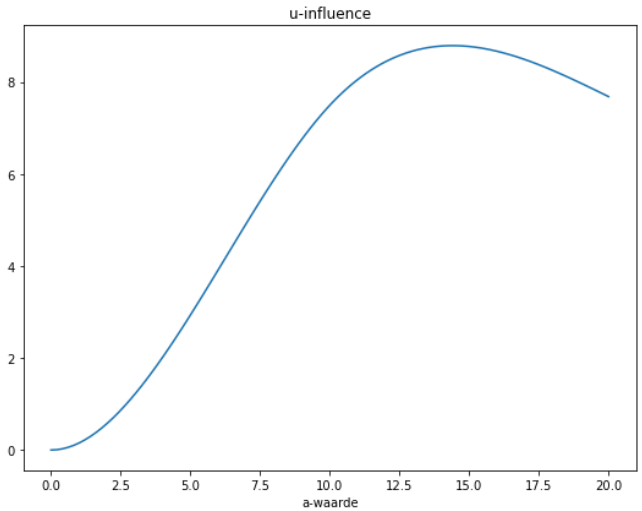

M-influence
a-waarde

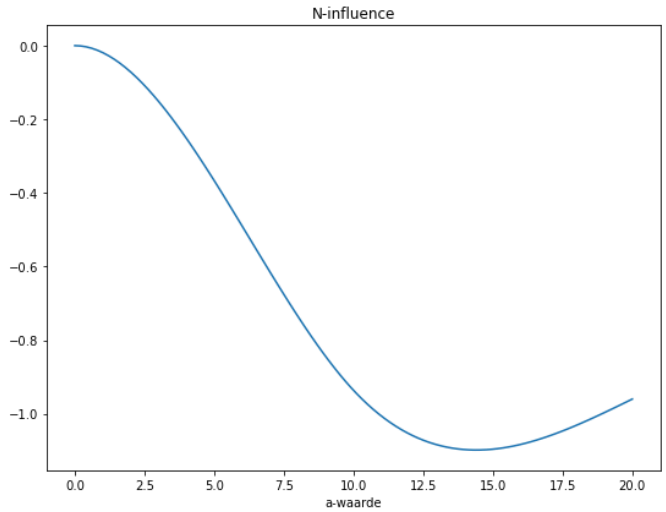

V-influence
a-waarde


Am-influence
a-waarde


Av-influence
a-waarde

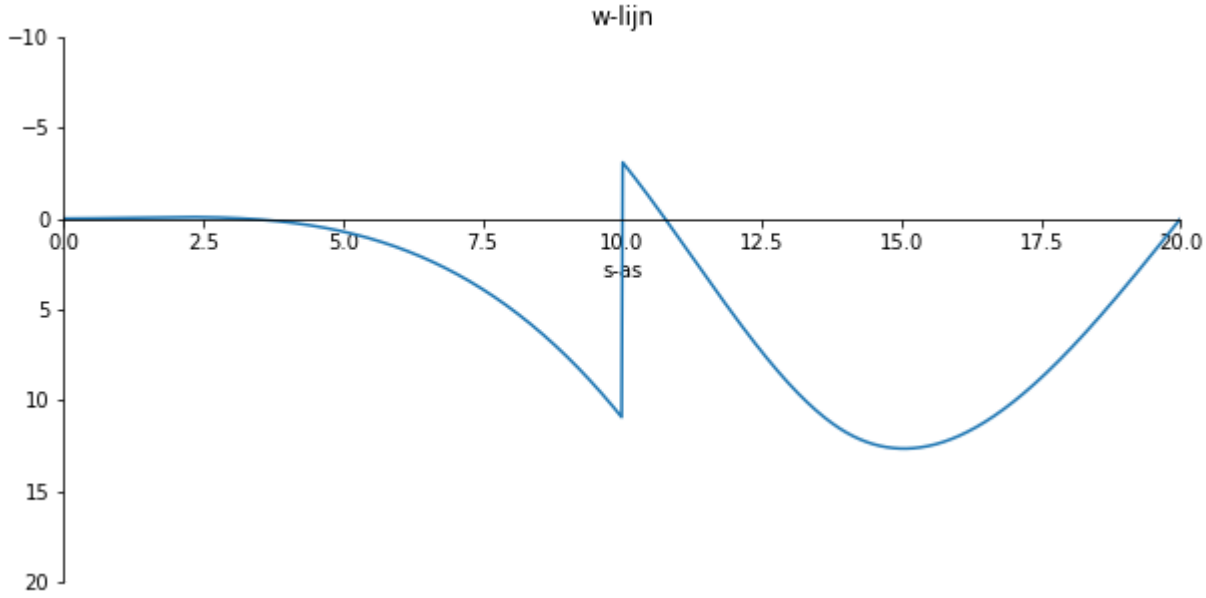

Ah-influence
a-waarde

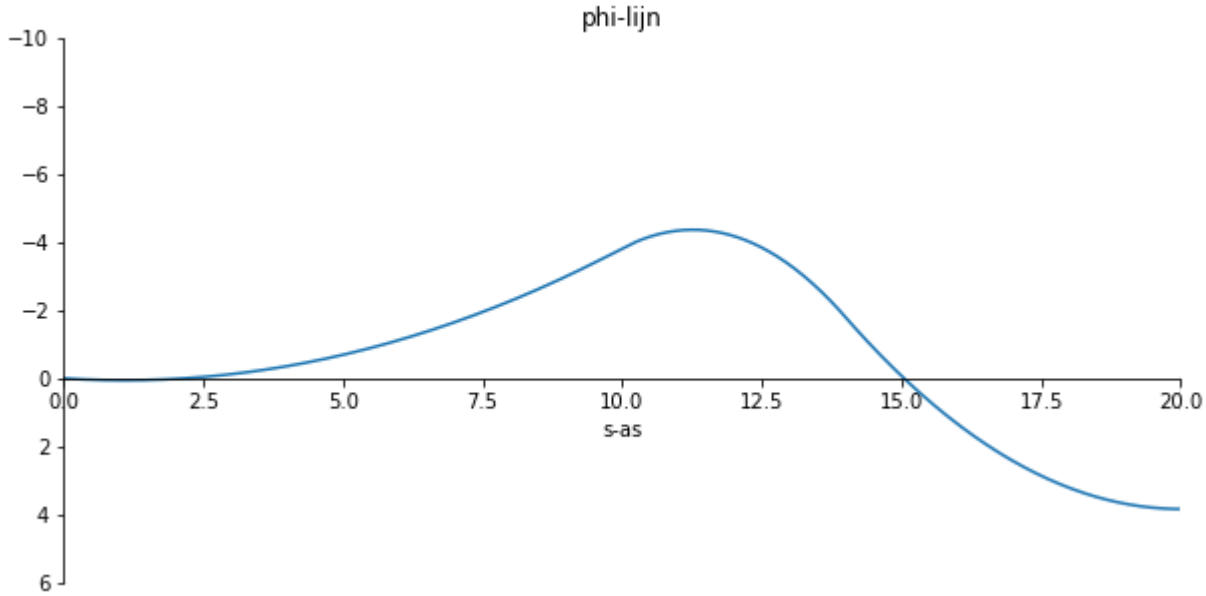

u-influence
a-waarde

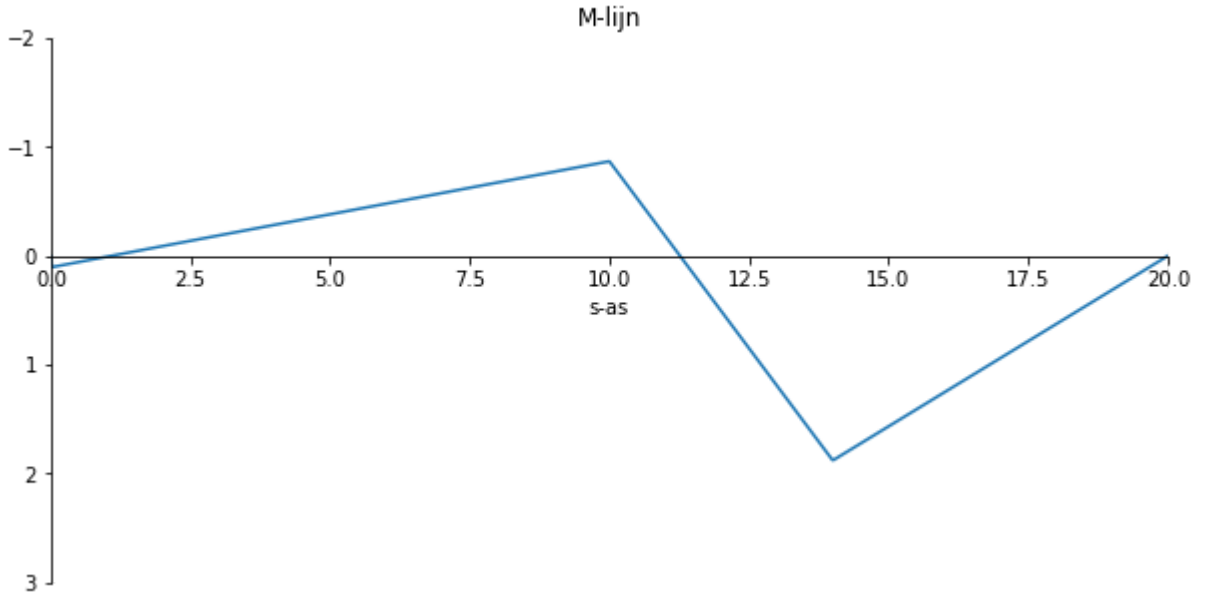

N-influence
a-waarde

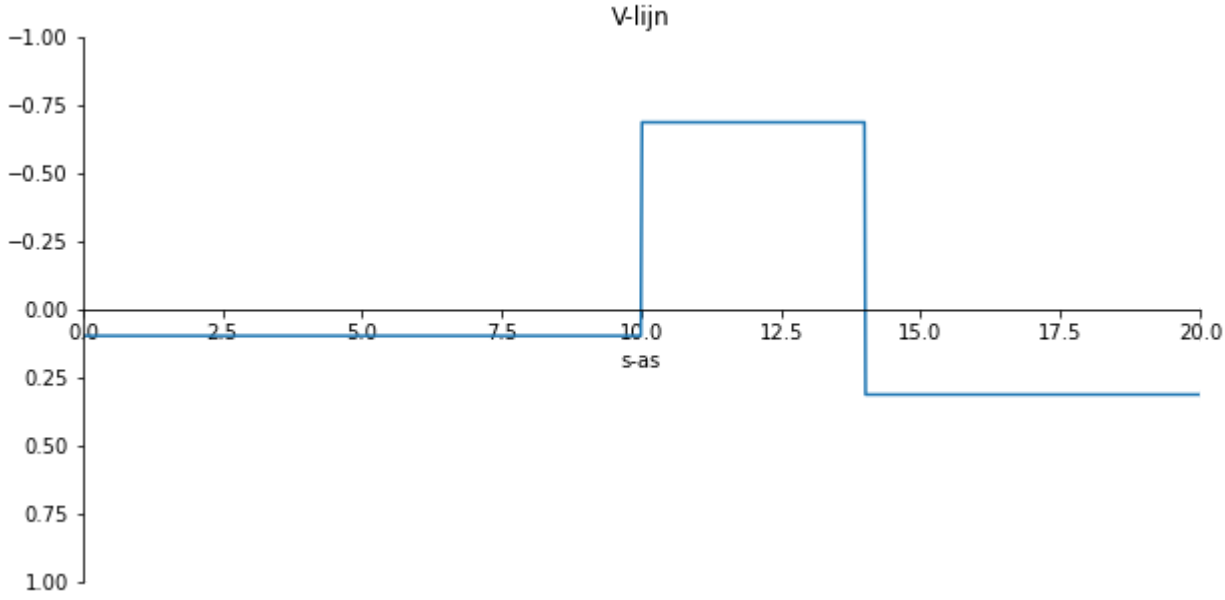
```
#Voorbeeld 7, plotten van de interne krachtslijnen
a_value = 14
condition1 = And(10 <= a_value, a_value <= 20)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90

w_numpy = sp.lambdify(s, w.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
phi_numpy = sp.lambdify(s, phi.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piece
M_numpy = sp.lambdify(s, M.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
V_numpy = sp.lambdify(s, V.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
u_numpy = sp.lambdify(s, U.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise
N_numpy = sp.lambdify(s, N.subs(a, a_value).subs(alpha, alpha_value).rewrite(sp.Piecewise

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), w_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([20, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), phi_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([6, -10])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), M_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([3, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), V_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1, -1])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
```

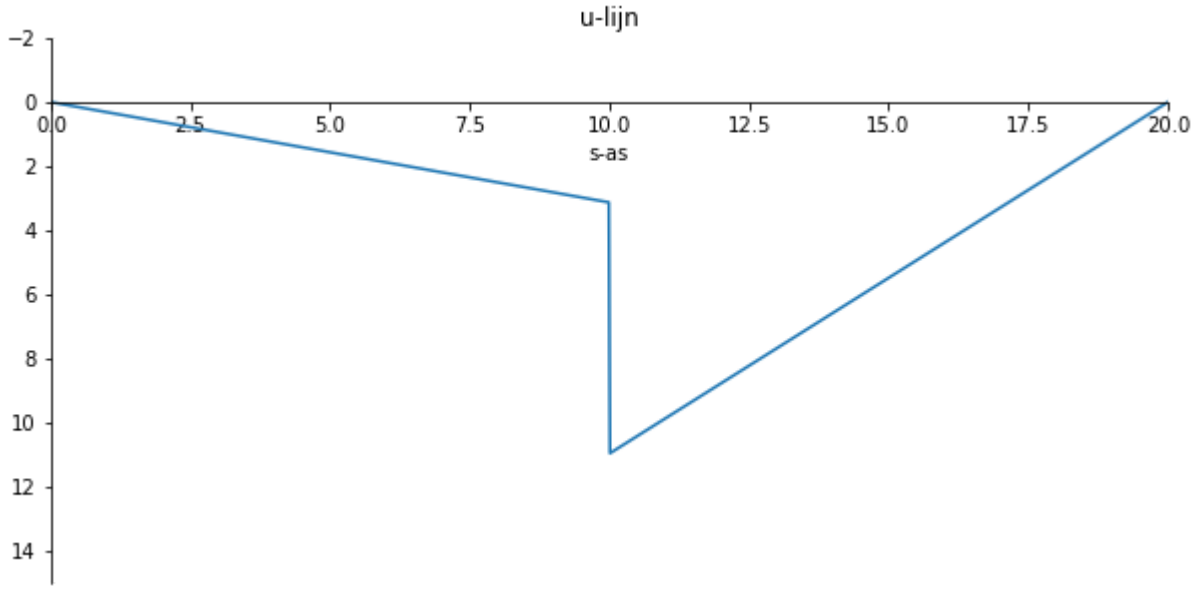
```
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('V-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), u_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([15, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')
plt.xlabel('s-as')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, 20, 1000), N_numpy(np.linspace(0, 20, 1000)))
plt.xlim([0, 20])
plt.ylim([1, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')
plt.xlabel('s-as')

plt.show()
```

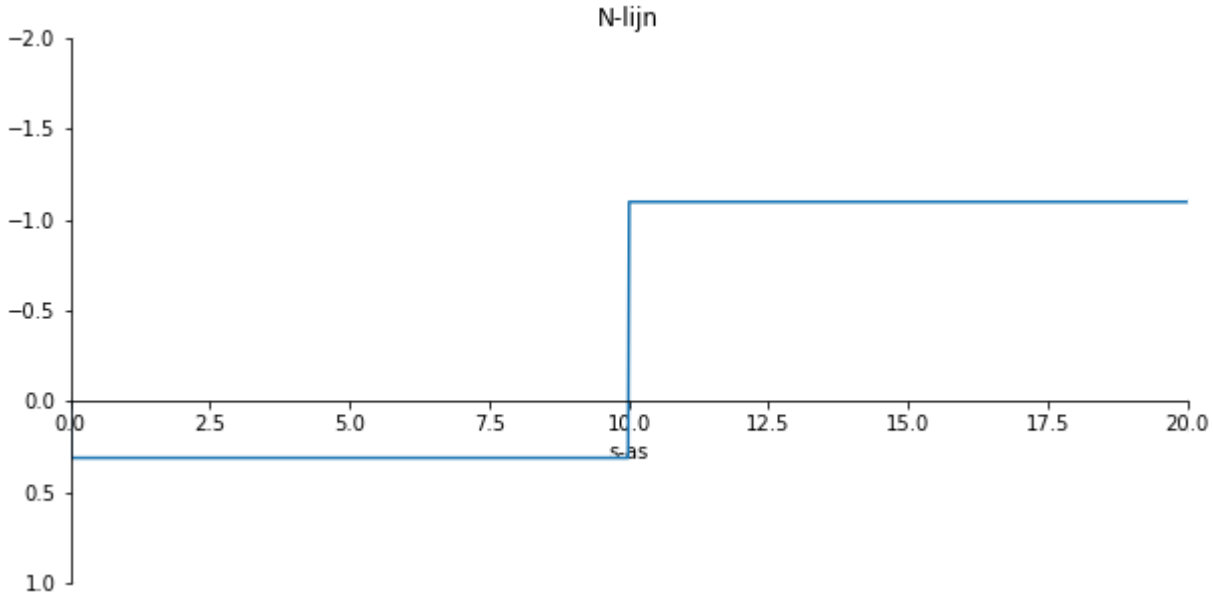

w-lijn
−10
−5
0
5
10
15
20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


phi-lijn
−10
−8
−6
−4
−2
0
2
4
6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

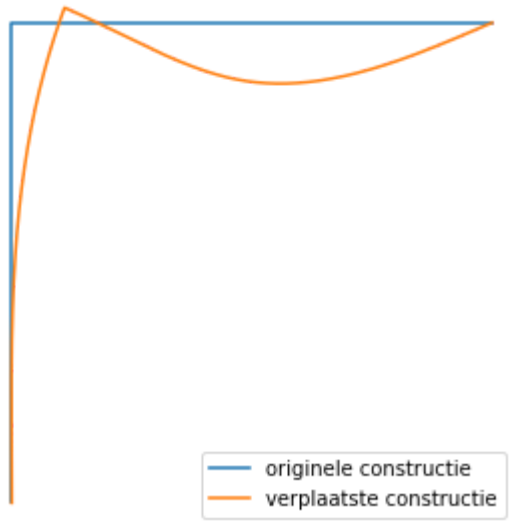

M-lijn
−2
−1
0
1
2
3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


V-lijn
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as


u-lijn
−2
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

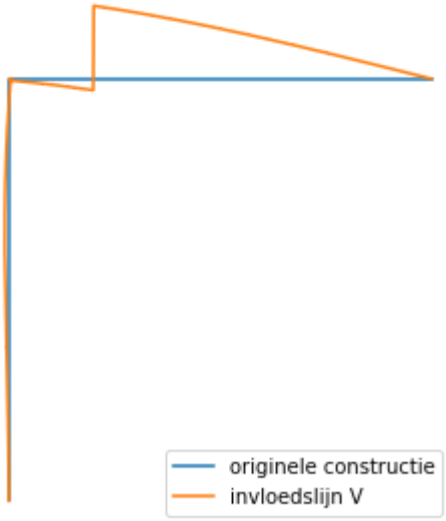

N-lijn
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
s-as

```
#Voorbeeld 7, plotten 2D van de doorbuiging

Am = 325109158819507*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/1000000
Av = -399284549423957*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/100000
Cv = 59182492503551*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/12500000
Bv = -148350781208901*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/200000
Ah = -148350781208901*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/200000
Ch = -325109158819507*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/100000
Bh = 399284549423957*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/1000000
uc = 59182492503551*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/12500000
wc = -39349781682361*sf(10, a, 1)*sp.sin(174532925199433*alpha/10000000000000000)/2000000



w = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) + (
U = (-Ah*sf(x, 0, 1) - Ch*sf(x, 10, 1) - uc*sf(x, 10, 0) - Bh*sf(x, 20, 1) + 1*sp.sin(hoe

a_value2 = 14
condition1 = And(10 < a_value2, a_value2 <= 20)

if condition1:
    alpha_value2 = 0
else:
    alpha_value2 = 90


u_test = U.subs(a, a_value2).subs(alpha, alpha_value2)
w_test = w.subs(a, a_value2).subs(alpha, alpha_value2)

x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * (x-1
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * -x
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * -10)
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * w_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * u_te
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * -u_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * w_te
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)



plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)),z_structure_numpy(np.linspace(0,20,101)
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```


originele constructie
verplaatste constructie

```
# voorbeeld 7, 2D van de invloedslijn
x_value = 12

condition1 = And(10 <= x_value, x_value <= 20)

if condition1:
    alpha_value = 0
else:
    alpha_value = 90


w2 = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) +
wV2 = ((1/2)*Am*sf(x, 0, 2) + (1/6)*Av*sf(x, 0, 3) + (1/6)*sp.cos(hoek_rad)*sf(x, a, 3) +
phi2 = sp.diff(w2, x)*-1
M2 = sp.diff(phi2, x)
V2 = sp.diff(wV2, x, 3)


U2 = (-Ah*sf(x, 0, 1) - Ch*sf(x, 10, 1) - uc*sf(x, 10, 0) - Bh*sf(x, 20, 1) + 1*sp.sin(ho
N2 = sp.diff(U2, x)

Am2 = Am/factor
Av2 = Av/factor
Ah2 = Ah/factor

u_testt = U2.subs(factor, 10).subs(x, x_value).subs(alpha, alpha_value)
w_testt = w2.subs(factor, 10).subs(x, x_value).subs(alpha, alpha_value)
phi_testt = phi2.subs(factor, 3).subs(x, x_value).subs(alpha, alpha_value)
M_testt = M2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
V_testt = V2.subs(factor, 1/2).subs(x, x_value).subs(alpha, alpha_value)
N_testt = N2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
Am_testt = Am2.subs(factor, 1/2).subs(x, x_value).subs(alpha, alpha_value)
Av_testt = Av2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)
Ah_testt = Ah2.subs(factor, 1).subs(x, x_value).subs(alpha, alpha_value)


u_test = u_testt.subs(a, x)
w_test = w_testt.subs(a, x)
phi_test = phi_testt.subs(a, x)
M_test = M_testt.subs(a, x)
V_test = V_testt.subs(a, x)
N_test = N_testt.subs(a, x)
Am_test = Am_testt.subs(a, x)
Av_test = Av_testt.subs(a, x)
Ah_test = Ah_testt.subs(a, x)

x_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * (x-1
z_structure = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * -x
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * -10)
x_structure_numpy = sp.lambdify(x,x_structure)
z_structure_numpy = sp.lambdify(x,z_structure)

x_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * V_test
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * 0).e
z_res = ((sp.SingularityFunction(x,0,0) - sp.SingularityFunction(x,10,0)) * 0
               + (sp.SingularityFunction(x,10,0) - sp.SingularityFunction(x,20,0)) * V_te
x_res_numpy = sp.lambdify(x,x_res)
z_res_numpy = sp.lambdify(x,z_res)


plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,20,101)), z_structure_numpy(np.linspace(0,20,101
```

```
plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),z_
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```

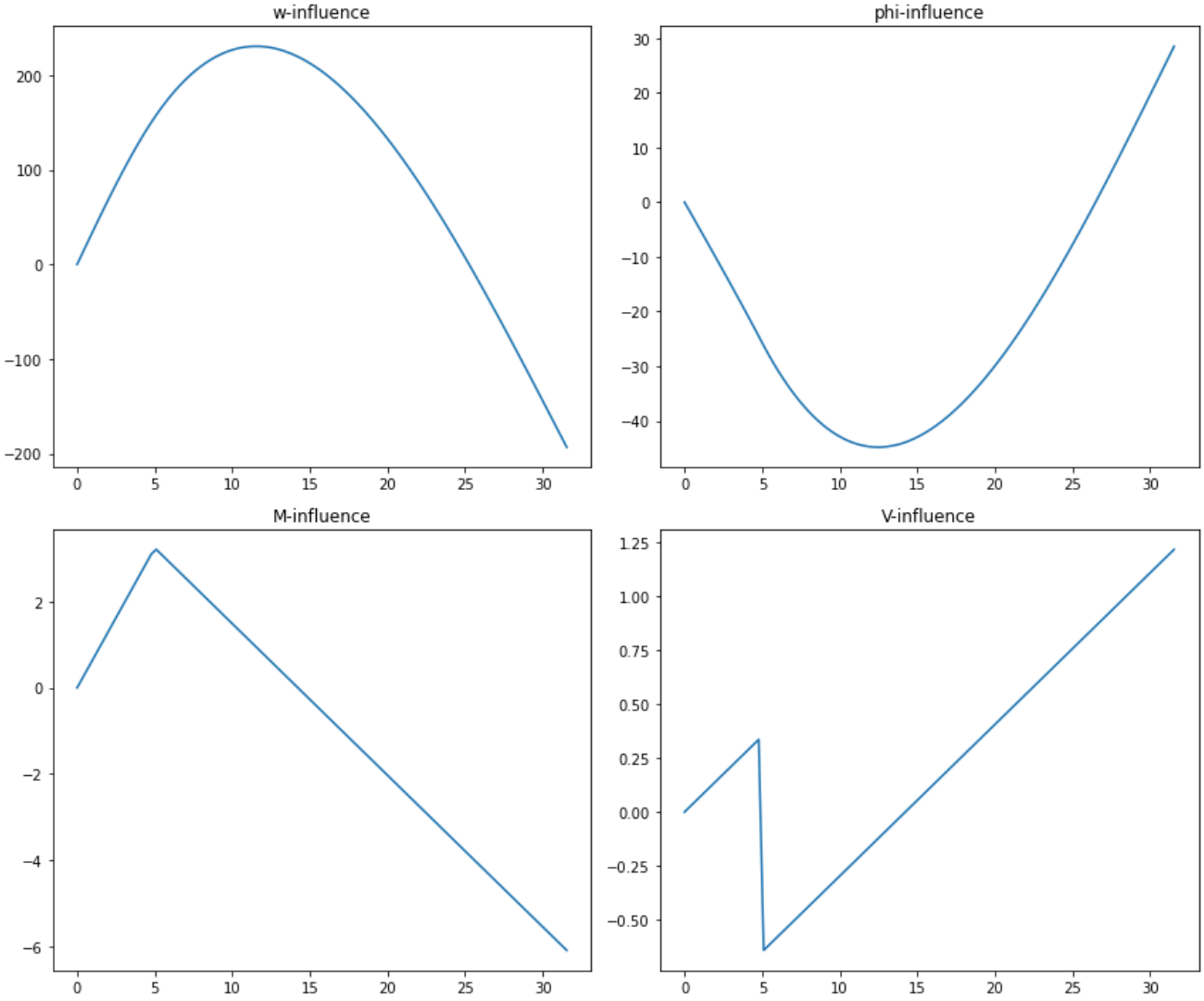



```
# voorbeeld 8, Onbekende uitrekenen

Av, Cv, Dv, Ev, Bv, Ah, Ch, Dh, Eh, uc, ud, ue, wc, wd, we, b, Cphi = sp.symbols('Av, Cv,
b = 5/(np.cos((30*np.pi)/180))

M202b = -Av*(20 + 2*b) - 1*sp.Function('sf')((20 + 2*b), a, 1) - Dv*(10 + b) - 10*Ev - Cv
w202b = (((20 + 2*b)**3)/6)*Av + (1/6)*sp.Function('sf')((20 + 2*b), a, 3) + (((10 + 2*b)
V202b = -Av - 1 - Cv - Dv - Ev - Bv
N202b = -Ah - Ch - Dh - Eh
N10 = -Ah - Ch - np.cos((240*np.pi)/180)*(-Ah) + np.sin((240*np.pi)/180)*(-Av - 1)
V10 = -Av - 1 - Cv - np.sin((240*np.pi)/180)*(-Ah) - np.cos((240*np.pi)/180)*(-Av - 1)
N10b = -Ah - Ch - Dh - np.cos((240*np.pi)/180)*(-Ah - Ch) + np.sin((240*np.pi)/180)*(-Av
V10b = -Av - 1 - Cv - Dv - np.sin((240*np.pi)/180)*(-Ah - Ch) - np.cos((240*np.pi)/180)*(
N102b = -Ah - Ch - Dh - Eh - np.cos((240*np.pi)/180)*(-Ah - Ch - Dh) + np.sin((240*np.pi)
V102b = -Av - 1 - Cv - Dv - Ev - np.sin((240*np.pi)/180)*(-Ah - Ch - Dh) - np.cos((240*np

u10 = -10*Ah - uc - np.cos((240*np.pi)/180)*(-10*Ah) + np.sin((240*np.pi)/180)*(((10**3)/
w10 = ((10**3)/6)*Av + (1/6)*sp.Function('sf')(10, a, 3) -10*Cphi + wc - np.sin((240*np.p
u10b = -(10 + b)*Ah - Ch*b - uc - ud - np.cos((240*np.pi)/180)*(-(10 + b)*Ah - Ch*b - uc)
w10b = (((10+b)**3)/6)*Av + (1/6)*sp.Function('sf')((10 + b), a, 3) + ((b**3)/6)*Cv -(10
u102b = -(10 + 2*b)*Ah - Ch*2*b - uc - Dh*b - ud - ue - np.cos((240*np.pi)/180)*(-(10 + 2
w102b = (((10 + 2*b)**3)/6)*Av + (1/6)*sp.Function('sf')((10 + 2*b), a, 3) + (((2*b)**3)/

oplossing = sp.solve((M202b, w202b, V202b, N202b, N10, V10, N10b, V10b, N102b, V102b, u10
oplossing_in_breuken = {variabele: sp.nsimplify(waarde) for variabele, waarde in oplossin
oplossing_df = pd.DataFrame([(str(variabele), waarde) for variabele, waarde in oplossing_

pd.set_option('display.latex.repr', True)
pd.set_option('display.max_colwidth', None)

display(oplossing_df)
```

	Variabele	Waarde
0	Av	121748225867393/100000000000000 - 702913709778989*sf(12618802153517/400000000000, a, 1)/10000000000000000
1	Cv	3294908014589*sf(12618802153517/400000000000, a, 1)/31250000000000 - 33262233880109/10000000000000
2	Dv	196855839628573*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000000 - 3881382645371/3125000000000000000000000000000
3	Ev	332622338801089/100000000000000 - 3294908014589*sf(12618802153517/400000000000, a, 1)/31250000000000
4	Bv	175728427444747*sf(12618802153517/400000000000, a, 1)/2500000000000000 - 221748225867393/100000000000000
5	Ah	340964316003319*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000000 - 215128062252757/1000000000000000000000000000000
6	Ch	48009899211323/25000000000000 - 608741129336967*sf(12618802153517/400000000000, a, 1)/10000000000000000
7	Dh	121748225867393*sf(12618802153517/400000000000, a, 1)/1000000000000000 - 48009899211323/12500000000000
8	Eh	48009899211323/25000000000000 - 304370564668483*sf(12618802153517/400000000000, a, 1)/5000000000000000
9	uc	-72168783648703*sf(10, a, 3)/500000000000000 - 236954847200147*sf(12618802153517/400000000000, a, 1)/5000000000000 + 457531754730549*sf(12618802153517/400000000000, a, 3)/10000000000000000 + 202913709778991/1000000000000
10	ud	433012701892219*sf(10, a, 3)/1000000000000000 - 72168783648703*sf(157735026918963/10000000000000, a, 3)/500000000000000 + 27760181294669*sf(12618802153517/400000000000, a, 1)/312500000000 - 162726856926153*sf(12618802153517/400000000000, a, 3)/2500000000000000 - 19542882696157/31250000000
11	ue	-433012701892219*sf(10, a, 3)/1000000000000000 + 433012701892219*sf(157735026918963/10000000000000, a, 3)/1000000000000000 - 72168783648703*sf(8618802153517/400000000000, a, 3)/500000000000000 - 1054994295827*sf(12618802153517/400000000000, a, 1)/39062500000 + 12085979560879*sf(12618802153517/400000000000, a, 3)/625000000000000 + 211229268249019/500000000000

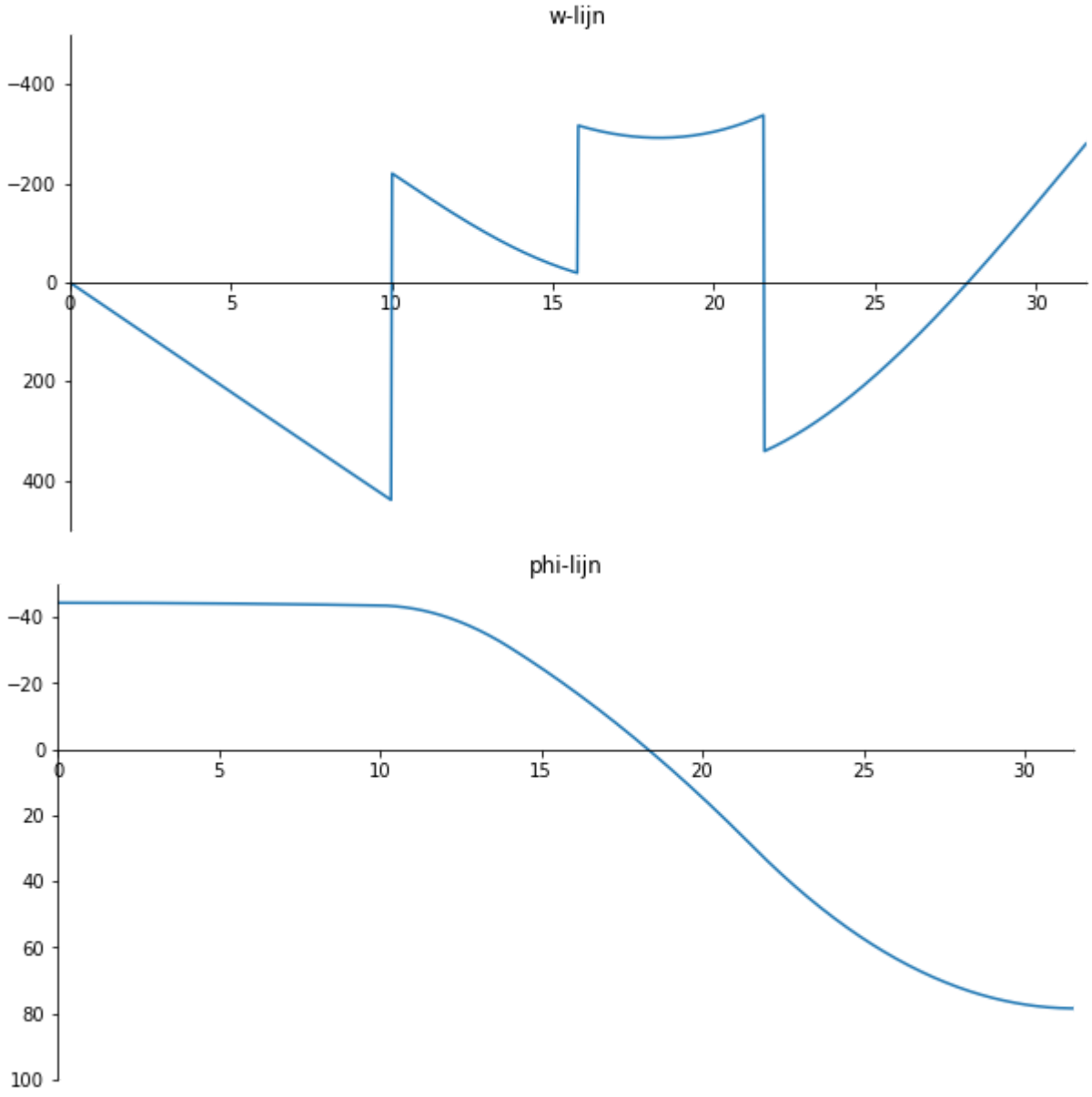
	Variabele	Waarde
12	wc	-sf(10, a, 3)/4 - 205208917225187*sf(12618802153517/400000000000, a, 1)/2500000000000 + 198117061317363*sf(12618802153517/400000000000, a, 3)/2500000000000000 + 351456854889497/1000000000000
13	wd	sf(10, a, 3)/4 - sf(157735026918963/10000000000000, a, 3)/4 - 57610369319079*sf(12618802153517/400000000000, a, 1)/5000000000000 + 457531754730549*sf(12618802153517/400000000000, a, 3)/10000000000000000 - 85463718761807/250000000000
14	we	sf(10, a, 3)/4 + sf(157735026918963/10000000000000, a, 3)/4 - sf(8618802153517/400000000000, a, 3)/4 + 60486859553083*sf(12618802153517/400000000000, a, 1)/625000000000 - 167468245269451*sf(12618802153517/400000000000, a, 3)/5000000000000000 - 365859824652891/500000000000
15	Cphi	-66437606423033*sf(12618802153517/400000000000, a, 1)/10000000000000 + 66039020439121*sf(12618802153517/400000000000, a, 3)/12500000000000000 + 10930456992633/250000000000

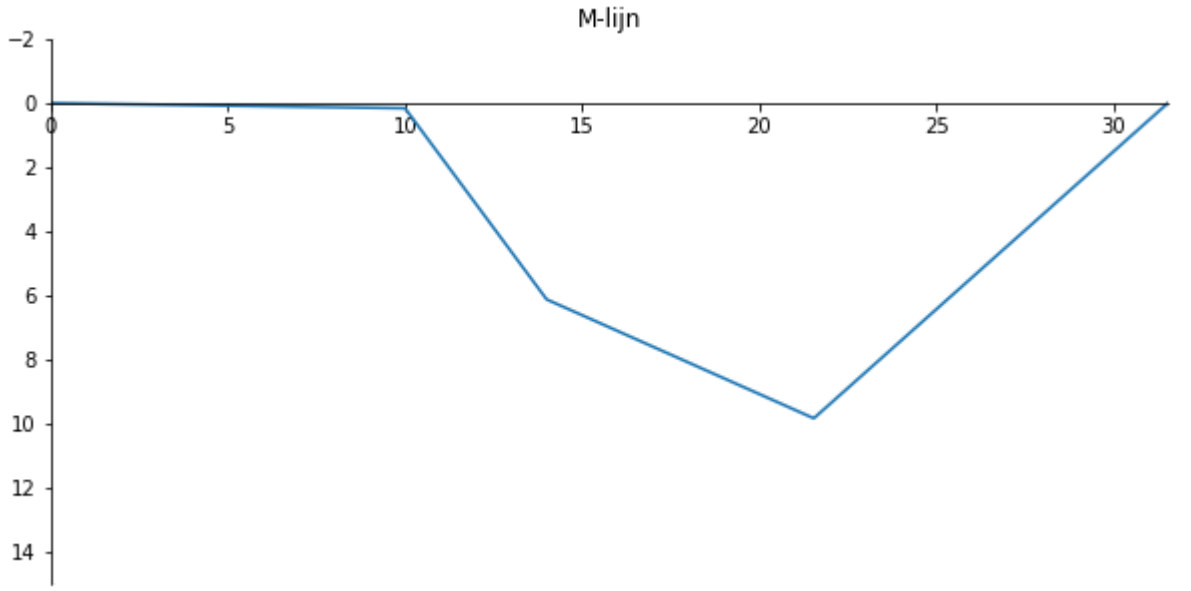
```
# voorbeeld 8, plotten van de invloedslijnen
Av = 121748225867393/100000000000000 - 702913709778989*sf(12618802153517/400000000000, a,
Cv = 3294908014589*sf(12618802153517/400000000000, a, 1)/31250000000000 - 33262233880109/
Dv = 196855839628573*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000000
Ev = 332622338801089/100000000000000 - 3294908014589*sf(12618802153517/400000000000, a, 1
Bv = 175728427444747*sf(12618802153517/400000000000, a, 1)/2500000000000000 - 22174822586
Ah = 340964316003319*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000000
Ch = 48009899211323/25000000000000 - 608741129336967*sf(12618802153517/400000000000, a, 1
Dh = 121748225867393*sf(12618802153517/400000000000, a, 1)/1000000000000000 - 48009899211
Eh = 48009899211323/25000000000000 - 304370564668483*sf(12618802153517/400000000000, a, 1
uc = -72168783648703*sf(10, a, 3)/500000000000000 - 236954847200147*sf(12618802153517/400
ud = 433012701892219*sf(10, a, 3)/1000000000000000 - 72168783648703*sf(157735026918963/10
ue = -433012701892219*sf(10, a, 3)/1000000000000000 + 433012701892219*sf(157735026918963/
wc = -sf(10, a, 3)/4 - 205208917225187*sf(12618802153517/400000000000, a, 1)/250000000000
wd = sf(10, a, 3)/4 - sf(157735026918963/10000000000000, a, 3)/4 - 57610369319079*sf(1261
we = sf(10, a, 3)/4 + sf(157735026918963/10000000000000, a, 3)/4 - sf(8618802153517/40000
Cphi = -66437606423033*sf(12618802153517/400000000000, a, 1)/10000000000000 + 66039020439


w = (1/6)*Av*sf(s, 0, 3) + (1/6)*sf(s, a, 3) + (1/6)*Cv*sf(s, 10, 3) + wc*sf(s, 10, 0) +
wV = (1/6)*Av*sf(s, 0, 3) + (1/6)*sf(s, a, 3) + (1/6)*Cv*sf(s, 10, 3) + (1/6)*Dv*sf(s, (1
phi = sp.diff(w, s)*-1
M = sp.diff(phi, s)
V = sp.diff(wV, s, 3)

U = -Ah*sf(s, 0, 1) - Ch*sf(s, 10, 1) - uc*sf(s, 10, 0) - Dh*sf(s, (10 + b), 1) - ud*sf(s
N = sp.diff(U, s)

s_value = 5
a_values = np.linspace(0, (20 + 2*b), 100)

w_values = [w.subs([(s, s_value), (a, a_value)]) for a_value in a_values]
phi_values = [phi.subs([(s, s_value), (a, a_value)]) for a_value in a_values]
M_values = [M.subs([(s, s_value), (a, a_value)]) for a_value in a_values]
V_values = [V.subs([(s, s_value), (a, a_value)]) for a_value in a_values]

plt.figure(figsize=(12, 10))

plt.subplot(221)
plt.plot(a_values, w_values)
plt.title('w-influence')
plt.subplot(222)
plt.plot(a_values, phi_values)
plt.title('phi-influence')
plt.subplot(223)
plt.plot(a_values, M_values)
plt.title('M-influence')
plt.subplot(224)
plt.plot(a_values, V_values)
plt.title('V-influence')

plt.tight_layout()
plt.show()
```

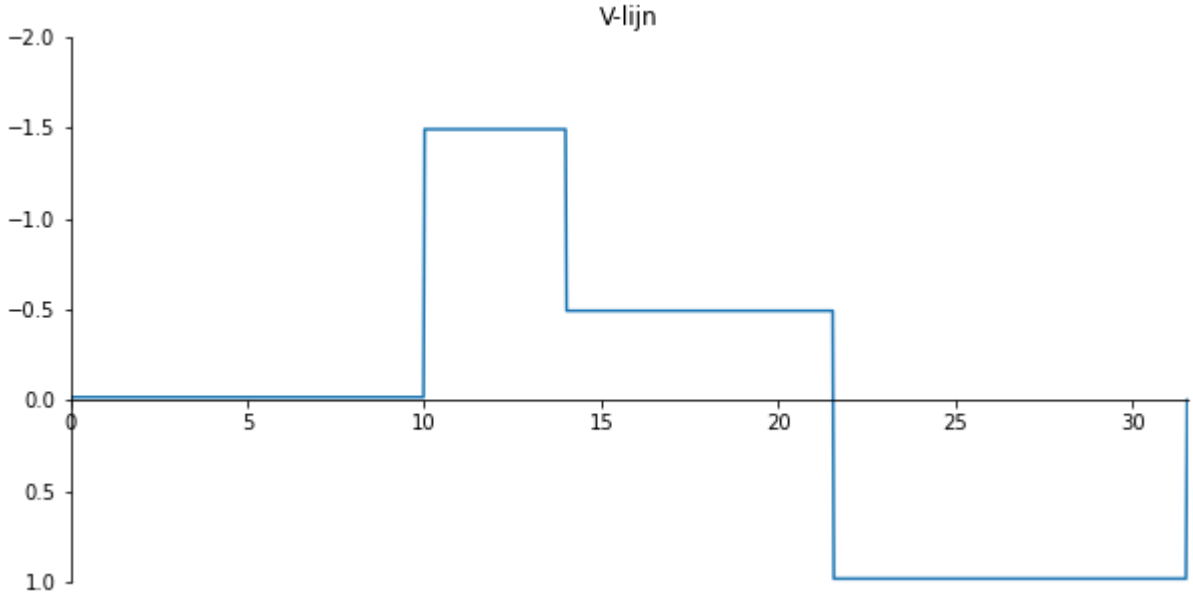

w-influence
phi-influence
M-influence
V-influence

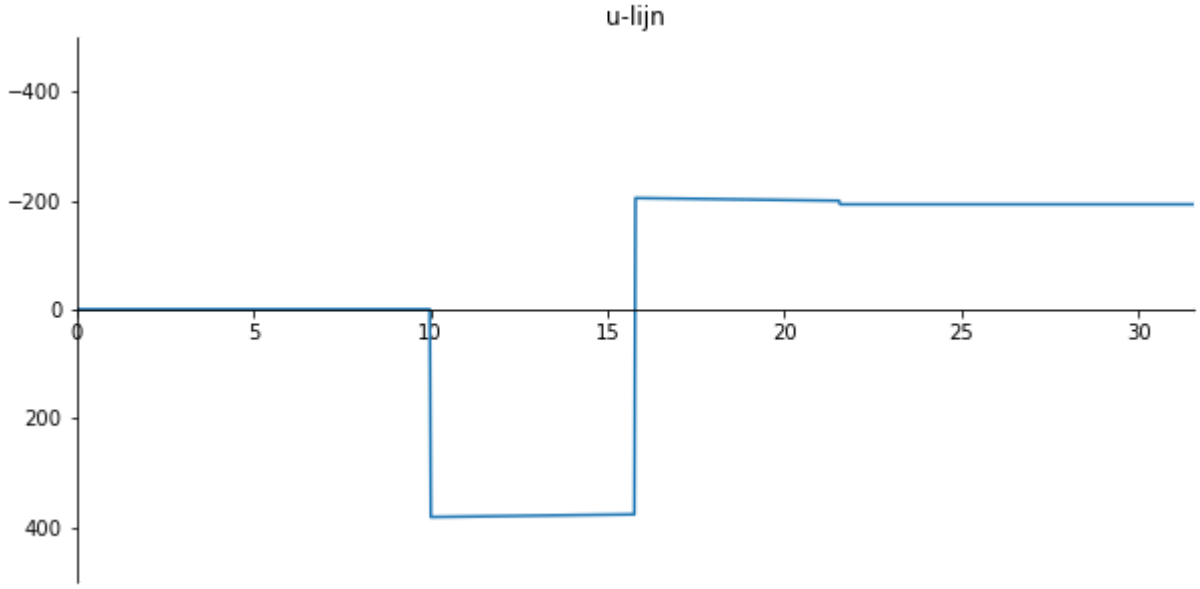
```
# Voorbeeld 8, interne krachtslijnen plotten
a_value = 14

w_numpy = sp.lambdify(s, w.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')
phi_numpy = sp.lambdify(s, phi.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')
M_numpy = sp.lambdify(s, M.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')
V_numpy = sp.lambdify(s, V.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')
u_numpy = sp.lambdify(s, U.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')
N_numpy = sp.lambdify(s, N.subs(a, a_value).rewrite(sp.Piecewise), modules='numpy')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), w_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([500, -500])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('w-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), phi_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([100, -50])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('phi-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), M_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([15, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('M-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), V_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([1, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('V-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), u_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([500, -500])
ax.spines['left'].set_position('zero')
```

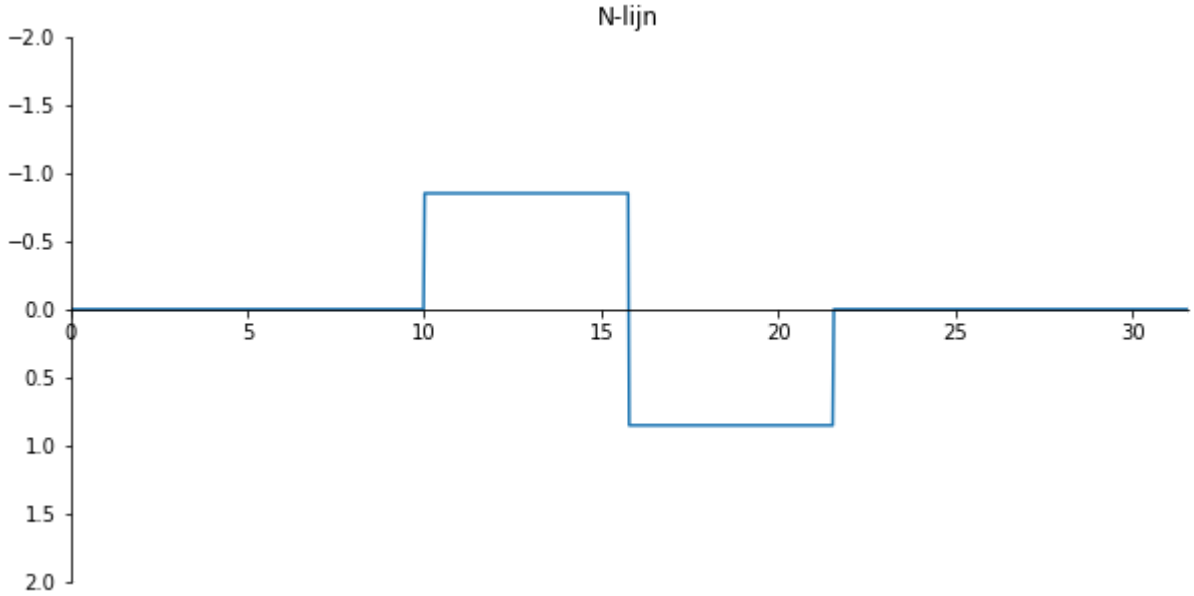
```
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('u-lijn')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(np.linspace(0, (20+2*b), 1000), N_numpy(np.linspace(0, (20+2*b), 1000)))
plt.xlim([0, (20+2*b)])
plt.ylim([2, -2])
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.yaxis.tick_left()
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
ax.xaxis.tick_bottom()
plt.title('N-lijn')

plt.show()
```




M-lijn
−2
0
2
4
6
8
10
12
14
0
5
10
15
20
25
30

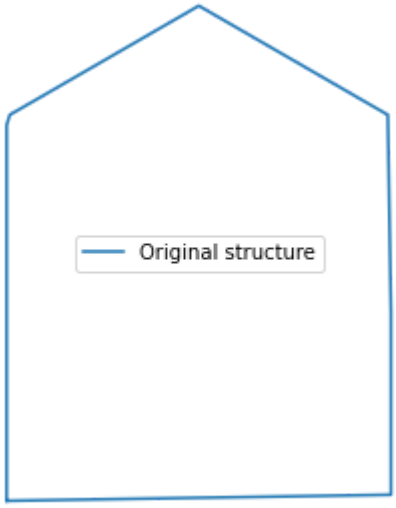

V-lijn
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30


u-lijn
−400
−200
0
200
400
0
5
10
15
20
25
30


N-lijn
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
5
10
15
20
25
30

```
#Voorbeeld 8, plotten 2D van de doorbuiging

Av = 121748225867393/100000000000000 - 702913709778989*sf(12618802153517/400000000000, a,
Cv = 3294908014589*sf(12618802153517/400000000000, a, 1)/31250000000000 - 33262233880109/
Dv = 196855839628573*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000
Ev = 332622338801089/100000000000000 - 3294908014589*sf(12618802153517/400000000000, a, 1
Bv = 175728427444747*sf(12618802153517/400000000000, a, 1)/2500000000000000 - 22174822586
Ah = 340964316003319*sf(12618802153517/400000000000, a, 1)/5000000000000000000000000000
Ch = 48009899211323/25000000000000 - 608741129336967*sf(12618802153517/400000000000, a, 1
Dh = 121748225867393*sf(12618802153517/400000000000, a, 1)/1000000000000000 - 48009899211
Eh = 48009899211323/25000000000000 - 304370564668483*sf(12618802153517/400000000000, a, 1
uc = -72168783648703*sf(10, a, 3)/500000000000000 - 236954847200147*sf(12618802153517/400
ud = 433012701892219*sf(10, a, 3)/1000000000000000 - 72168783648703*sf(157735026918963/10
ue = -433012701892219*sf(10, a, 3)/1000000000000000 + 433012701892219*sf(157735026918963/
wc = -sf(10, a, 3)/4 - 205208917225187*sf(12618802153517/400000000000, a, 1)/250000000000
wd = sf(10, a, 3)/4 - sf(157735026918963/10000000000000, a, 3)/4 - 57610369319079*sf(1261
we = sf(10, a, 3)/4 + sf(157735026918963/10000000000000, a, 3)/4 - sf(8618802153517/40000
Cphi = -66437606423033*sf(12618802153517/400000000000, a, 1)/10000000000000 + 66039020439


w = ((1/6)*Av*sf(x, 0, 3) + (1/6)*sf(x, a, 3) + (1/6)*Cv*sf(x, 10, 3) + wc*sf(x, 10, 0) +
U = (-Ah*sf(x, 0, 1) - Ch*sf(x, 10, 1) - uc*sf(x, 10, 0) - Dh*sf(x, (10 + b), 1) - ud*sf(


a_value = 5

u_test = U.subs(a, a_value).subs(factor, 500)
w_test = w.subs(a, a_value).subs(factor, 500)


x_structure = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * 0
               + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, (10+b), 0)
               + (sp.SingularityFunction(x, (10+b), 0) - sp.SingularityFunction(x, (10+2*
               + (sp.SingularityFunction(x, (10+2*b), 0) - sp.SingularityFunction(x, (20+

z_structure = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * -x
               + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, (10+b), 0)
               + (sp.SingularityFunction(x, (10+b), 0) - sp.SingularityFunction(x, (10+2*
               + (sp.SingularityFunction(x, (10+2*b), 0) - sp.SingularityFunction(x, (20+


x_res = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * w_test
               + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, (10+b), 0)
               + (sp.SingularityFunction(x, (10+b), 0) - sp.SingularityFunction(x, (10+2*
               + (sp.SingularityFunction(x, (10+2*b), 0) - sp.SingularityFunction(x, (20+

z_res = ((sp.SingularityFunction(x, 0, 0) - sp.SingularityFunction(x, 10, 0)) * -u_test
               + (sp.SingularityFunction(x, 10, 0) - sp.SingularityFunction(x, (10+b), 0)
               + (sp.SingularityFunction(x, (10+b), 0) - sp.SingularityFunction(x, (10+2*
               + (sp.SingularityFunction(x, (10+2*b), 0) - sp.SingularityFunction(x, (20+

b_value = (5/np.cos((30*np.pi)/180))


x_structure_numpy = sp.lambdify(x, x_structure.subs(b, b_value), 'numpy')
z_structure_numpy = sp.lambdify(x, z_structure.subs(b, b_value), 'numpy')

x_res_numpy = sp.lambdify(x, x_res.subs(b, b_value))
z_res_numpy = sp.lambdify(x, z_res.subs(b, b_value))


plt.figure(figsize=(10, 5))
plt.plot(x_structure_numpy(np.linspace(0,(20+2*b_value),101)),z_structure_numpy(np.linspa
```

```
# plt.plot(x_res_numpy(np.linspace(0,20,2010))+x_structure_numpy(np.linspace(0,20,2010)),
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.axis('off')
plt.legend()
plt.show()
```



Ezzat: Curved beams

Contents

De methode van Macaulay is een methode die de mogelijkheid biedt om zowel de interne krachten als de vervormingen van construties te analyseren. De discontiue belastingen die op de constructie werken zijn in één vergelijking te beschrijven door gebruik van de singulariteitsfunctie te maken. Hierdoor kan de volledige constructiee op een gestructureerde manier worden gemodelleerd met slechts één differenti aalvergelijking.

In het werkelijkheid kunnen liggers schuin geplaatst zijn of een gebogen vorm hebben. Echter, is er nog geen onderzoek gedaan naar de toepassing van Macaulay's methode met niet rechte liggers. Dit rapport probeert de volgende vraag en de te beantwoorden:

Hoe kan de methode van Macaulay worden toegepast met betrekking tot zowel schuine liggers als kromme liggers?

Om de schuine liggers te kunnen analyseren, worden de evenwichtsvergelijkingen eerst opgesteld voor een hellingssnede (snede van een balk die onder hoek θ staat met de horizontale as).

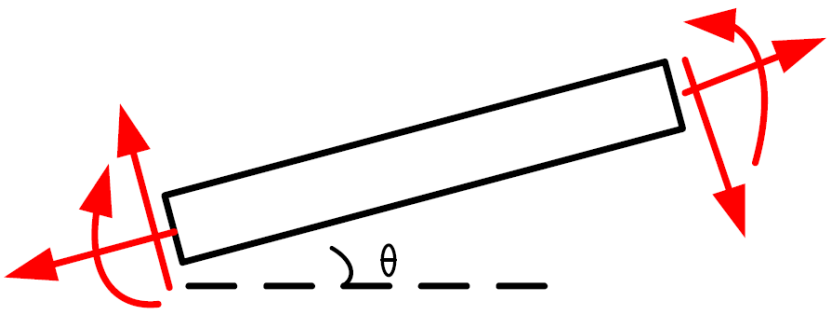


Fig. 26 Hellingsnede

Daarmee worden de volgende differentiaalvergelijkingen verkregen die later worden toegepast op verschillende constructies.

$$\frac{dV}{dx} = -\sin(\theta) \cdot q_x - \cos(\theta) \cdot q_z$$
$$\frac{dN}{dx} = -\cos(\theta) \cdot q_x + \sin(\theta) \cdot q_x$$
$$\frac{dM}{dx} = \frac{V}{\cos(\theta)}$$

Er blijkt dat deze differentiaalvergelijkingen de juiste waarden geven voor zowel de interne krachten als de vervormingen van de constructies.

Op dezelfde manier worden differentiaalvergelijkingen afgeleid voor een kniksnede (snede van een geknikte constructie waar de eerste staaf onder een hoek θ staat met de horizontale as en de tweede staaf onder een hoek $\Delta\theta$ staat ten opzicht van de eerste staaf).

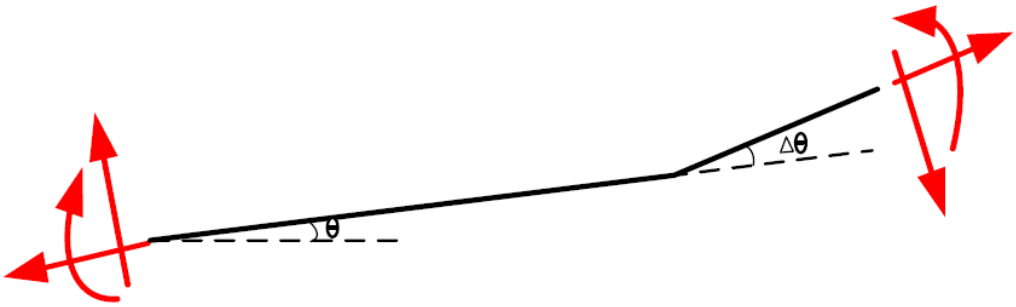


Fig. 27 Kniksnede

Onder de aanname dat een kromme lijn is een opeenvolging van lijnsegmenten die elkaar opvlogen met kleine hoekverschillen, kan $\Delta\theta$ als klein worden beschouwd. Door deze aanname, in combinatie van de drie evenwichtsvergelijkingen voor de kinksnede, worden opnieuw dezelfde differentiaalvergelijkingen verkregen die al eerder zijn afgeleid voor de hellingssnede. Het enige verschil nu is dat de hoek θ niet langer constant maar varieert telkens wanneer de kromme van richting verandert. De hoek θ kan zowel globaal (globale θ is de afgeleide van de constructoefunctoe) als lokaal (lokale θ is de hoek onder elke lijnsegment van de kromme) worden bepaald. Het blijkt dat het toepassen van de lokale θ een nauwkeurige analyse oplevert van zowel de krachten als de doorbuiging.

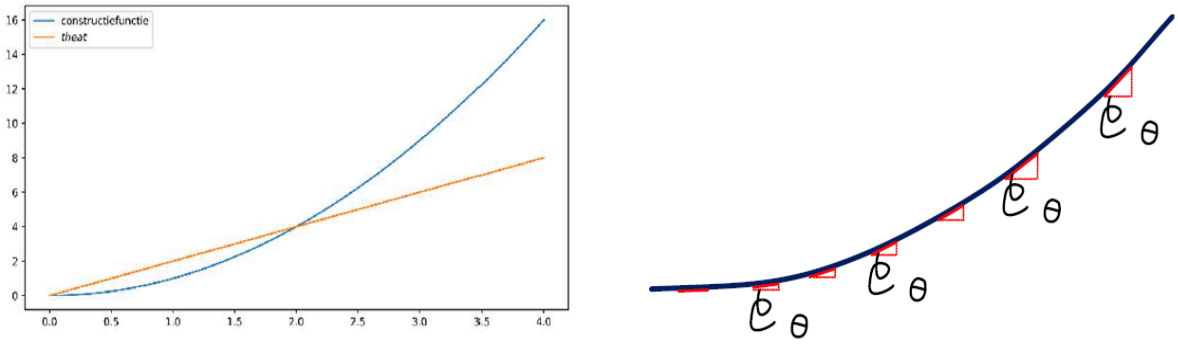


Fig. 28 Globale en lokale θ

Dit onderzoek illustreert de veelzijdigheid van de Macaulay-methode en presenteert nieuwe inzichten in het omgaan met schuine en kromme liggers, inclusief een innovatieve aanpak voor geknikte constructies.

Qadriyeh (2024)

Documenten

- TU Delft Education repository
- GitHub repository, examples also shown in this book:
  - Code example kinked structure
  - Code example variants integrating
  - Code example 1 to 3
  - Code example 4 and 5

Geknikte constructie

Contents


```
import sympy as sp
import numpy as np
from sympy import symbols
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

```
EI = symbols('EI')
x = symbols('x')
```

Benadering

```
x, Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('x, C_v, C_m, C_phi, C_w, A_v, B_v, M

l = 4
F1 = 10
F2 = 10

theta = sp.atan(0.25)
alpha = sp.atan(0.5)
```

```
## Krachtenvergelijking van staaf 1
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1))
qx1 = sp.nsimplify(Ah * sf(x, 0, -1))
```

```
```

```
## Krachtenvergelijking van staaf 2

qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1))
qx2 = 0
```

```
## V = V1 + V2
## Dwarskracht in de eerste staaf
V1 = sp.integrate(-qx1 * sp.sin(theta), x) + sp.integrate(-qz1 * sp.cos(theta), x)
## Dwasrkracht in de tweede staaf
V2 = sp.integrate(-qx2 * sp.sin(alpha), x) + sp.integrate(-qz2 * sp.cos(alpha), x)

V = V1 + V2 + Cv
```

```
## N = N1 + N2
## Normaalkracht in de eerste staaf
N1 = sp.integrate(-qx1 * sp.cos(theta), x) + sp.integrate(qz1 * sp.sin(theta), x)
## Normaalkracht in de tweedee staaf
N2 = sp.integrate(-qx2 * sp.cos(alpha), x) + sp.integrate(qz2 * sp.sin(alpha), x)
N = N1 + N2 + Cn
```

```
M1 = sp.integrate(V1/sp.cos(theta), x)
M2 = sp.integrate(V2/sp.cos(alpha), x)
M = M1 + M2 + Cm
phi = sp.integrate(M, x) + Cphi
W = sp.integrate(phi, x) + Cw
```

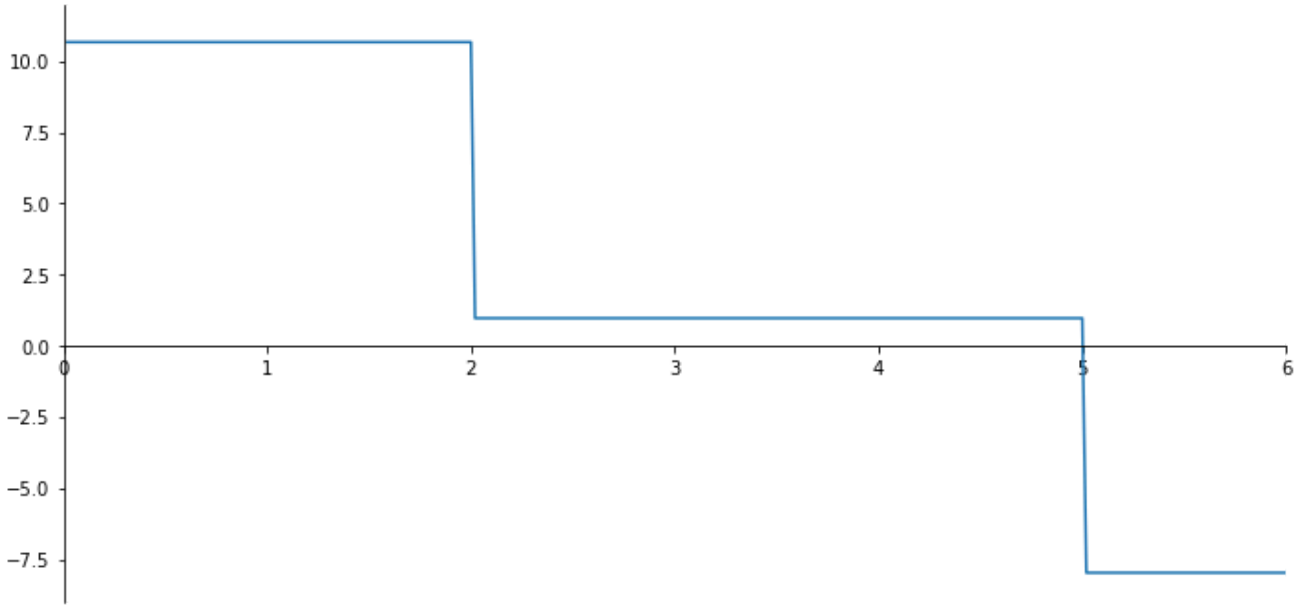
```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)

eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)

eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l )

eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) )

equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cn, Cm, Cphi, Cw, Av, Bv, MA, Cv, Ah))
```

```
display(solutions)
```

```
{C_n: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 11.0539215686274,
 B_v: 8.92036924928217,
 M_A: 15.9722222222222,
 C_v: 0.0,
 A_h: 0.234204793028313}
```

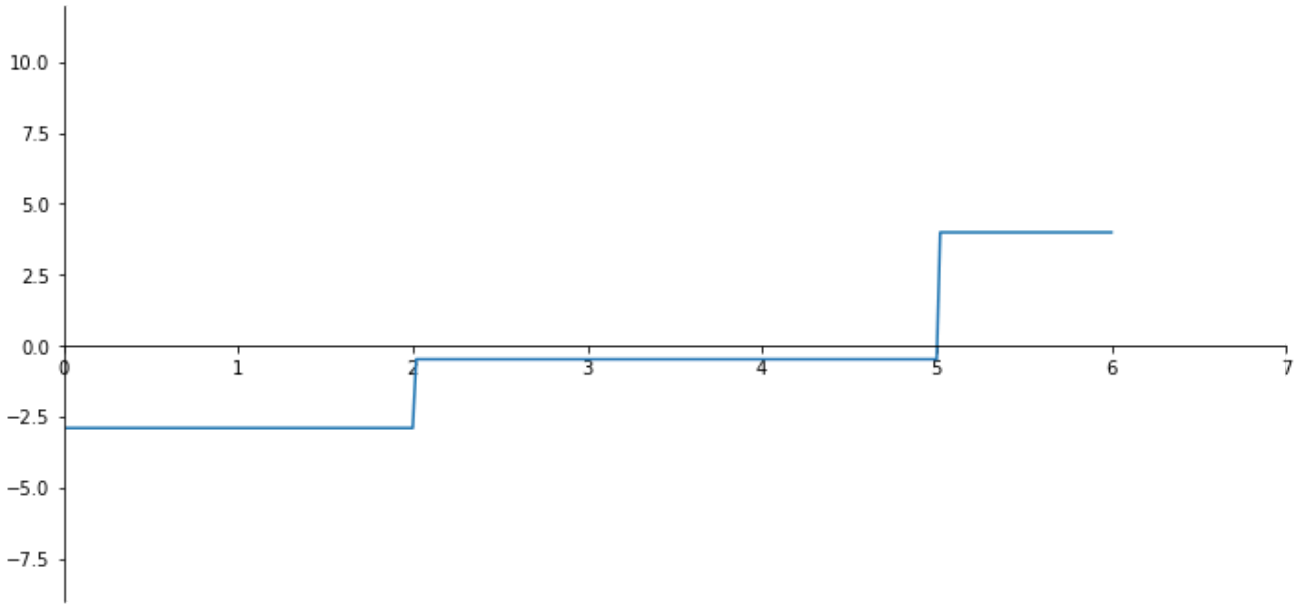
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)

print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list));
```

```
[0.9656511 0.9656511]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 7)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list));
```

```
[-0.48282555 -0.48282555]
```



Iteratie 1

```
```

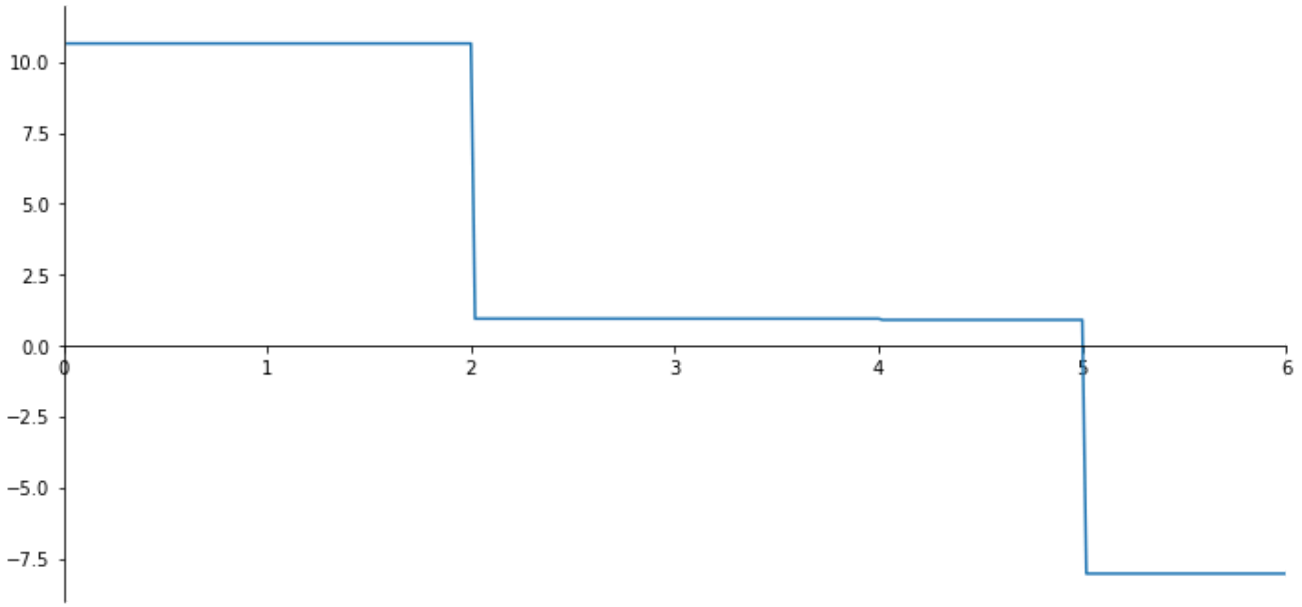
```
Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('C_v, C_m, C_phi, C_w, A_v, B_v, M_A, A_

# Define qz and qx
DwaL =  0.9656511
DwaR =  0.9656511
NorL = -0.48282555
NorR = -0.48282555
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1) + DwaL * sf
qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1) - DwaR *sf(x, l, -1)
qx1 = sp.nsimplify(Ah * sf(x, 0, -1) + DwaL * sf(x, l, -1) * sp.sin(theta)  + NorL * sf(x
qx2 = sp.nsimplify(-DwaR * sf(x, l, -1)*sp.sin(alpha) -NorR * sf(x, l, -1)* sp.cos(alpha)


# Define V as a function of x
## staaf 1
V1 = sp.cos(theta) * sp.integrate( -qz1 , x)
V2 = sp.integrate(- (sp.sin(theta) * sp.tan(theta) * qx1), x)
## staaf 2
V3 = sp.cos(alpha) * sp.integrate( -qz2 , x)
V4 = sp.integrate(- (sp.sin(alpha) * sp.tan(alpha) * qx2), x)

V =V1 + V2 + V3 + V4 + Cv

# Define M as an integral of V
## staaf 1
M1 = sp.integrate( (V1  / sp.cos(theta)), x)
M2 = sp.integrate( (V2  / sp.cos(theta)), x)
## sfaaf 2
M3 = sp.integrate((V3 / sp.cos(alpha)), x)
M4 = sp.integrate((V4 / sp.cos(alpha)), x)
M  = M1 + M2 + M3+ M4 + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw

N1 = sp.integrate(sp.sin(theta) * (-qx1 + qz1), x) + Cn
N2 = sp.integrate(sp.sin(alpha) * (-qx2 + qz2), x)
N = N1+ N2 +Cn
```

```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)
eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)
eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l)
eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) +1 )
equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, MA, Cn, Ah))
display(solutions)
```

```
{C_v: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 10.9868559793420,
 B_v: 8.98907082505678,
 M_A: 15.9363582385174,
 C_n: 0.0,
 A_h: 0.174590935654146}
```

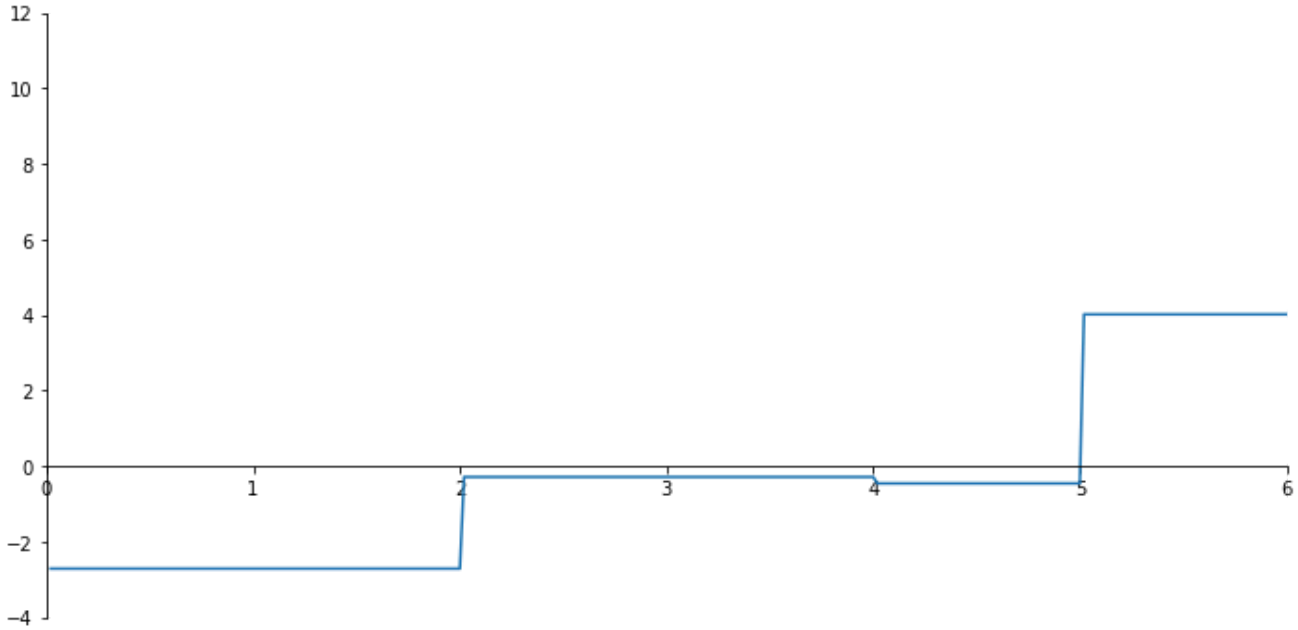
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)
print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list));
```

```
[0.9468048  0.90420254]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-4, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list));
```

```
[-0.28169225 -0.45210127]
```



Iteratie 2

```
```

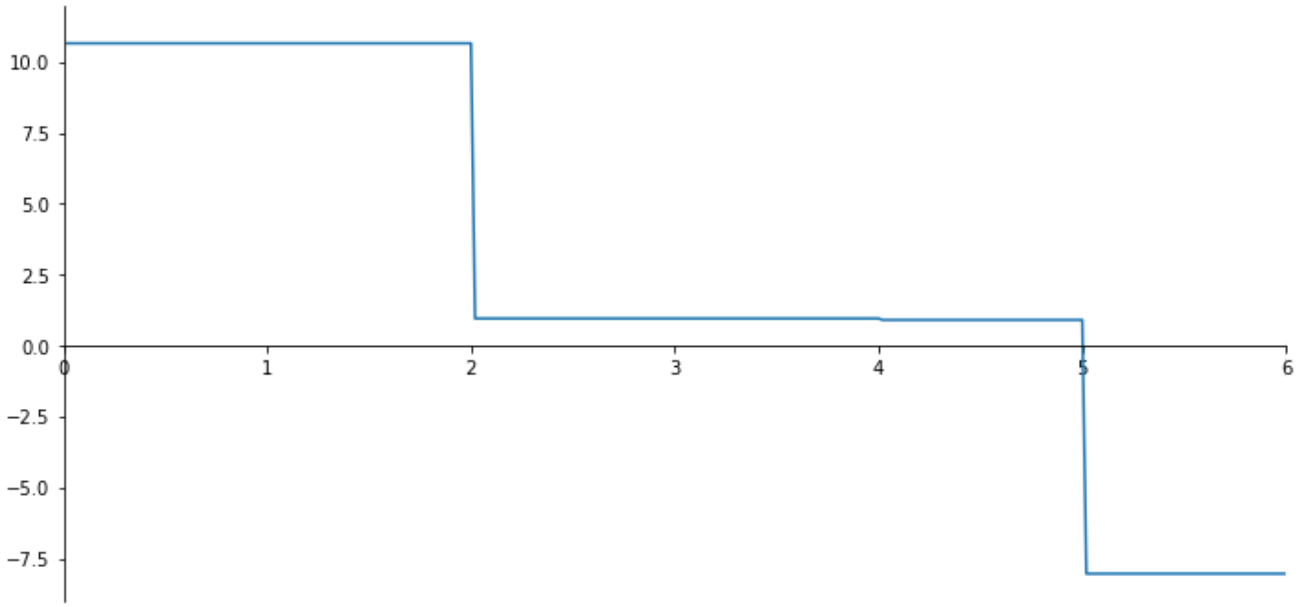
```
Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('C_v, C_m, C_phi, C_w, A_v, B_v, M_A, A_

# Define qz and qx
DwaL =  0.9468048
DwaR =  0.90420254
NorL = -0.28169225
NorR = -0.45210127
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1) + DwaL * sf
qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1) - DwaR *sf(x, l, -1)
qx1 = sp.nsimplify(Ah * sf(x, 0, -1) + DwaL * sf(x, l, -1) * sp.sin(theta)  + NorL * sf(x
qx2 = sp.nsimplify(-DwaR * sf(x, l, -1)*sp.sin(alpha) -NorR * sf(x, l, -1)* sp.cos(alpha)


# Define V as a function of x
## staaf 1
V1 = sp.cos(theta) * sp.integrate( -qz1 , x)
V2 = sp.integrate(- (sp.sin(theta) * sp.tan(theta) * qx1), x)
## staaf 2
V3 = sp.cos(alpha) * sp.integrate( -qz2 , x)
V4 = sp.integrate(- (sp.sin(alpha) * sp.tan(alpha) * qx2), x)

V =V1 + V2 + V3 + V4 + Cv

# Define M as an integral of V
## staaf 1
M1 = sp.integrate( (V1  / sp.cos(theta)), x)
M2 = sp.integrate( (V2  / sp.cos(theta)), x)
## sfaaf 2
M3 = sp.integrate((V3 / sp.cos(alpha)), x)
M4 = sp.integrate((V4 / sp.cos(alpha)), x)
M  = M1 + M2 + M3+ M4 + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw

N1 = sp.integrate(sp.sin(theta) * (-qx1 + qz1), x) + Cn
N2 = sp.integrate(sp.sin(alpha) * (-qx2 + qz2), x)
N = N1+ N2 + Cn
```

```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)
eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)
eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l)
eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) +1 )
equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, MA, Cn, Ah))
display(solutions)
```

```
{C_v: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 10.9850963259576,
 B_v: 8.99087340809321,
 M_A: 15.9483989566231,
 C_n: 0.0,
 A_h: 0.0420835912550643}
```

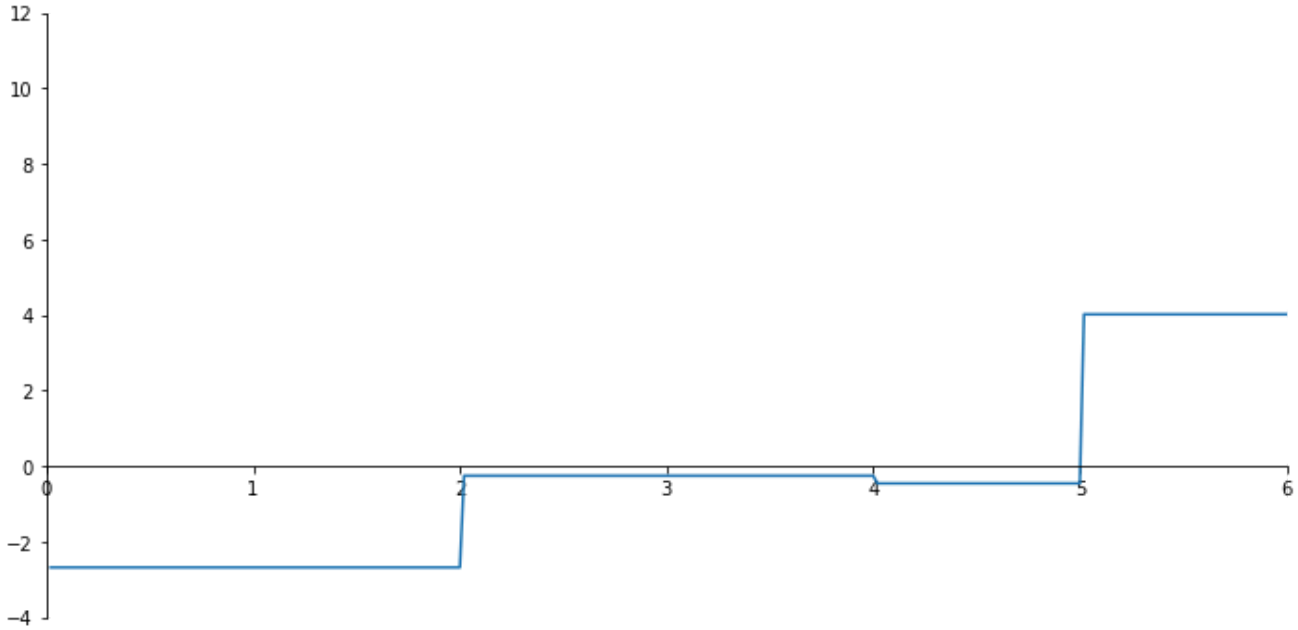
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)
print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list));
```

```
[0.95313212 0.90259026]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-4, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list));
```

```
[-0.24912772 -0.45129513]
```



Iteratie 3

```
```

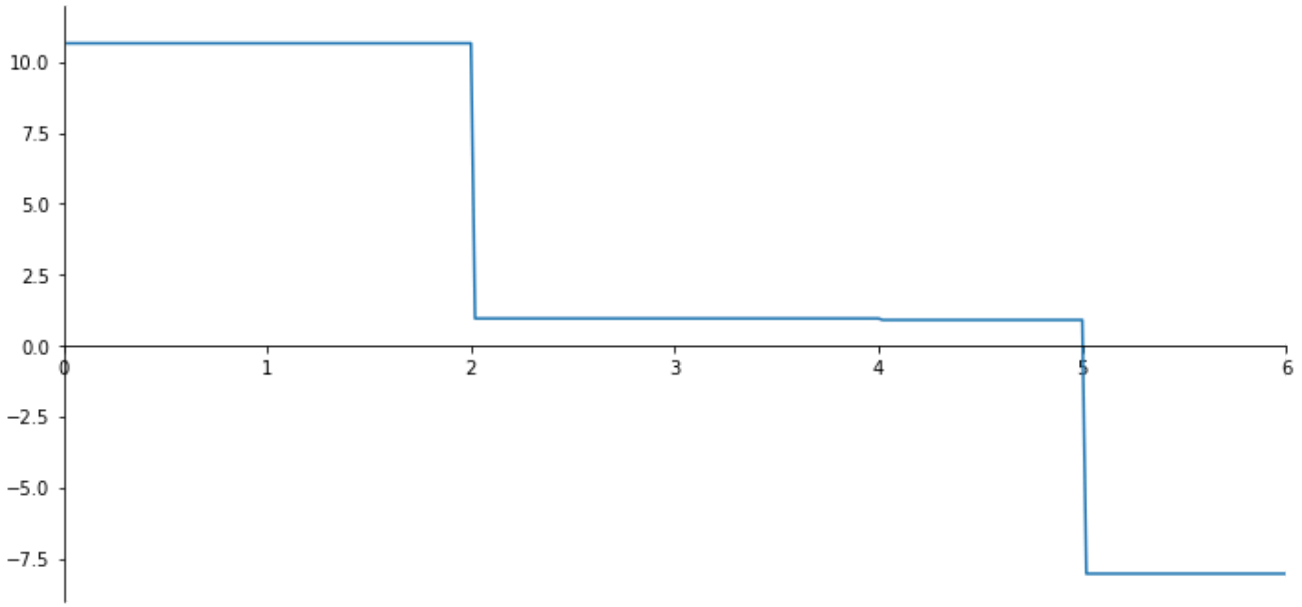
```
Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('C_v, C_m, C_phi, C_w, A_v, B_v, M_A, A_

# Define qz and qx
DwaL =  0.95313212
DwaR =  0.90259026
NorL = -0.24912772
NorR = -0.45129513
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1) + DwaL * sf
qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1) - DwaR *sf(x, l, -1)
qx1 = sp.nsimplify(Ah * sf(x, 0, -1) + DwaL * sf(x, l, -1) * sp.sin(theta)  + NorL * sf(x
qx2 = sp.nsimplify(-DwaR * sf(x, l, -1)*sp.sin(alpha) -NorR * sf(x, l, -1)* sp.cos(alpha)


# Define V as a function of x
## staaf 1
V1 = sp.cos(theta) * sp.integrate( -qz1 , x)
V2 = sp.integrate(- (sp.sin(theta) * sp.tan(theta) * qx1), x)
## staaf 2
V3 = sp.cos(alpha) * sp.integrate( -qz2 , x)
V4 = sp.integrate(- (sp.sin(alpha) * sp.tan(alpha) * qx2), x)

V =V1 + V2 + V3 + V4 + Cv

# Define M as an integral of V
## staaf 1
M1 = sp.integrate( (V1  / sp.cos(theta)), x)
M2 = sp.integrate( (V2  / sp.cos(theta)), x)
## sfaaf 2
M3 = sp.integrate((V3 / sp.cos(alpha)), x)
M4 = sp.integrate((V4 / sp.cos(alpha)), x)
M  = M1 + M2 + M3+ M4 + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw

N1 = sp.integrate(sp.sin(theta) * (-qx1 + qz1), x) + Cn
N2 = sp.integrate(sp.sin(alpha) * (-qx2 + qz2), x)
N = N1+ N2 + Cn
```

```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)
eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)
eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l)
eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) +1 )
equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, MA, Cn, Ah))
display(solutions)
```

```
{C_v: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 10.9840850145227,
 B_v: 8.99190939177049,
 M_A: 15.9502774939007,
 C_n: 0.0,
 A_h: 0.00962195189033341}
```

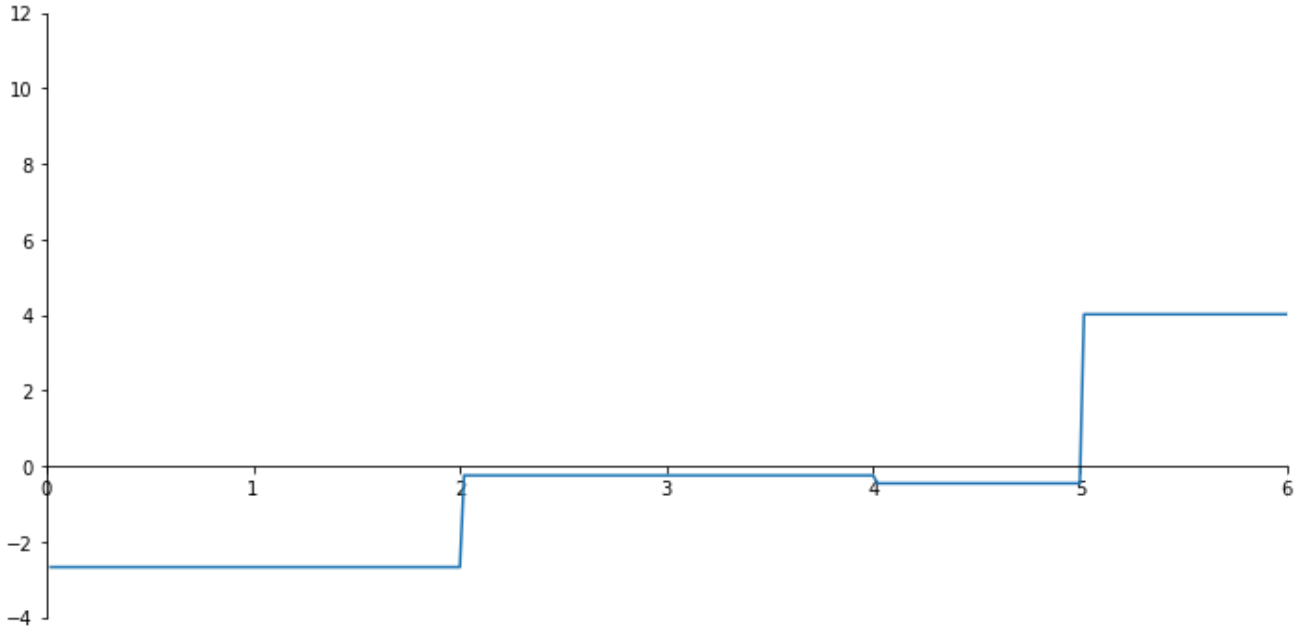
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)
print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list));
```

```
[0.95411928 0.90166365]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-4, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list));
```

```
[-0.24100934 -0.45083183]
```



Iteratie 4

```
```

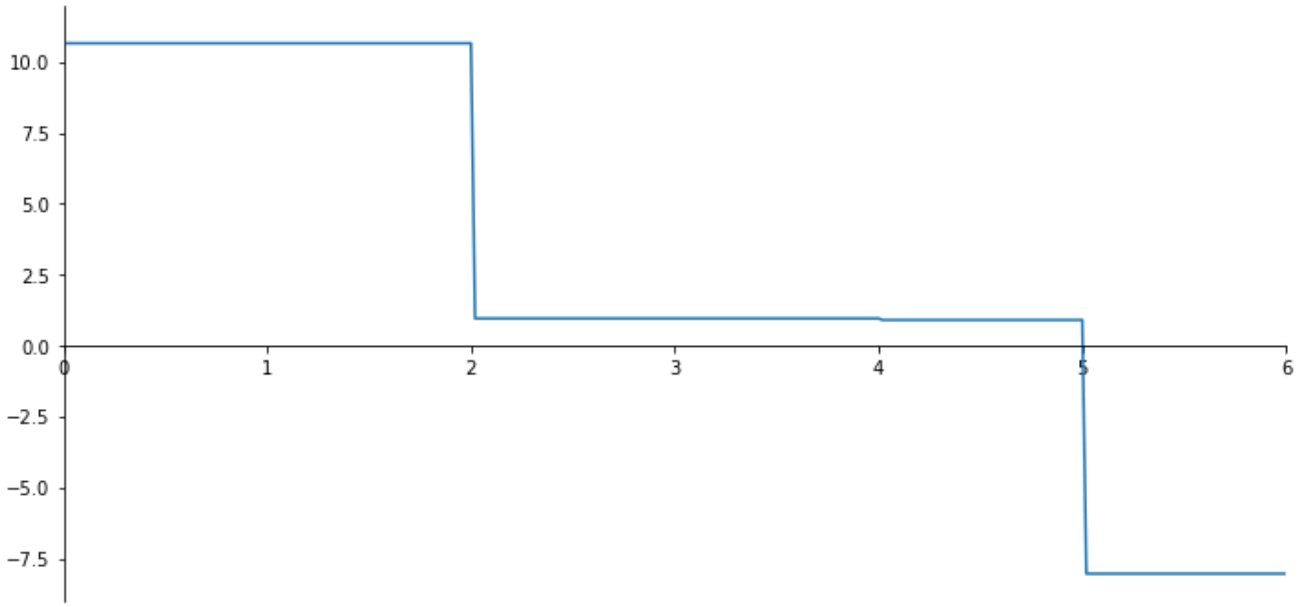
```
Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('C_v, C_m, C_phi, C_w, A_v, B_v, M_A, A_

# Define qz and qx
DwaL =  0.95411928
DwaR =  0.90166365
NorL = -0.24100934
NorR = -0.45083183
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1) + DwaL * sf
qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1) - DwaR *sf(x, l, -1)
qx1 = sp.nsimplify(Ah * sf(x, 0, -1) + DwaL * sf(x, l, -1) * sp.sin(theta)  + NorL * sf(x
qx2 = sp.nsimplify(-DwaR * sf(x, l, -1)*sp.sin(alpha) -NorR * sf(x, l, -1)* sp.cos(alpha)


# Define V as a function of x
## staaf 1
V1 = sp.cos(theta) * sp.integrate( -qz1 , x)
V2 = sp.integrate(- (sp.sin(theta) * sp.tan(theta) * qx1), x)
## staaf 2
V3 = sp.cos(alpha) * sp.integrate( -qz2 , x)
V4 = sp.integrate(- (sp.sin(alpha) * sp.tan(alpha) * qx2), x)

V =V1 + V2 + V3 + V4 + Cv

# Define M as an integral of V
## staaf 1
M1 = sp.integrate( (V1  / sp.cos(theta)), x)
M2 = sp.integrate( (V2  / sp.cos(theta)), x)
## sfaaf 2
M3 = sp.integrate((V3 / sp.cos(alpha)), x)
M4 = sp.integrate((V4 / sp.cos(alpha)), x)
M  = M1 + M2 + M3+ M4 + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw

N1 = sp.integrate(sp.sin(theta) * (-qx1 + qz1), x) + Cn
N2 = sp.integrate(sp.sin(alpha) * (-qx2 + qz2), x)
N = N1 + N2 + Cn
```

```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)
eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)
eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l)
eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) +1 )
equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, MA, Cn, Ah))
display(solutions)
```

```
{C_v: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 10.9838786097141,
 B_v: 8.99212083108564,
 M_A: 15.9507502786053,
 C_n: 0.0,
 A_h: 0.00222200751313955}
```

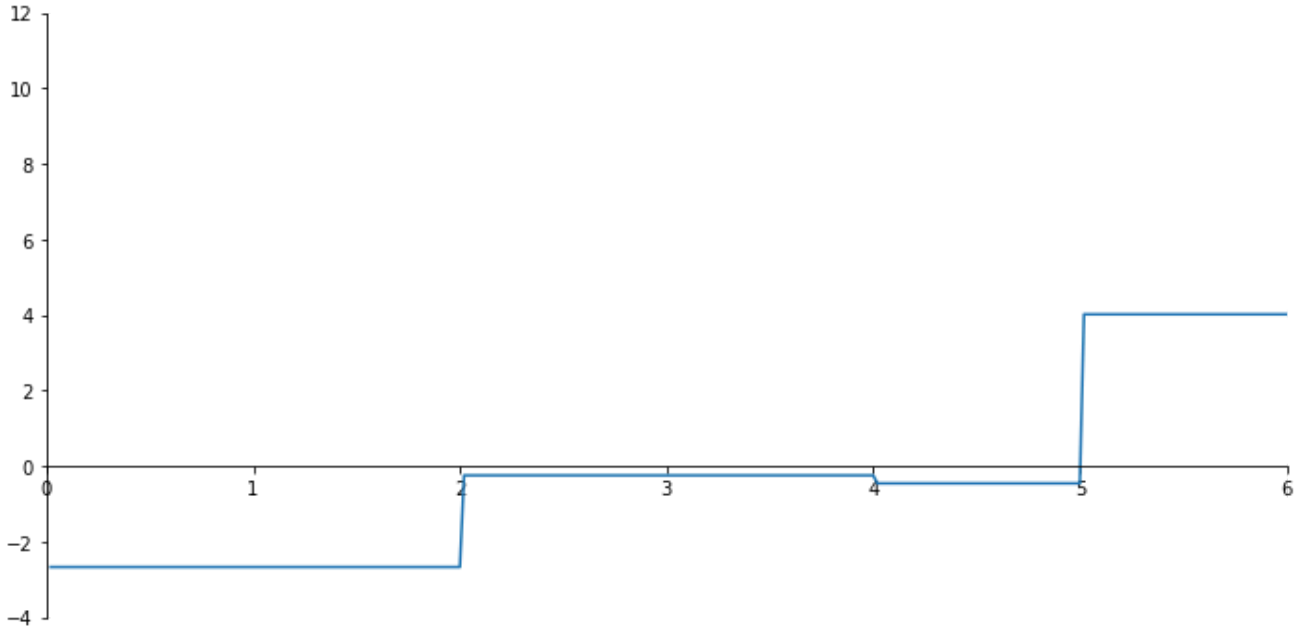
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)
print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list));
```

```
[0.95436773 0.90147453]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-4, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list));
```

```
[-0.23916453 -0.45073727]
```



Iteratie 5

```
```

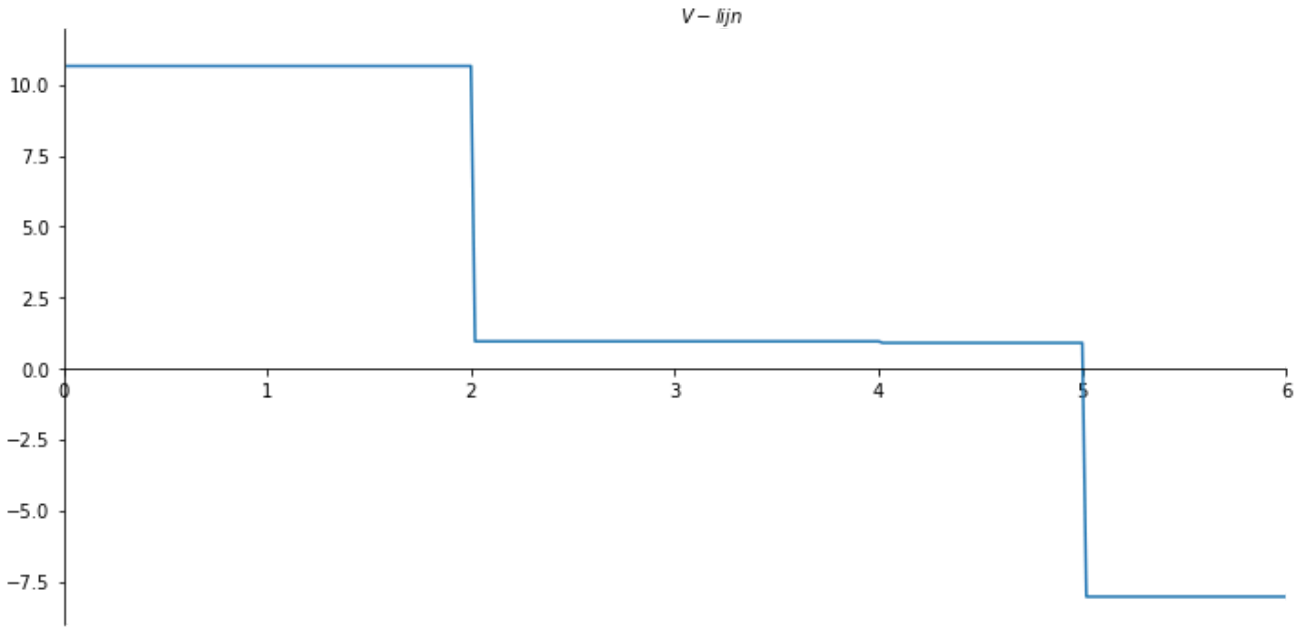
```
Cv, Cm, Cphi, Cw, Av, Bv, MA, Ah, Cn= sp.symbols('C_v, C_m, C_phi, C_w, A_v, B_v, M_A, A_

# Define qz and qx
DwaL =  0.95436773
DwaR =  0.90147453
NorL = -0.23916453
NorR = -0.45073727
qz1 = sp.nsimplify(-Av * sf(x, 0, -1) + MA* sf(x, 0, -2) + F1 *sf(x, l/2, -1) + DwaL * sf
qz2 = sp.nsimplify(-Bv * sf(x, (3/2)*l, -1) + F2 *sf(x, (5/4)*l, -1) - DwaR *sf(x, l, -1)
qx1 = sp.nsimplify(Ah * sf(x, 0, -1) + DwaL * sf(x, l, -1) * sp.sin(theta)  + NorL * sf(x
qx2 = sp.nsimplify(-DwaR * sf(x, l, -1)*sp.sin(alpha) -NorR * sf(x, l, -1)* sp.cos(alpha)


# Define V as a function of x
## staaf 1
V1 = sp.cos(theta) * sp.integrate( -qz1 , x)
V2 = sp.integrate(- (sp.sin(theta) * sp.tan(theta) * qx1), x)
## staaf 2
V3 = sp.cos(alpha) * sp.integrate( -qz2 , x)
V4 = sp.integrate(- (sp.sin(alpha) * sp.tan(alpha) * qx2), x)

V =V1 + V2 + V3 + V4 + Cv

# Define M as an integral of V
## staaf 1
M1 = sp.integrate( (V1  / sp.cos(theta)), x)
M2 = sp.integrate( (V2  / sp.cos(theta)), x)
## sfaaf 2
M3 = sp.integrate((V3 / sp.cos(alpha)), x)
M4 = sp.integrate((V4 / sp.cos(alpha)), x)
M  = M1 + M2 + M3+ M4 + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw

N1 = sp.integrate(sp.sin(theta) * (-qx1 + qz1), x) + Cn
N2 = sp.integrate(sp.sin(alpha) * (-qx2 + qz2), x)
N = N1 + N2 + Cn
```

```
eq1 = V.subs(x, -1)
eq2 = M.subs(x, -1)
eq3 = V.subs(x, (3/2)*l + 1)
eq4 = phi.subs(x, 0)
eq5 = W.subs(x, 0)
eq6 = M.subs(x, (3/2)*l)
eq7 = W.subs(x, (3/2)*l)
eq8 = N.subs(x, -0.01)
eq9 = N.subs(x, ((3/2)*l) +1 )
equations = [eq1-0, eq2-0, eq3-0, eq4-0, eq5-0, eq6-0, eq7 -0, eq8-0, eq9-0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, MA, Cn, Ah))
display(solutions)
```

```
{C_v: 0.0,
 C_m: 0.0,
 C_phi: 0.0,
 C_w: 0.0,
 A_v: 10.9838298415814,
 B_v: 8.99217079631337,
 M_A: 15.9508575377216,
 C_n: 0.0,
 A_h: 0.000512138382148610}
```

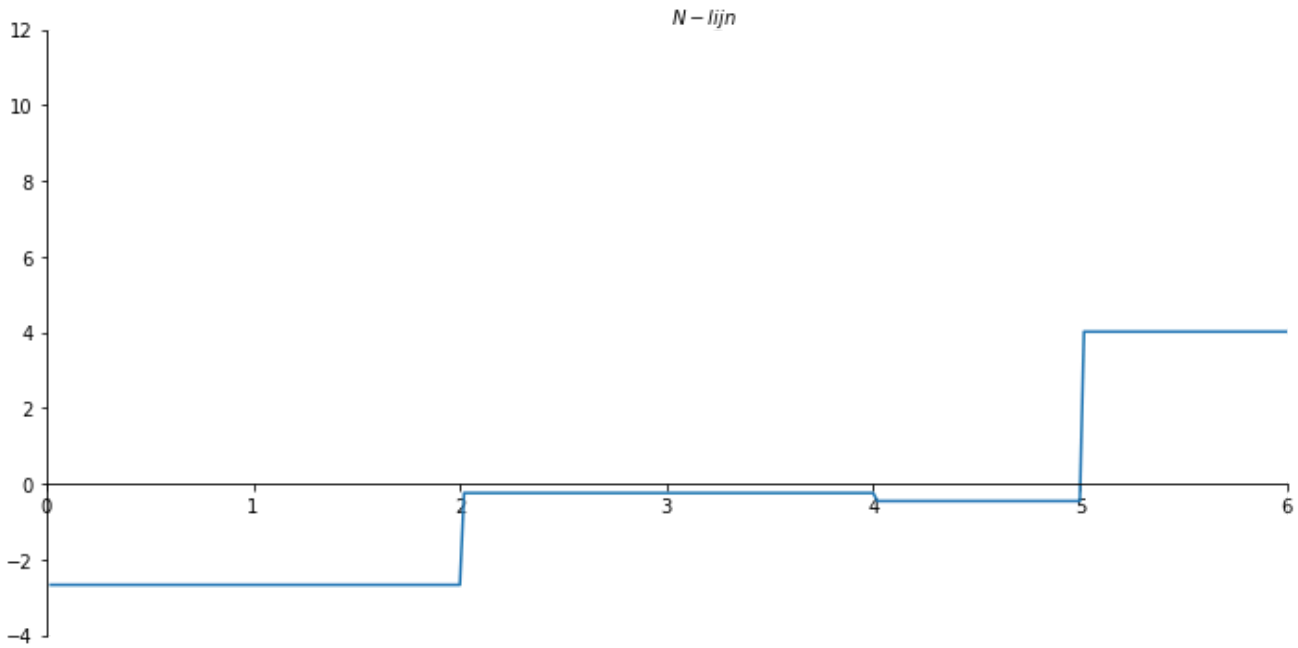
```
x_val = np.linspace(0, 3*l/2, 301)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify())
V_list = V_numpy(x_val)
print(V_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-9, 12)
ax.set_xlabel("$V-lijn$")
ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(V_list))
plt.savefig('V_lijnvoorbeeld4', dpi=450);
```

```
[0.95442409 0.90142984]
```



```
x_val = np.linspace(0, 3*l/2, 301)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify())
N_list = N_numpy(x_val)

print(N_list[200:202])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-4, 12)
ax.set_xlabel("$N-lijn$")
ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, np.array(N_list))
plt.savefig('N_lijnvoorbeeld4', dpi=450);
```

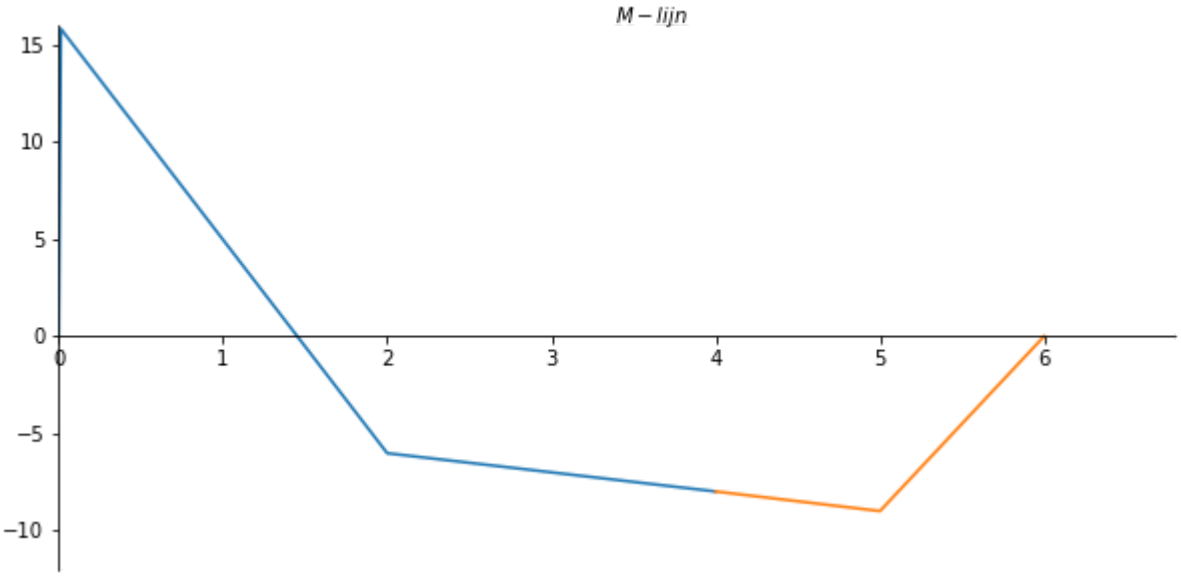
```
[-0.238738   -0.45071492]
```




```
x_val1 = np.linspace(0, l, 301)
M_numpy1 = sp.lambdify(x,M.subs(solutions).rewrite(sp.Piecewise).simplify())
M_list1 = M_numpy1(x_val1)

x_val2 = np.linspace(l, 3*l/2, 301)
M_numpy2 = sp.lambdify(x,M.subs(solutions).rewrite(sp.Piecewise).simplify())
M_list2 = M_numpy2(x_val2)
```

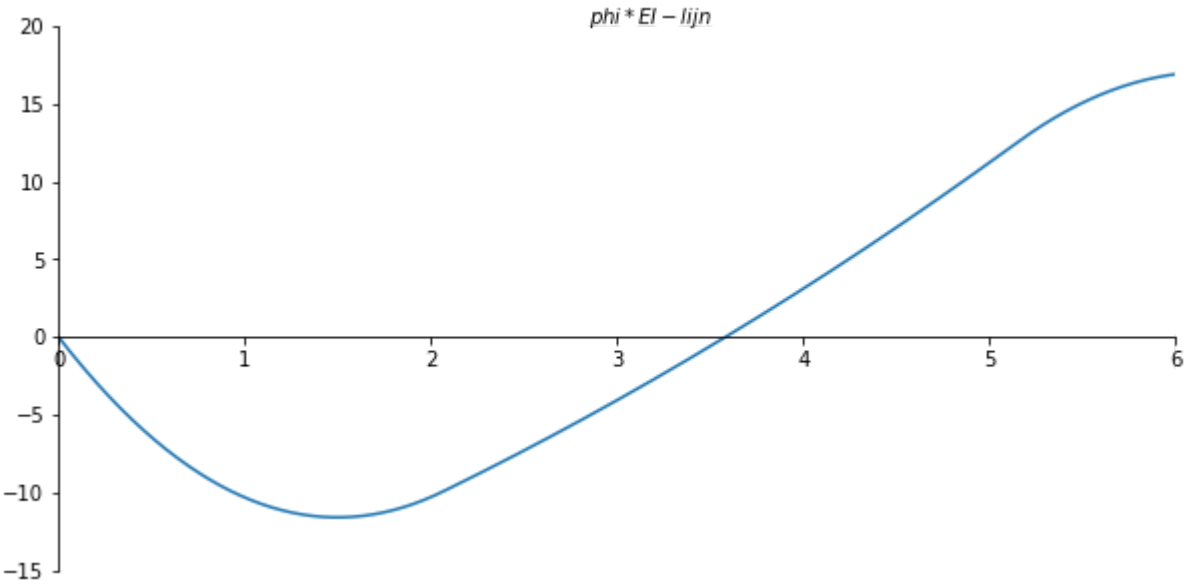
```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6.8)
ax.set_ylim(-12, 16)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val1,  -np.array(M_list1) , label='M-lijn')
ax.plot(x_val2,  -np.array(M_list2) , label='M-lijn');
plt.savefig('M_lijnvoorbeeld4', dpi=450)
;
```

```
''
```



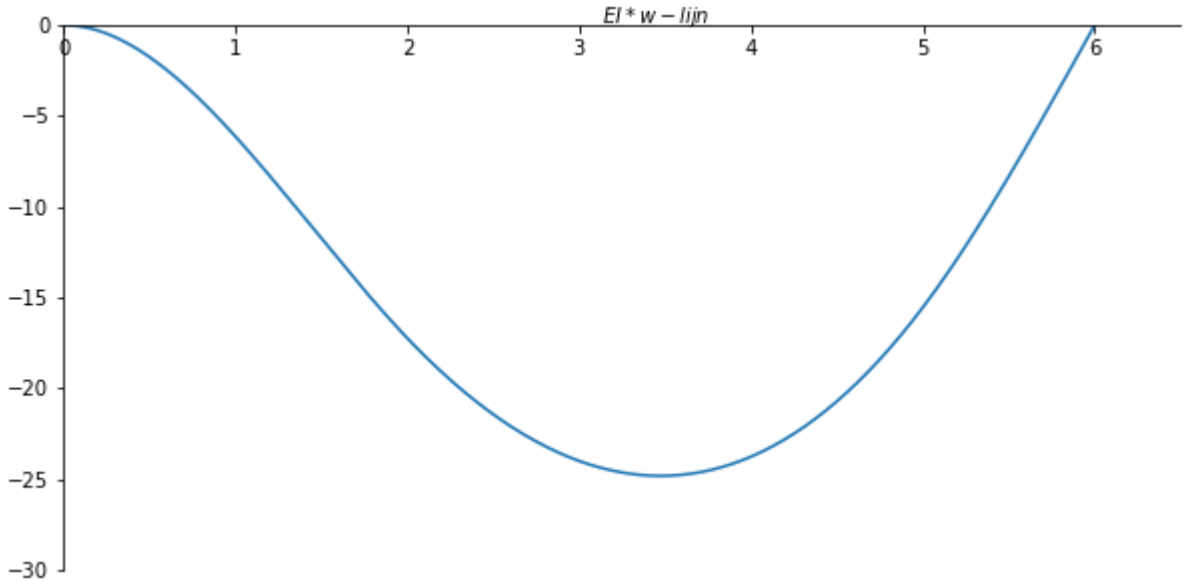

```
x_val = np.linspace(0, 3*l/2, 901)
phi_numpy = sp.lambdify(x,phi.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitu
phi_list = phi_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6)
ax.set_ylim(-15, 20)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$phi*EI-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(phi_list) , label='phi-lijn')
plt.savefig('phi_lijnvoorbeeld4', dpi=450);
```




```
x_val = np.linspace(0, 3*l/2, 901)
W_numpy = sp.lambdify(x,W.subs(solutions).rewrite(sp.Piecewise).simplify())
W_list = W_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 6.5)
ax.set_ylim(-30, 0)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI * w-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val, - np.array(W_list) , label='w-lijn')
plt.savefig('w_lijnvoorbeeld4', dpi=450);
```


EI * w – lijn
0
−5
−10
−15
−20
−25
−30
0
1
2
3
4
5
6

Varianten

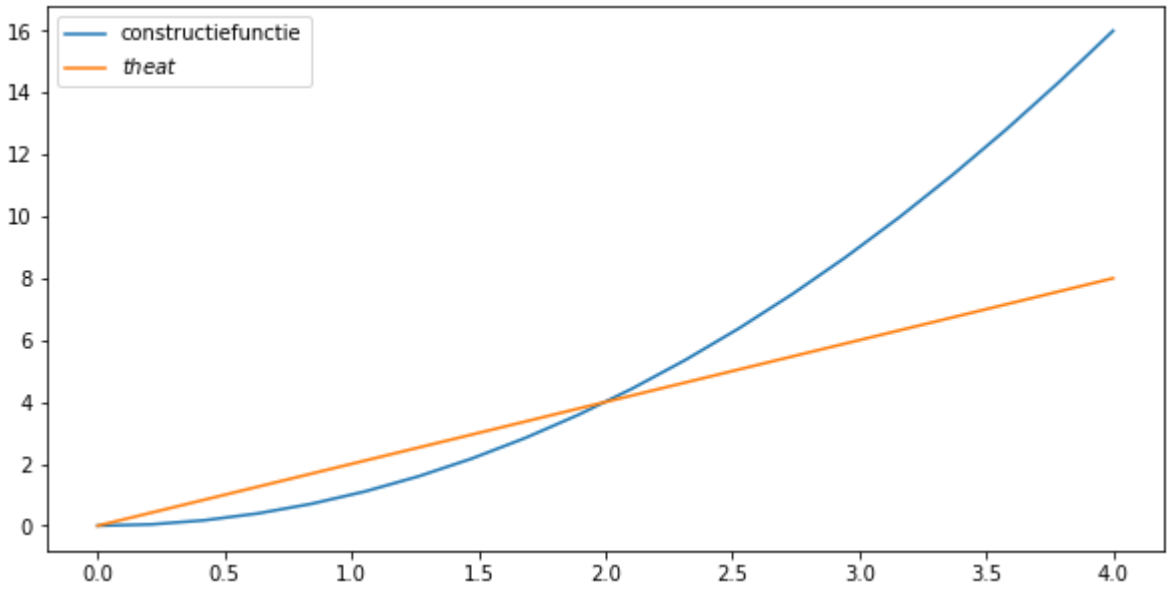
Contents


```
import sympy as sp
import numpy as np
from sympy import symbols
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

```
x = np.linspace(0, 4, 20)
```

```
y = x**2
hell = 2*x
```

```
fig = plt.figure(figsize=(10, 5))
plt.plot(x, y, label='constructiefunctie')
plt.plot(x, hell, label="$theat$")
plt.legend()
plt.savefig('gelobale theta', dpi=450);
```



```
EI = symbols('EI')
x = symbols('x')
```

```
Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn = sp.symbols('Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn')

# Define F and l
F = 10  ## KN
l = 4   ## m
theta = x/10

# Define qz and qx
qz = -Av * sf(x, 0, -1) + F * sf(x, l/2, -1) - Bv * sf(x, l, -1)
qx = Ah * sf(x, 0, -1)
```

Variant 1

```
#V = sp.integrate(-qx * sp.sin(theta), x) + sp.integrate(-qz * sp.cos(theta), x) + Cv
#N = sp.integrate(-qx * sp.cos(theta), x) + sp.integrate(qz * sp.sin(theta), x) + Cn
#M = sp.integrate(V/sp.cos(theta), x) + Cm
#phi = sp.integrate(M, x) + Cphi
#W = sp.integrate(-phi, x) + Cw

## Pyhton Reageert niet meer
```

Variant 2

```
cos_theta = 1 - (theta**2/sp.factorial(2)) + (theta**4/sp.factorial(4)) - (theta**6/sp.fa
sin_theta = theta - (theta**3/sp.factorial(3)) + (theta**5/sp.factorial(5)) - (theta**7/s
```

```
#V = sp.integrate(-qx * sin_theta, x) + sp.integrate(-qz * cos_theta, x) + Cv
#N = sp.integrate(-qx * cos_theta, x) + sp.integrate(qz * sin_theta, x) + Cn

#M = sp.integrate(V/cos_theta, x) + Cm
#phi = sp.integrate(M, x) + Cphi
#W = sp.integrate(-phi, x) + Cw

## python geeft een error bij het integreren
```

Variant 3

```

```

```
#V = sp.integrate(-qx , x) * sp.integrate(sp.sin(theta), x) + sp.integrate(-qz, x)* sp.in
#N = sp.integrate(-qx, x)* sp.integrate(sp.cos(theta), x) + sp.integrate(qz, x) * sp.inte
#M = sp.integrate(V, x) / (sp.integrate(sp.cos(theta), x)) + Cm
#phi = sp.integrate(M, x) + Cphi
#W = sp.integrate(-phi, x) + Cw
```

Variant 4

```
#V = sp.integrate(-qx , x) * sp.integrate(sin_theta, x) + sp.integrate(-qz, x)* sp.integr
#N = sp.integrate(-qx, x)* sp.integrate(cos_theta, x) + sp.integrate(qz, x) * sp.integrat
#M = sp.integrate(V, x) / (sp.integrate(cos_theta, x)) + Cm
#phi = sp.integrate(M, x) + Cphi
#W = sp.integrate(-phi, x) + Cw
```

Voorbeelden 1 t/m 3

Contents


```
import sympy as sp
import numpy as np
from sympy import symbols
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

```
EI = symbols('EI')
x = symbols('x')
```

```
## het oplossen van differentiaal vergelijking
## in x-richting
# Define the symbols
cv, cn, qx, qz, theta = sp.symbols('cv, cn, qx, qz, theta')  ## cv = dv/dx, cn = dn/dx

# Define the equations
eq1 = (qx* sp.tan(theta)) + (sp.sin(theta) * cv) + (sp.cos(theta) * cn)
eq2 = (qz) + (sp.cos(theta) * cv) - (sp.sin(theta) * cn)

# Solve the system of equations
solution = sp.solve((eq1, eq2), (cv, cn))

# Display the solution
print("Solution in the x-direction:")
display("dV/dx =", solution[cv])
display("dN/dx =", solution[cn])
```

```
Solution in the x-direction:
```

```
'dV/dx ='
```

$$-qx \sin(\theta) \tan(\theta) - qz \cos(\theta)$$

```
'dN/dx ='
```

$$(-qx + qz)\sin(\theta)$$

```
## het oplossen van differentiaal vergelijking

## in x-richting

# Define the symbols
cv, cn, qx, qz, theta = sp.symbols('cv, cn, qx, qz, theta')  ## cv = dv/dx, cn = dn/dx

# Define the equations
eq1 = (qx) + (sp.sin(theta) * cv) + (sp.cos(theta) * cn)
eq2 = (qz) + (sp.cos(theta) * cv) - (sp.sin(theta) * cn)

# Solve the system of equations
solution = sp.solve((eq1, eq2), (cv, cn))

# Display the solution
print("Solution in the x-direction:")
display("dV/dx =", solution[cv])
display("dN/dx =", solution[cn])
```

```
Solution in the x-direction:
```

```
'dV/dx ='
```

$$-qx\sin(\theta) - qz\cos(\theta)$$

```
'dN/dx ='
```

$$-qx\cos(\theta) + qz\sin(\theta)$$

Voorbeeld 1

```
Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, Cu = sp.symbols('Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, Cu')

# F, l en theta definiëren
F = 10  ## KN
l = 4   ## m
theta = sp.atan(1/2)

# qz en qx definiëren
qz = -Av * sf(x, 0, -1) + F * sf(x, l/2, -1) - Bv * sf(x, l, -1)
qx = Ah * sf(x, 0, -1)
```

```
# V definiëren als een functie van x
V = sp.integrate(-qx * sp.sin(theta), x) + sp.integrate(-qz * sp.cos(theta), x) + Cv
# N definiëren als een functie van x
N = sp.integrate(-qx * sp.cos(theta), x) + sp.integrate(qz * sp.sin(theta), x) + Cn
```

```
# N definiëren als een functie van x
M = sp.integrate(V/sp.cos(theta), x) + Cm

# phi definiëren als een functie van x
phi = sp.integrate(M, x) + Cphi

# W definiëren als een functie van x
W = sp.integrate(-phi, x) + Cw
```

```
# u definiëren als een functie van x
u = sp.integrate(N, x) + Cu
```

```
display("V:", V)
display("M:", M)
display("phi:", phi)
display("W:", W)
display("N:", N)
display("u:", u)
```

```
'V:'
```

$$-0.447213595499958Ah\langle x\rangle^{0} + 0.894427190999916Av\langle x\rangle^{0} + 0.894427190999916B$$

```
'M:'
```

$$-0.5Ah\langle x\rangle^{1} + 1.0Av\langle x\rangle^{1} + 1.0Bv\langle x-4\rangle^{1} + Cm + 1.11803398874989Cvx - 10.0\langle$$

```
'phi:'
```

$$-0.25Ah\langle x\rangle^2 + 0.5Av\langle x\rangle^2 + 0.5Bv\langle x-4\rangle^2 + Cmx + Cphi + 0.5590169943749470$$

```
'W:'
```

$$0.0833333333333333Ah\langle x\rangle^3 - 0.166666666666667Av\langle x\rangle^3 - 0.166666666666667Bv$$

```
'N:'
```

$$-0.894427190999916Ah\langle x\rangle^0 - 0.447213595499958Av\langle x\rangle^0 - 0.447213595499958B$$

```
'u:'
```

$$-0.894427190999916Ah\langle x\rangle^1 - 0.447213595499958Av\langle x\rangle^1 - 0.447213595499958B$$

```
## Voorwarden

eq1 = V.subs(x, -1)
eq2 = V.subs(x, l+1)

eq3 = M.subs(x, 0)
eq4 = M.subs(x, l)

eq5 = W.subs(x, l)
eq6 = W.subs(x, 0)

eq7 = N.subs(x, -1)
eq8 = N.subs(x, l+1)

eq9 = u.subs(x, 0)

equations = [eq1 -0, eq2-0,eq3-0,eq4-0,eq5-0,eq6-0, eq7-0, eq8 -0, eq9 - 0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, Cu))
```

```
print(solutions)
```

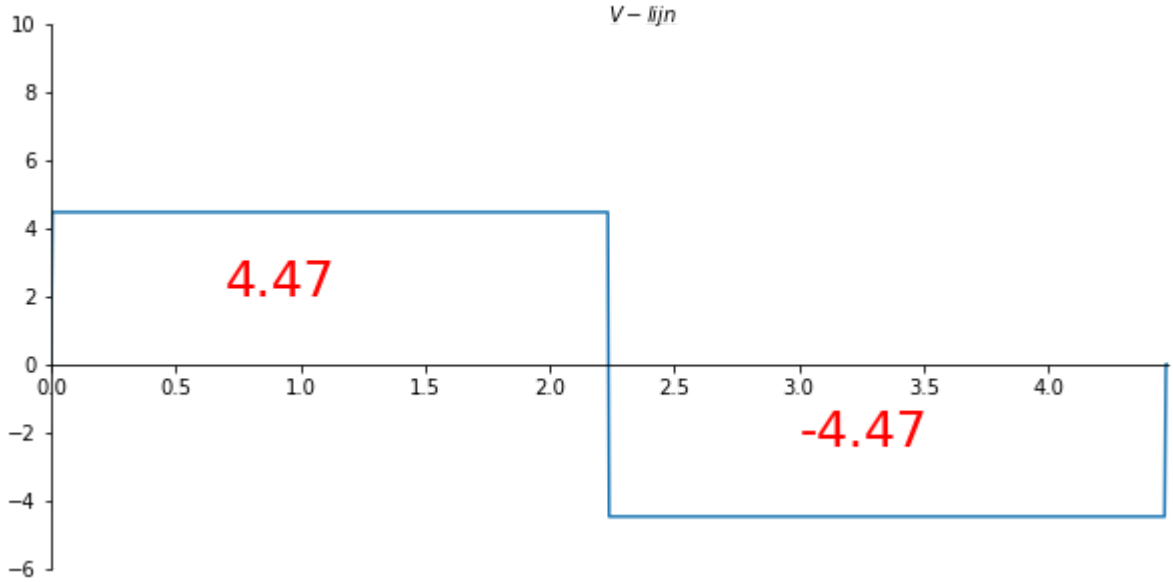
```
{Cv: 0.0, Cm: 0.0, Cphi: -10.0000000000000, Cw: 0.0, Av: 5.00000000000000, Bv: 5.00000000
```

```
l/sp.cos(theta)
```

$$4.47213595499958$$

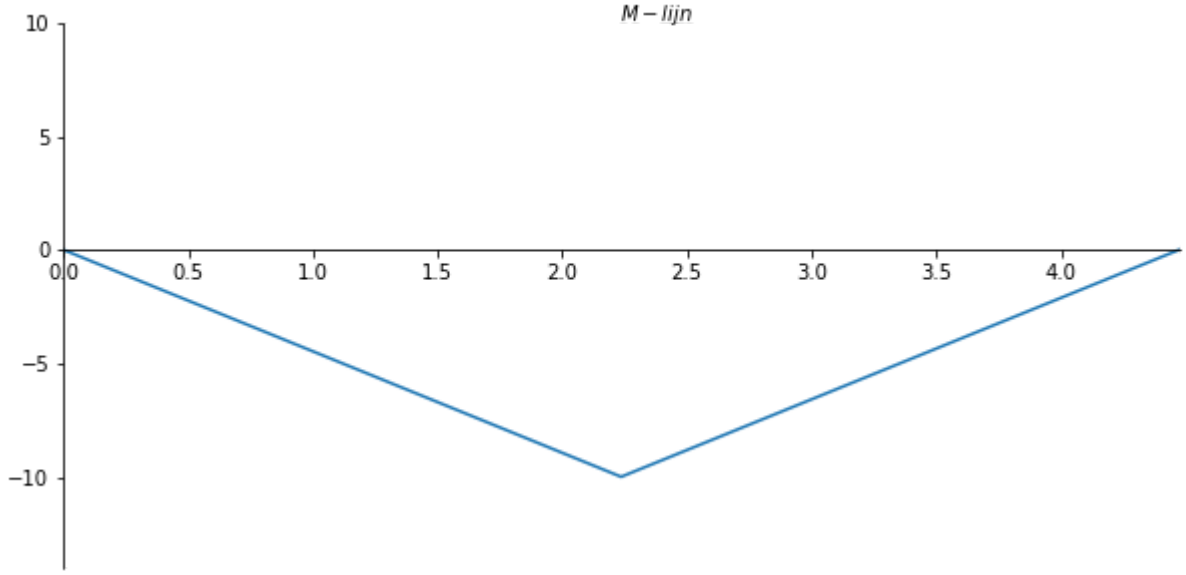
```
x_val = np.linspace(0, 4.47213595499958, 901)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
V_list = V_numpy(x_val)

#print(V_list)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-6, 10)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$V-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(V_list) , label='V-lijn')
plt.text(0.7, 2, f'{V_list[100]:.2f}', fontsize=25, color='red')
plt.text(3, -2.4, f'{V_list[500]:.2f}', fontsize=25, color='red')
plt.savefig('V_lijnvoorbeeld1', dpi=450);
```



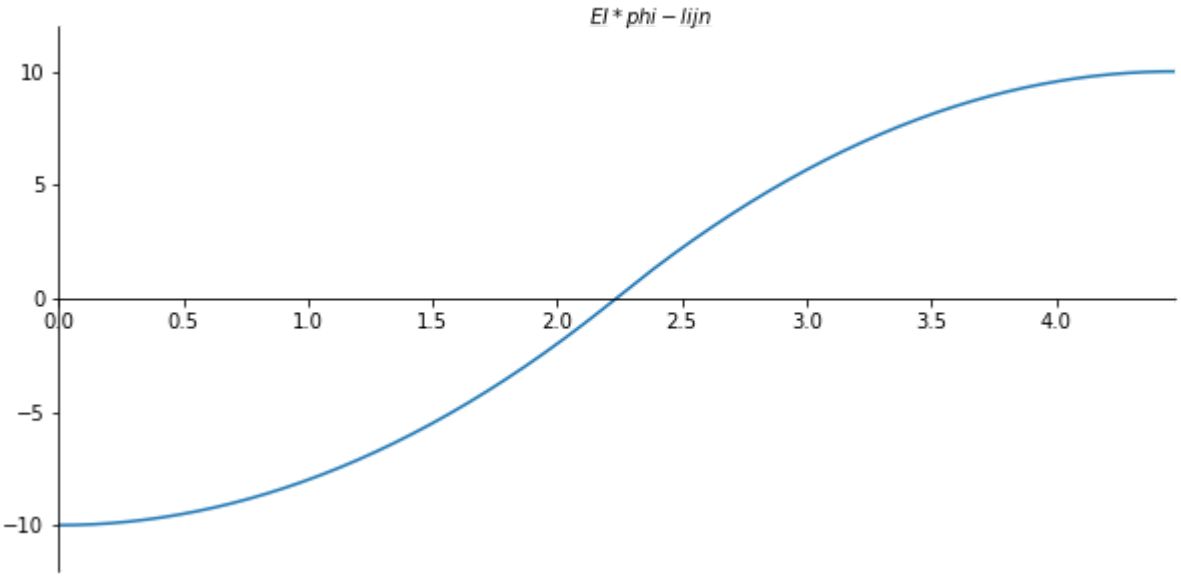
```
x_val = np.linspace(0, l, 901)
M_numpy = sp.lambdify(x,M.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
M_list = M_numpy(x_val)
#print(M_list)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-14, 10)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), - np.array(M_list) , label='M-lijn');
plt.savefig('M_lijnvoorbeeld1', dpi=450);
```



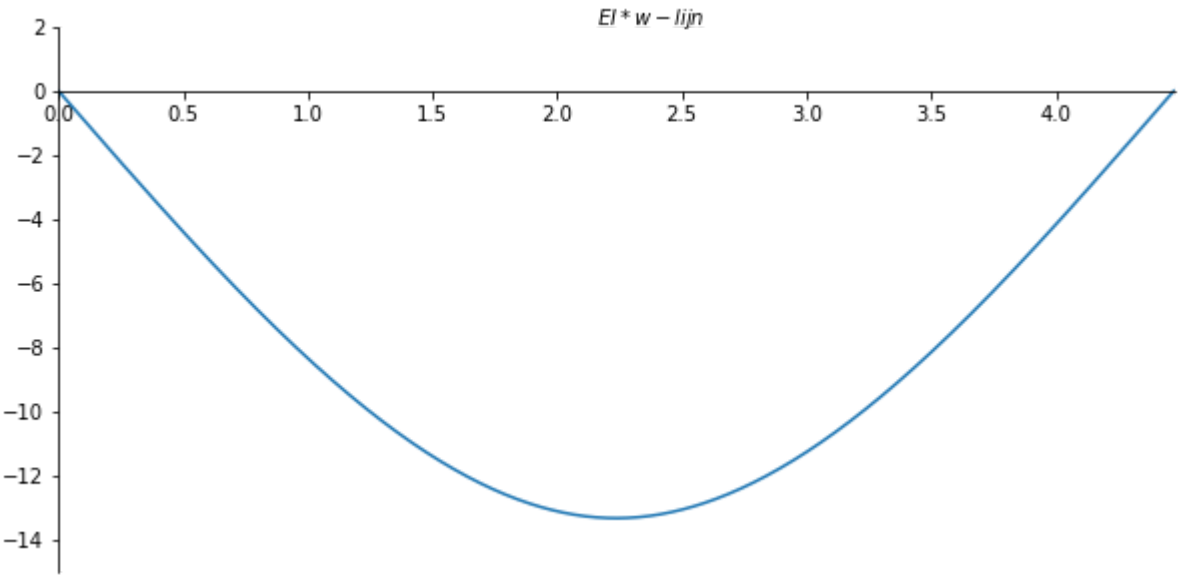

```
x_val = np.linspace(0, l, 901)
phi_numpy = sp.lambdify(x,phi.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitu
phi_list = phi_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-12, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI*phi-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(phi_list) , label='phi-lijn')
plt.savefig('phi_lijnvoorbeeld1', dpi=450);
```



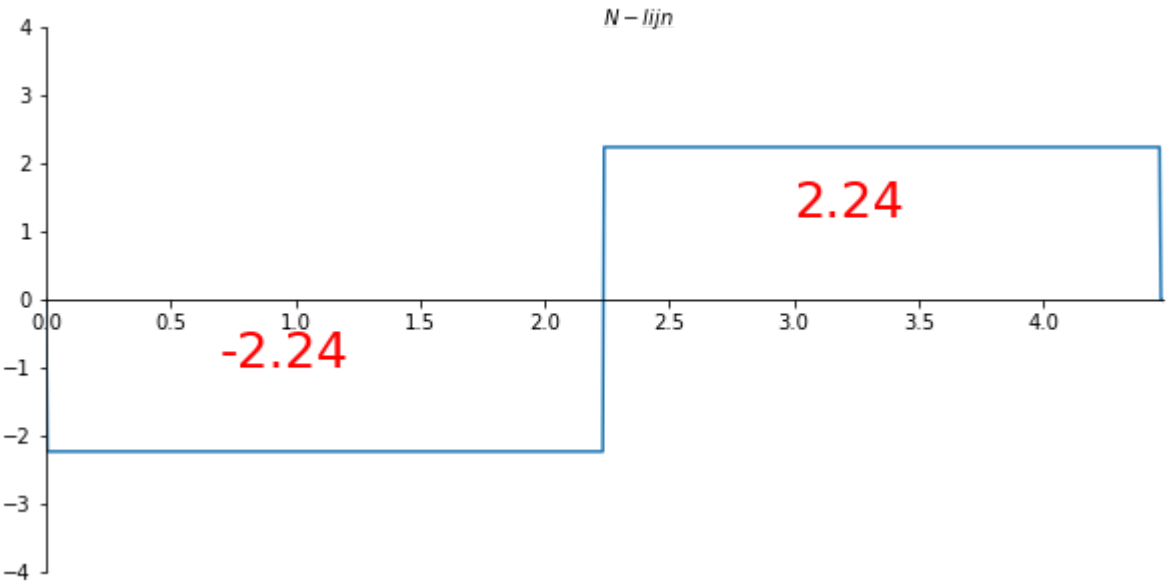

```
x_val = np.linspace(0, l, 901)
W_numpy = sp.lambdify(x,W.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
W_list = W_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-15, 2)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI*w-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), - np.array(W_list) , label='w-lijn')
plt.savefig('w_lijnvoorbeeld1', dpi=450);
```



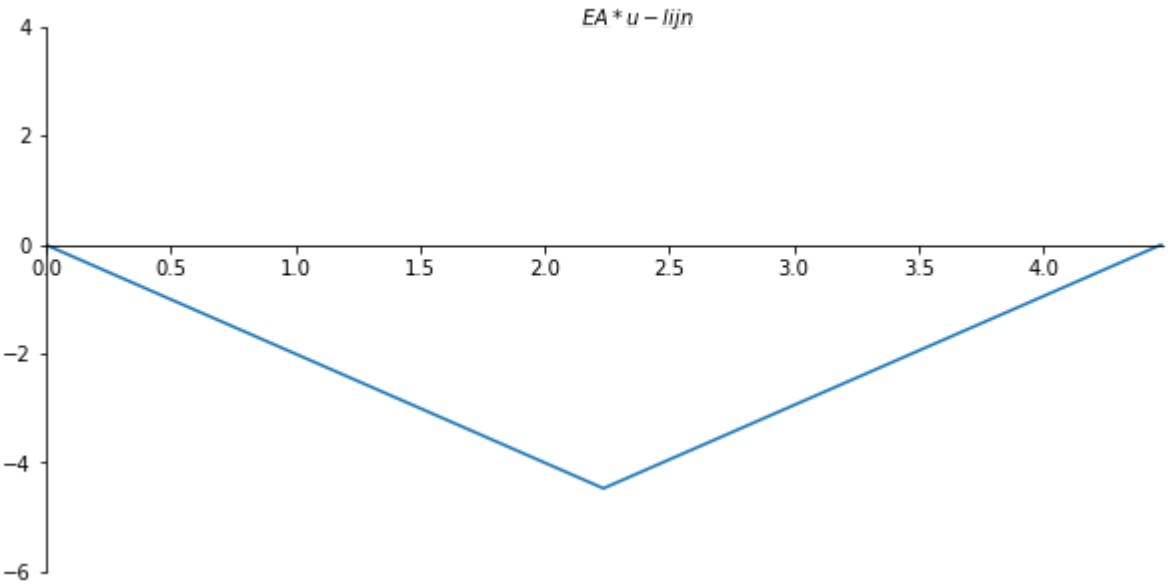

```
x_val = np.linspace(0,  4.47213595499958, 901)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
N_list = N_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-4, 4)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$N-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.text(0.7, -1, f'{N_list[100]:.2f}', fontsize=25, color='red')
plt.text(3, 1.2, f'{N_list[500]:.2f}', fontsize=25, color='red')
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(N_list) , label='N-lijn')
plt.savefig('N_lijnvoorbeeld1', dpi=450);
```




```
x_val = np.linspace(0, l, 301)
u_numpy = sp.lambdify(x,u.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
u_list = u_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-6, 4)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EA*u-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(u_list) , label='u-lijn')
plt.savefig('u_lijnvoorbeeld1', dpi=450);
```



Voorbeeld 2

```
Cv, Cm, Cphi, Cw, Av, MA, Ah, Cn, Cu = sp.symbols('Cv, Cm, Cphi, Cw, Av, MA, Ah, Cn, Cu')

# F, l en theta definiëren
F = 10  ## KN
l = 4   ## m
theta = sp.atan(1/2)

# qz en qx definiëren
qz = -Av * sf(x, 0, -1) + MA * sf(x, 0, -2) + F * sf(x, l, -1)
qx = Ah * sf(x, 0, -1) - F * sf(x, l, -1)
```

```
# V definiëren als een functie van x
V = sp.integrate(-qx * sp.sin(theta), x) + sp.integrate(-qz * sp.cos(theta), x) + Cv
# N definiëren als een functie van x
N = sp.integrate(-qx * sp.cos(theta), x) + sp.integrate(qz * sp.sin(theta), x) + Cn
```

```
# N definiëren als een functie van x
M = sp.integrate(V/sp.cos(theta), x) + Cm

# phi definiëren als een functie van x
phi = sp.integrate(M, x) + Cphi

# W definiëren als een functie van x
W = sp.integrate(-phi, x) + Cw
```

```
# u definiëren als een functie van x
u = sp.integrate(N, x) + Cu
```

```
display("V:", V)
display("M:", M)
display("phi:", phi)
display("W:", W)
display("N:", N)
display("u:", u)
```

```
'V:'
```

$$-0.447213595499958Ah\langle x\rangle^{0} + 0.894427190999916Av\langle x\rangle^{0} + Cv - 0.894427190999$$

```
'M:'
```

$$-0.5Ah\langle x\rangle^{1} + 1.0Av\langle x\rangle^{1} + Cm + 1.11803398874989Cvx - 1.0MA\langle x\rangle^{0} - 5.0\langle x -$$

```
'phi:'
```

$$-0.25Ah\langle x\rangle^2 + 0.5Av\langle x\rangle^2 + Cmx + Cphi + 0.559016994374947Cvx^2 - 1.0MA\langle x$$

```
'W:'
```

$$0.0833333333333333Ah\langle x\rangle^3 - 0.166666666666667Av\langle x\rangle^3 - \frac{Cmx^2}{2} - Cphix - 0.1$$

```
'N:'
```

$$-0.894427190999916Ah\langle x\rangle^0 - 0.447213595499958Av\langle x\rangle^0 + Cn + 0.44721359549$$

```
'u:'
```

$$-0.894427190999916Ah\langle x\rangle^1 - 0.447213595499958Av\langle x\rangle^1 + Cnx + Cu + 0.44721$$

```
## Voorwarden

eq1 = V.subs(x, -1)
eq2 = V.subs(x, l+0.1)

eq3 = M.subs(x, -1)
eq4 = M.subs(x, l)

eq5 = phi.subs(x, 0)
eq6 = W.subs(x, 0)

eq7 = N.subs(x, -1)
eq8 = N.subs(x, l+0.1)
eq9 = u.subs(x, 0)
equations = [eq1 -0, eq2-0,eq3-0,eq4-0,eq5-0,eq6-0, eq7-0, eq8 -0, eq9 -0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, MA, Ah, Cn, Cu))
```

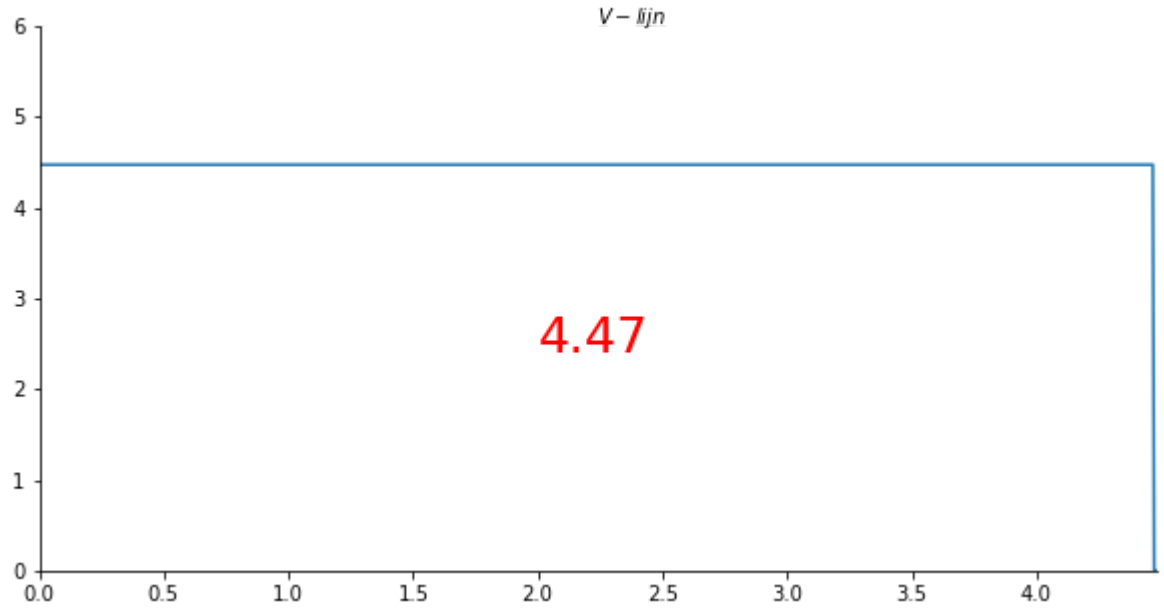
```
print(solutions)
```

```
{Cv: 0.0, Cm: 0.0, Cphi: 0.0, Cw: 0.0, Av: 9.99999999999998, MA: 20.0000000000000, Ah: 9.
```

V-lijn

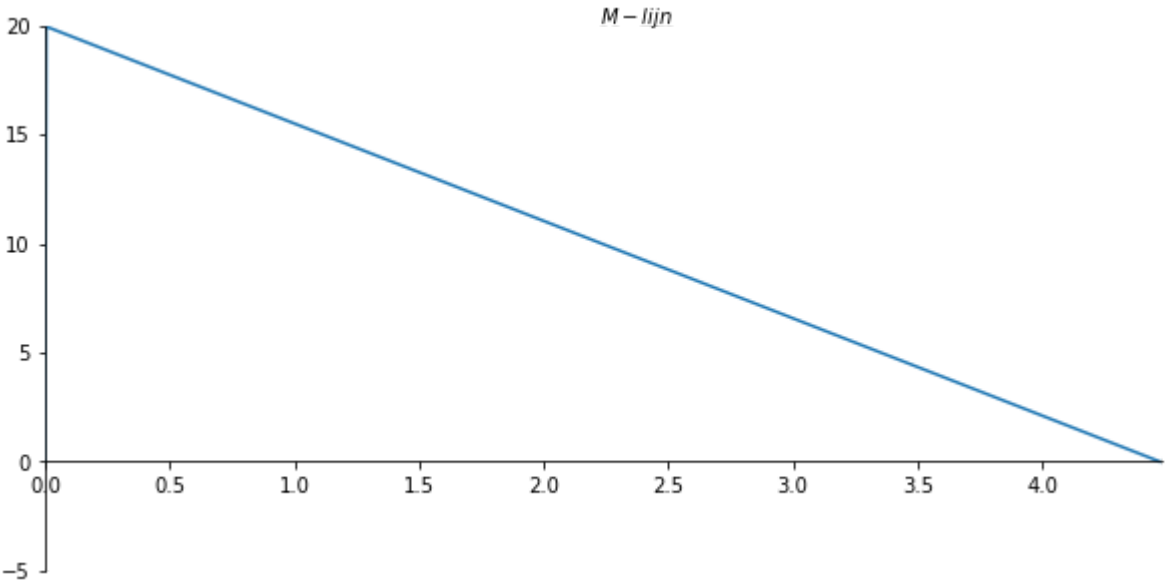
```
x_val = np.linspace(0, 4.47213595499958, 901)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
V_list = V_numpy(x_val)
#print(V_list)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(0, 6)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$V-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.text(2, 2.4, f'{V_list[100]:.2f}', fontsize=25, color='red')
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(V_list) , label='V-lijn')
plt.savefig('V_lijnvoorbeeld2', dpi=450);
```



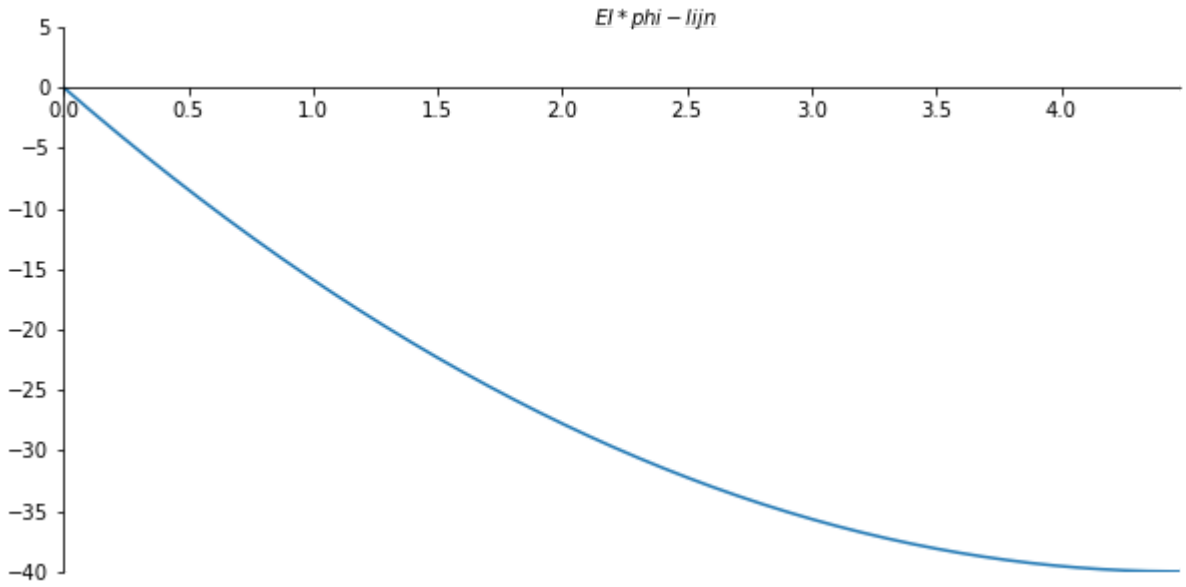

```
x_val = np.linspace(0, l, 901)
M_numpy = sp.lambdify(x,M.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
M_list = M_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-5, 20)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), - np.array(M_list) , label='M-lijn');
plt.savefig('M_lijnvoorbeeld2', dpi=450);
```




```
x_val = np.linspace(0, l, 901)
phi_numpy = sp.lambdify(x,phi.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitu
phi_list = phi_numpy(x_val)
```

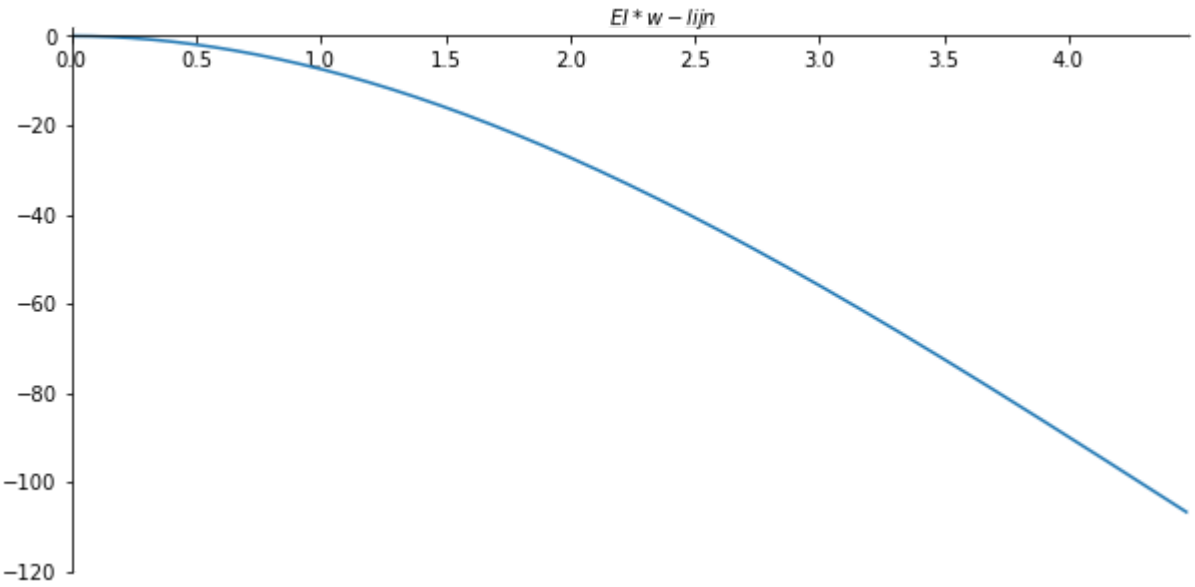
```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-40, 5)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI*phi-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(phi_list) , label='phi-lijn')
plt.savefig('phi_lijnvoorbeeld2', dpi=450);
```



w-lijn

```
x_val = np.linspace(0, l, 901)
W_numpy = sp.lambdify(x,W.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
W_list = W_numpy(x_val)
```

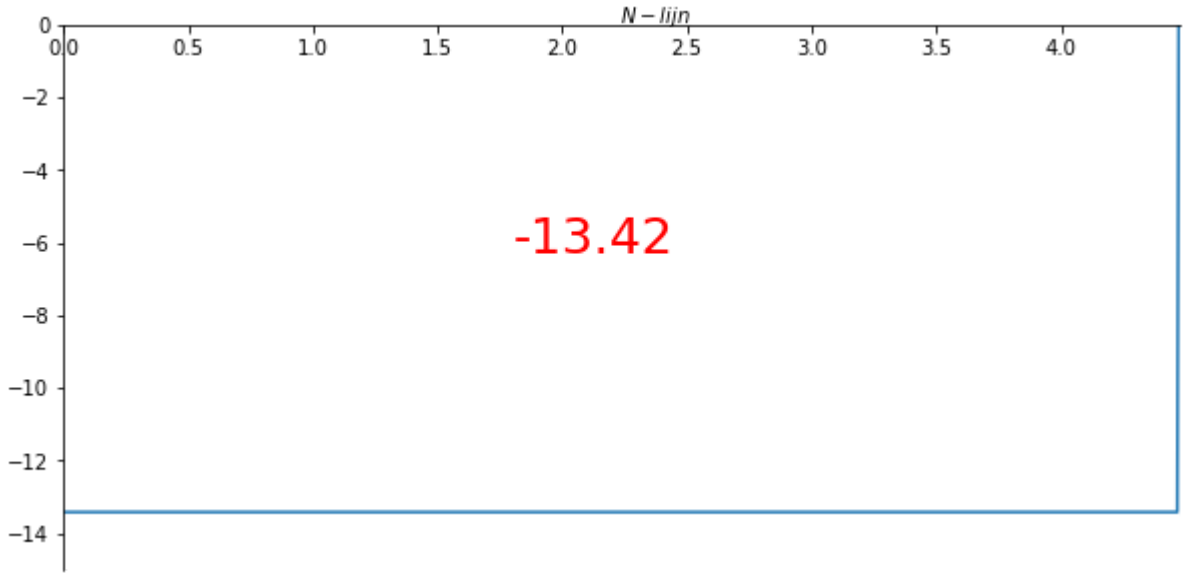
```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-120, 2)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI*w-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), - np.array(W_list) , label='w-lijn')
plt.savefig('w_lijnvoorbeeld2', dpi=450);
```



N-lijn

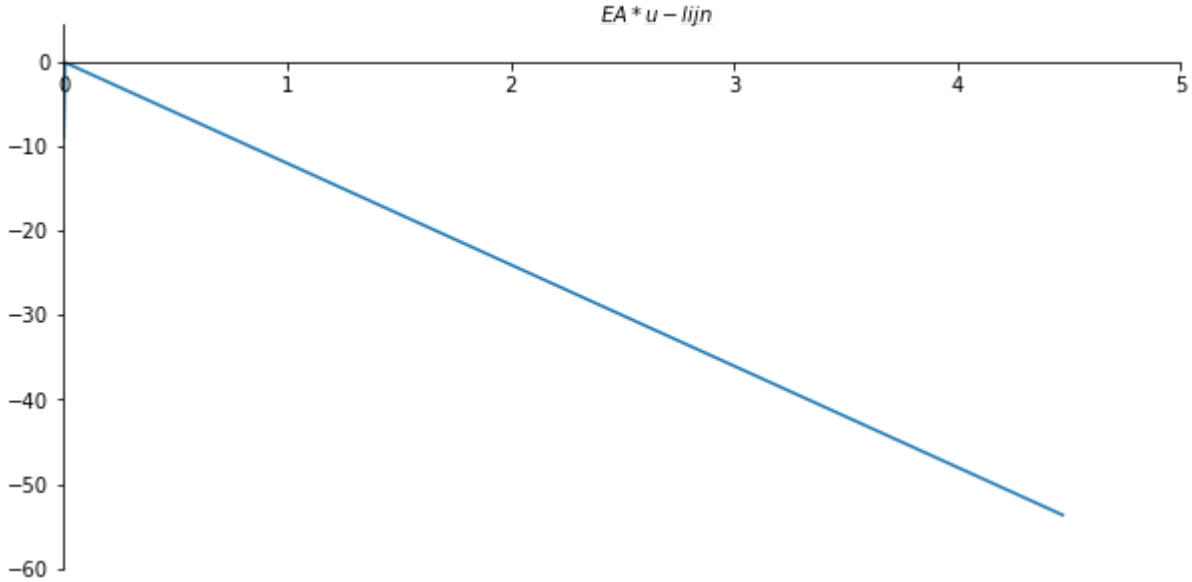
```
x_val = np.linspace(0, 4.47213595499958, 901)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
N_list = N_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-15, 0)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.set_xlabel("$N-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.text(1.8, -6.3, f'{N_list[100]:.2f}', fontsize=25, color='red')
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(N_list) , label='N-lijn')
plt.savefig('N_lijnvoorbeeld2', dpi=450);
```




```
x_val = np.linspace(0, l, 901)
u_numpy = sp.lambdify(x,u.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
u_list = u_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 5)
ax.set_ylim(-60, 4.48)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EA*u-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(u_list) , label='u-lijn')
plt.savefig('u_lijnvoorbeeld2', dpi=450);
```



Voorbeeld 3

```
Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, MA = sp.symbols('Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, MA')

# F, l en theta definiëren
F = 10  ## KN
l = 4   ## m
T = 10
theta = sp.atan(1/2)

# qz en qx definiëren
qz = -Av * sf(x, 0, -1) + MA*sf(x, 0, -2) - Bv * sf(x, l, -1) + T *sf(x, l/4, -2)
qx = Ah * sf(x, 0, -1) -F*sf(x, l/2, -1)
```

```
# V definiëren als een functie van x
V = sp.integrate(-qx * sp.sin(theta), x) + sp.integrate(-qz * sp.cos(theta), x) + Cv
# N definiëren als een functie van x
N = sp.integrate(-qx * sp.cos(theta), x) + sp.integrate(qz * sp.sin(theta), x) + Cn
```

```
# N definiëren als een functie van x
M = sp.integrate(V/sp.cos(theta), x) + Cm

# phi definiëren als een functie van x
phi = sp.integrate(M, x) + Cphi

# W definiëren als een functie van x
W = sp.integrate(-phi, x) + Cw
```

```
# u definiëren als een functie van x
u = sp.integrate(N, x) + Cu
```

```
display("V:", V)
display("M:", M)
display("phi:", phi)
display("W:", W)
display("N:", N)
display("u:", u)
```

```
'V:'
```

$$-0.447213595499958Ah\langle x\rangle^{0} + 0.894427190999916Av\langle x\rangle^{0} + 0.894427190999916B$$

```
'M:'
```

$$-0.5Ah\langle x\rangle^{1} + 1.0Av\langle x\rangle^{1} + 1.0Bv\langle x-4\rangle^{1} + Cm + 1.11803398874989Cvx - 1.0M$$

```
'phi:'
```

$$-0.25Ah\langle x\rangle^{2} + 0.5Av\langle x\rangle^{2} + 0.5Bv\langle x-4\rangle^{2} + Cmx + Cphi + 0.5590169943749470$$

```
'W:'
```

$$0.0833333333333333Ah\langle x\rangle^{3} - 0.166666666666667Av\langle x\rangle^{3} - 0.166666666666667Bv$$

```
'N:'
```

$$-0.894427190999916Ah\langle x\rangle^{0} - 0.447213595499958Av\langle x\rangle^{0} - 0.447213595499958B$$

```
'u:'
```

$$-0.894427190999916Ah\langle x\rangle^{1} - 0.447213595499958Av\langle x\rangle^{1} - 0.447213595499958B$$

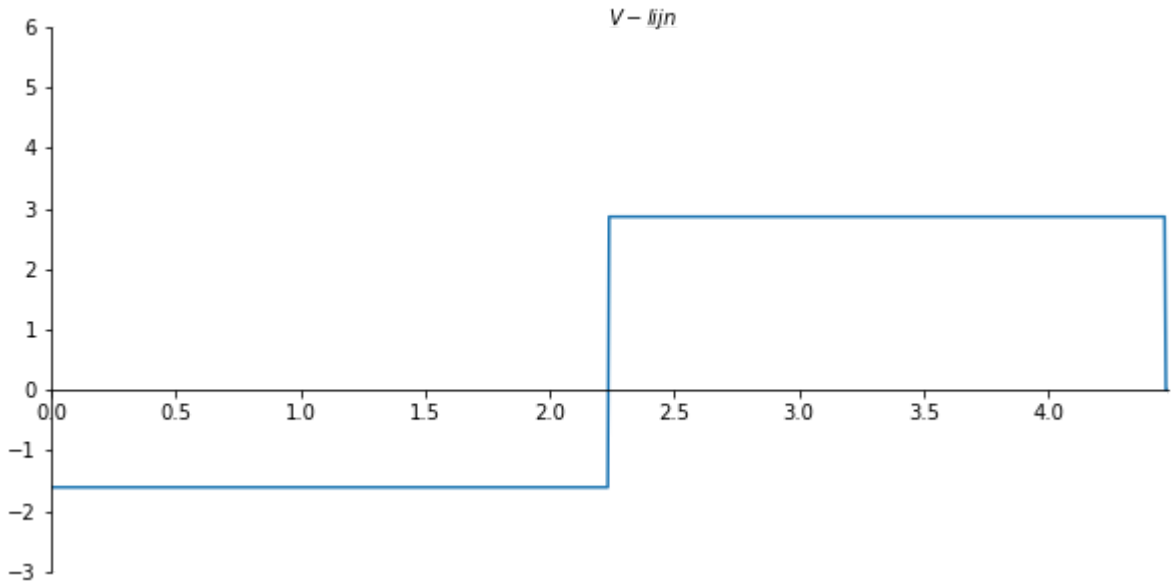
```
eq1 = V.subs(x, -1)
eq2 = V.subs(x, l+1)

eq3 = M.subs(x, -1)
eq4 = M.subs(x, l)

eq5 = W.subs(x, 0)
eq6 = phi.subs(x, 0)

eq7 = N.subs(x, -1)
eq8 = W.subs(x, l)
eq9 = N.subs(x, l+1)

eq10 = u.subs(x, 0)
equations = [eq1 - 0, eq2 - 0 , eq3 - 0, eq4 - 0, eq5 - 0, eq6 - 0, eq7 - 0, eq8 -0, eq9
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, MA, Cn, Ah, Bv, Cu))
```

```
print(solutions)
```

```
{Cv: 0.0, Cm: 0.0, Cphi: 0.0, Cw: 0.0, Av: 3.20312500000000, MA: -7.18750000000000, Cn: 0
```

```
x_val = np.linspace(0, 4.4721359549995, 901)
V_numpy = sp.lambdify(x,V.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
V_list = V_numpy(x_val)
```

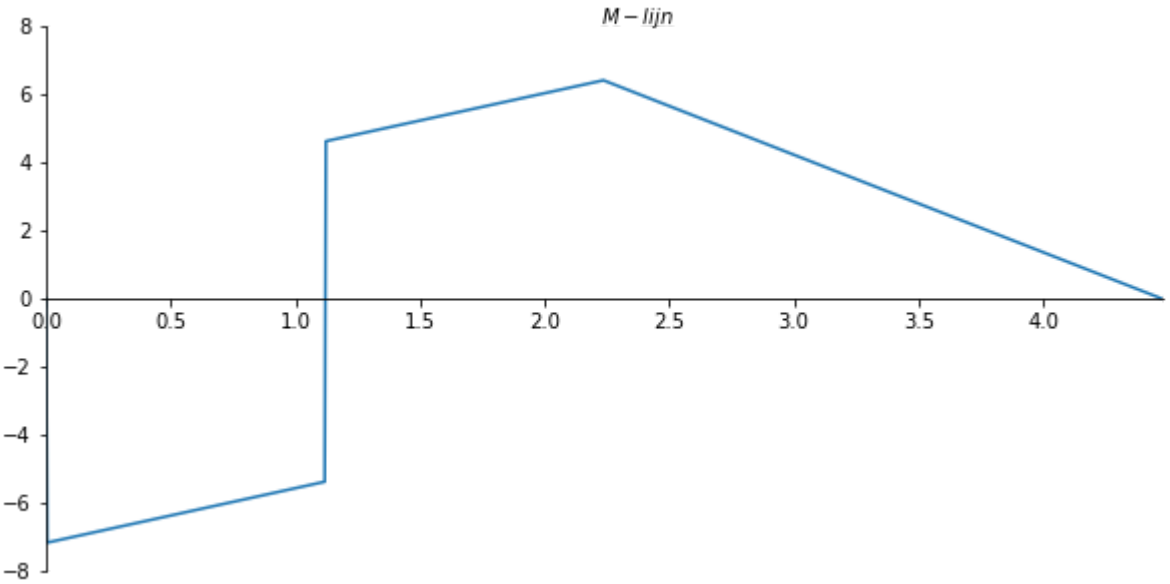
```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-3, 6)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$V-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(V_list) , label='V-lijn')
plt.savefig('V_lijnvoorbeeld3', dpi=450);
```




```
x_val = np.linspace(0, l, 901)
M_numpy = sp.lambdify(x,M.subs(solutions).rewrite(sp.Piecewise).simplify())
M_list = M_numpy(x_val)
```

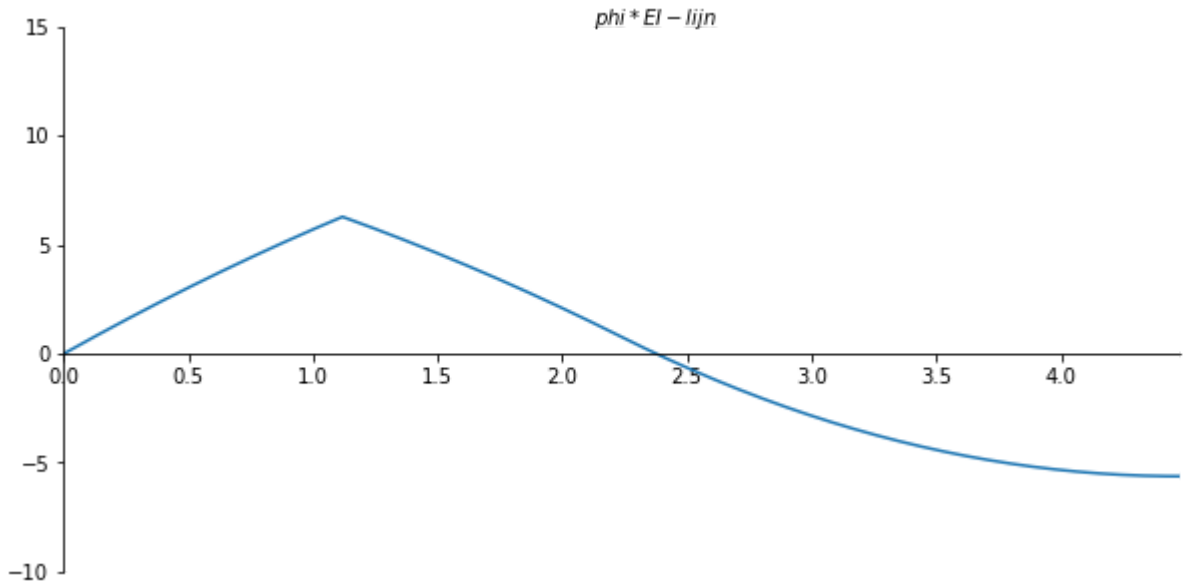
```
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-8, 8)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  -np.array(M_list) , label='M-lijn');
plt.savefig('M_lijnvoorbeeld3', dpi=450);
```



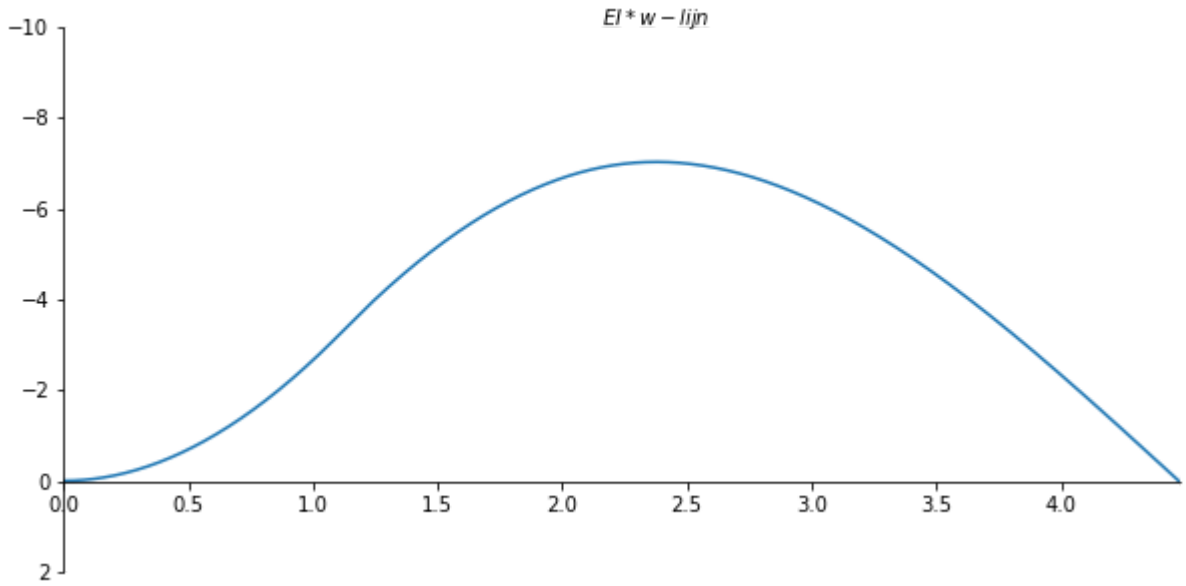

```
x_val = np.linspace(0, l, 901)
phi_numpy = sp.lambdify(x,phi.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitu
phi_list = phi_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-10, 15)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$phi*EI-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta), np.array(phi_list) , label='phi-lijn')
plt.savefig('phi_lijnvoorbeeld3', dpi=450);
```



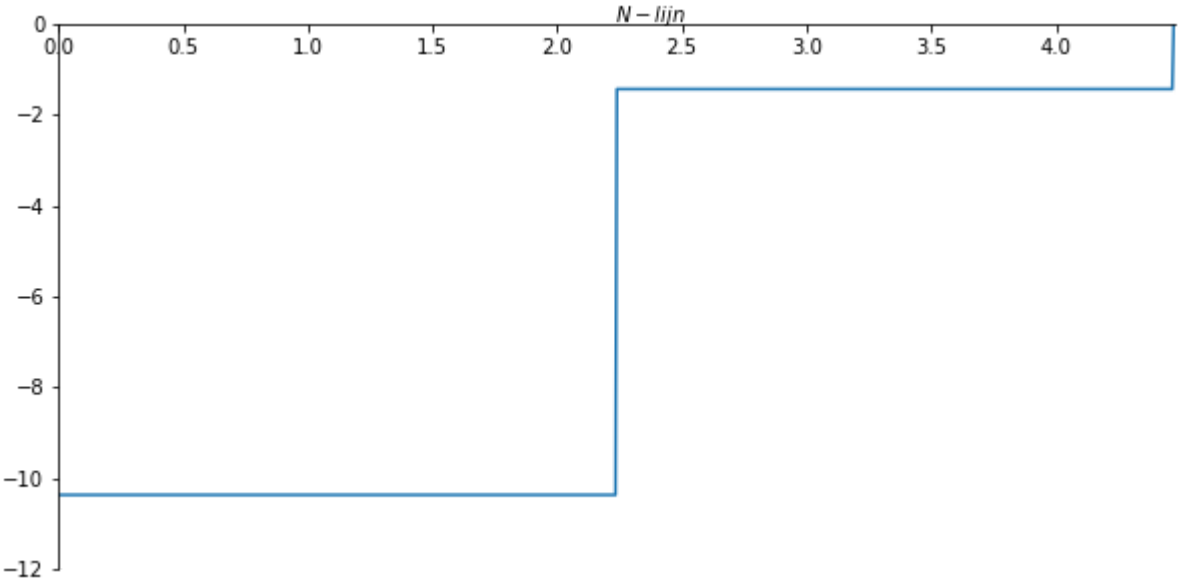

```
x_val = np.linspace(0, l, 901)
W_numpy = sp.lambdify(x,W.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
W_list = W_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(2, -10)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$EI * w-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(W_list) , label='w-lijn')
plt.savefig('w_lijnvoorbeeld3', dpi=450);
```



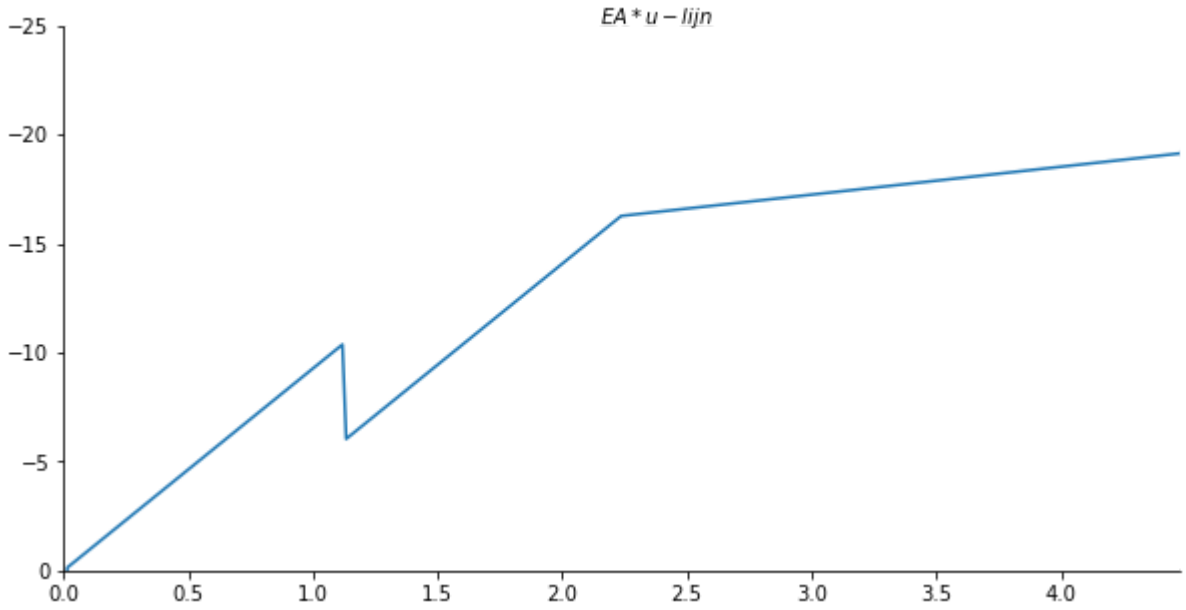
```
x_val = np.linspace(0, 4.47213595499958, 901)
N_numpy = sp.lambdify(x,N.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
N_list = N_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(-12, 0)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.set_xlabel("$N-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(N_list) , label='N-lijn')
plt.savefig('N_lijnvoorbeeld3', dpi=450);
```




```
x_val = np.linspace(0, 1, 301)
u_numpy = sp.lambdify(x,u.subs(solutions).rewrite(sp.Piecewise).simplify()) #substitute f
u_list = u_numpy(x_val)
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.48)
ax.set_ylim(0, -25)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)

ax.set_xlabel("$EA*u-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/sp.cos(theta),  np.array(u_list) , label='N-lijn')
plt.savefig('u_lijnvoorbeeld3', dpi=450);
```



Voorbeeld 4 en 5

Contents


```
import sympy as sp
import numpy as np
from sympy import symbols
sf = sp.SingularityFunction
import matplotlib.pyplot as plt
```

kromme ligger voorbeeld 1

```
l = 4
x = np.linspace(0, l, 552)
dx = (x[1]-x[0])
print(dx)
```

```
0.007259528130671506
```

```
y = x**2/ 20
#print(y)
```

```
dy = []
for i in range(len(y)):
    dy.append(y[i] - y[i-1])
```

```
theta = np.arctan(dy/dx)
```

```
cos = np.cos(theta)[1:]
sin = np.sin(theta)[1:]
```

```

```

```
E, I = symbols('E, I')
x = symbols('x')
```

```
## Voorbeeld 1: opgelegd op twee steunpunten

Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, theta = sp.symbols('Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, t
# Define F and l
F = 10  ## KN

# Define qz and qx
qz = -Av * sf(x, 0, -1) + F * sf(x, l/2, -1) - Bv * sf(x, l, -1)
qx = +Ah * sf(x, 0, -1)
```

```
# Define V as a function of x
V = sp.sin(theta) * sp.integrate(-qx, x) + sp.cos(theta) * sp.integrate(-qz, x) + Cv
# N definiëren als een functie van x
N = sp.cos(theta) * sp.integrate(-qx, x) + sp.sin(theta) * sp.integrate(qz, x) + Cn

# N definiëren als een functie van x
M = sp.integrate(V, x) / sp.cos(theta) + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw


# Display the expressions
display("V:", V)
display("M:", M)
display("phi:", phi)
display("W:", W)
display( N)
```

```
'V:'
```

$$-Ah\sin\left(\theta\right)\langle x\rangle^{0} + Cv + \left(Av\langle x\rangle^{0} + Bv\langle x - 4\rangle^{0} - 10\langle x - 2.0\rangle^{0}\right)\cos\left(\theta\right)$$

```
'M:'
```

$$Cm + \frac{-Ah\sin\left(\theta\right)\langle x\rangle^{1} + Cvx + \left(Av\langle x\rangle^{1} + Bv\langle x - 4\rangle^{1} - 10\langle x - 2.0\rangle^{1}\right)\cos\left(\theta\right)}{\cos\left(\theta\right)}$$

```
'phi:'
```

$$Cmx + Cphi + \frac{-\frac{Ah\sin(\theta)\langle x\rangle^2}{2} + \frac{Av\cos(\theta)\langle x\rangle^2}{2} + \frac{Bv\cos(\theta)\langle x-4\rangle^2}{2} + \frac{Cvx^2}{2} - 5\cos(\theta)\langle x -}{\cos(\theta)}$$

```
'W:'
```

$$-\frac{Cmx^2}{2} - Cphix + Cw - \frac{-\frac{Ah\sin(\theta)\langle x\rangle^3}{6} + \frac{Av\cos(\theta)\langle x\rangle^3}{6} + \frac{Bv\cos(\theta)\langle x-4\rangle^3}{6} + \frac{Cvx^3}{6} - \frac{5}{}}{\cos(\theta)}$$

$$-Ah\cos(\theta)\langle x\rangle^0 + Cn + \left(-Av\langle x\rangle^0 - Bv\langle x-4\rangle^0 + 10\langle x-2.0\rangle^0\right)\sin(\theta)$$

```
eq1 = V.subs(x, -1)
eq2 = V.subs(x, l+1)

eq3 = M.subs(x, 0)
eq4 = M.subs(x, l)

eq5 = W.subs(x, l)
eq6 = W.subs(x, 0)
eq7 = N.subs(x, -1)
eq8 = N.subs(x, l+1)
equations = [eq1 -0, eq2-0,eq3-0,eq4-0,eq5-0,eq6-0, eq7-0, eq8 -0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn))
```

```
print(solutions)
```

```
{Cv: 0.0, Cm: 0.0, Cphi: -10.0000000000000, Cw: 0.0, Av: 5.00000000000000, Bv: 5.00000000
```

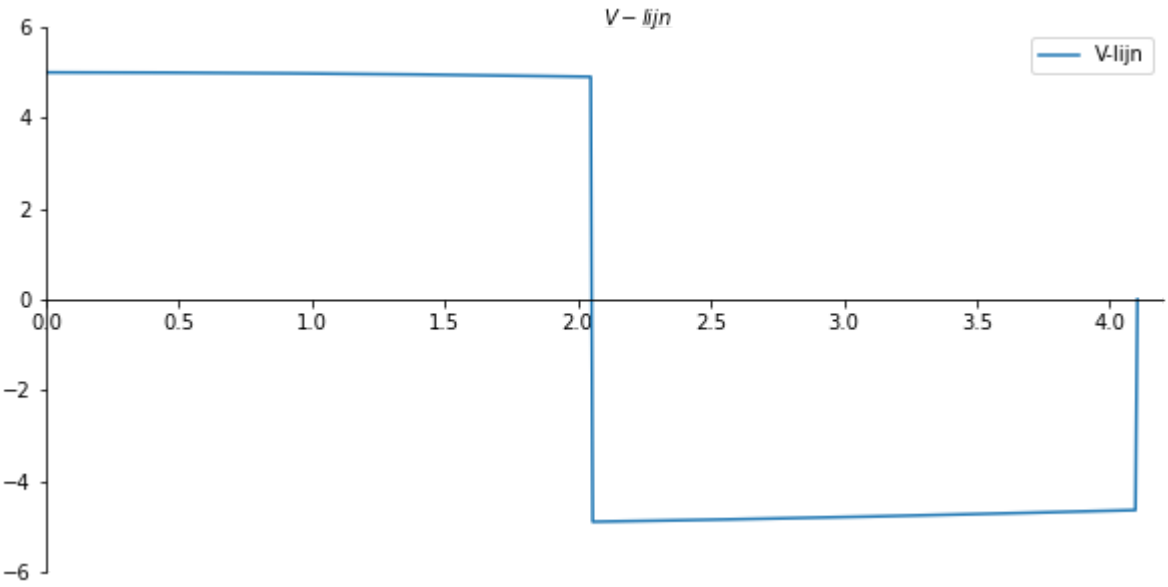
```
Cv= 0.0
Cm= 0.0
Cphi= -10
Cw= 0.0,
Av= 5.00000000000000
Bv= 5.00000000000000
Ah= 0.0
Cn= 0.0
```

```
display(V)
```

$$-Ah\sin\left(\theta\right)\langle x\rangle^{0}+Cv+\left(Av\langle x\rangle^{0}+Bv\langle x-4\rangle^{0}-10\langle x-2.0\rangle^{0}\right)\cos\left(\theta\right)$$

```
x_val = np.linspace(0, l, 550)
V_list = []
for i in range(len(x_val)):
    V_list.append((Av*sf(x_val[i], 0, 0)+ Bv*sf(x_val[i], 4, 0) - 10*sf(x_val[i], 2, 0))
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-6, 6)
ax.set_xlim(0, 4.2)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$V-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[316], np.array(V_list), label='V-lijn')
ax.legend();
x_val[-1]/cos[316] #De lengte van de kromme ligger.
plt.savefig('V_par_lijnvoorbeeld1', dpi=450);
```

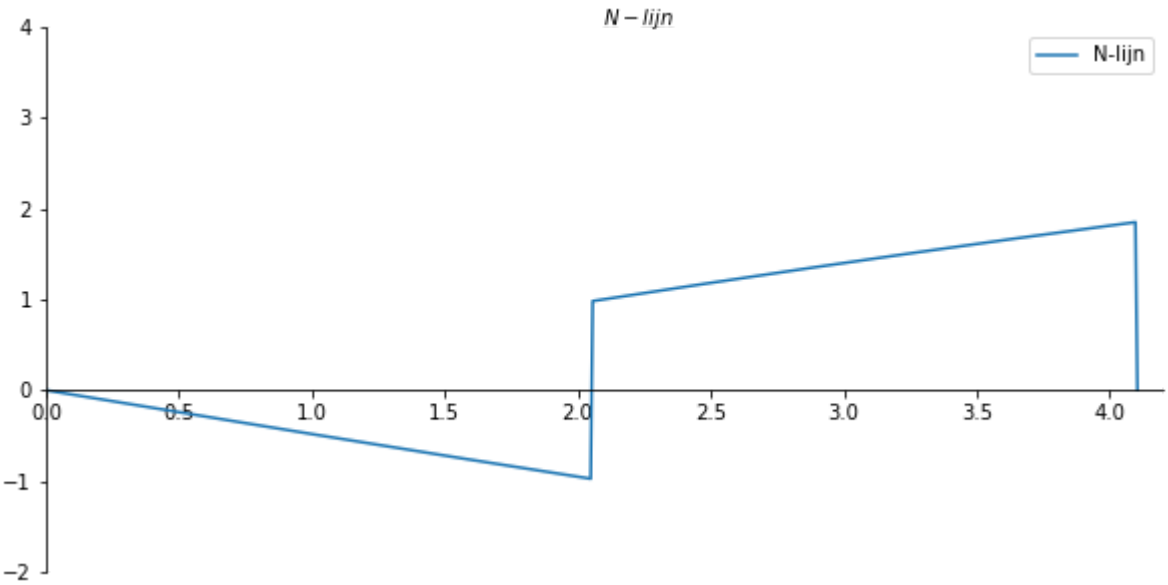



```
display(N)
```

$$-Ah\cos\left(\theta\right)\langle x\rangle^{0}+Cn+\left(-Av\langle x\rangle^{0}-Bv\langle x-4\rangle^{0}+10\langle x-2.0\rangle^{0}\right)\sin\left(\theta\right)$$

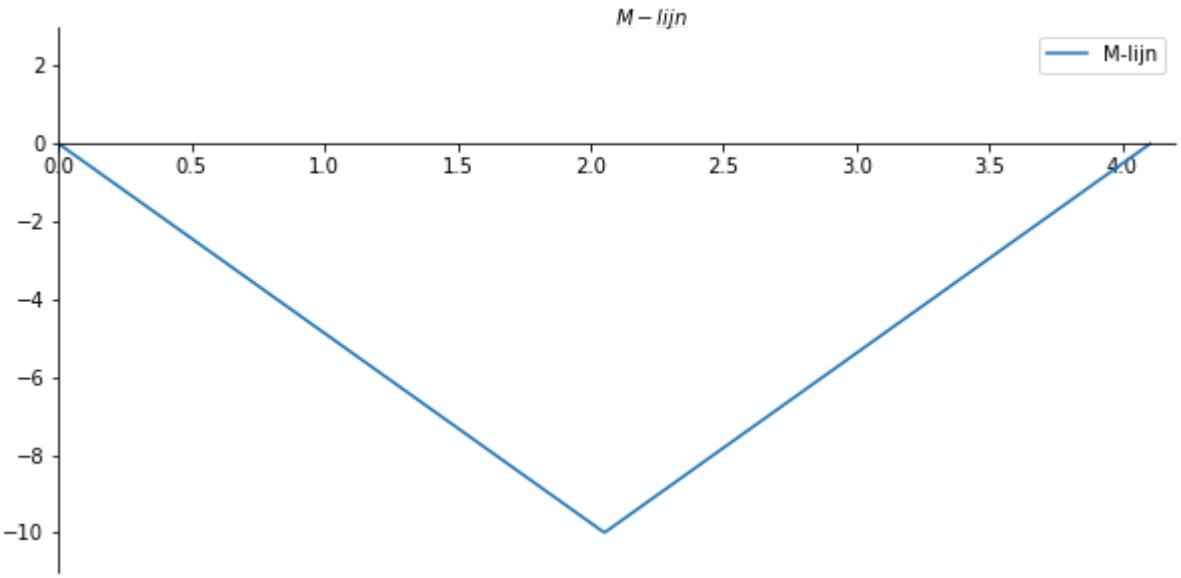
```
x_val = np.linspace(0, l, 550)
N_list = []
for i in range(len(x_val)):
    N_list.append((-Av*sf(x_val[i], 0,0) -Bv*sf(x_val[i],4,0)+ 10*sf(x_val[i], 2, 0)) * s
#print(N_list)
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-2, 4)
ax.set_xlim(0, 4.2)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$N-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[316], np.array(N_list), label='N-lijn')
ax.legend()
plt.savefig('N_par_lijnvoorbeeld1', dpi=450);
```



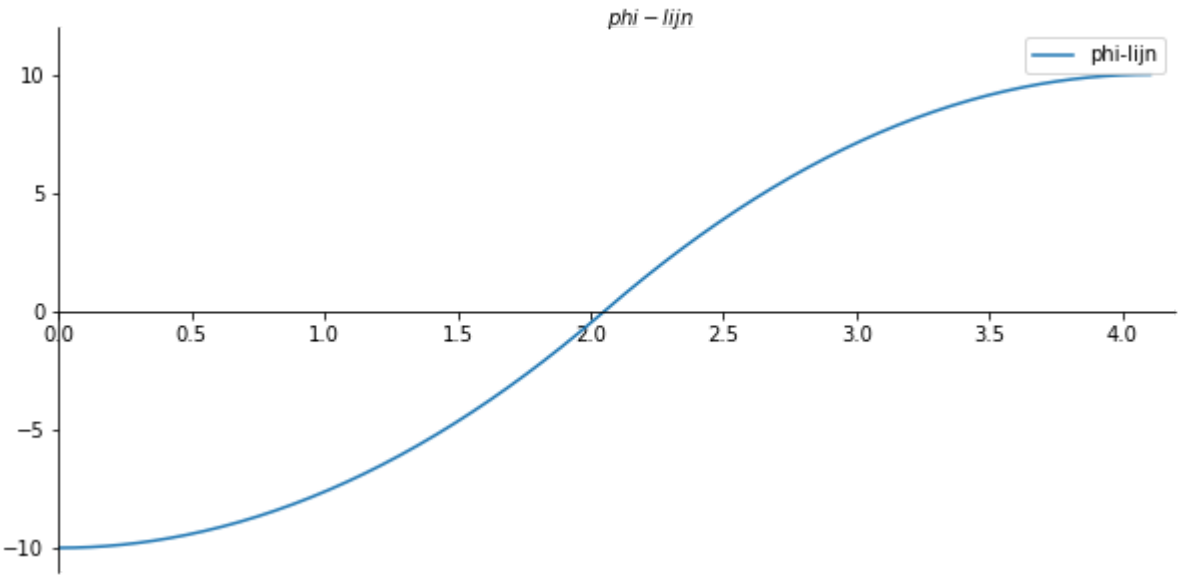

```
x_val = np.linspace(0, 4, 550)
M_list = []
for i in x_val:
    M_list.append((Av * sf(i, 0, 1) - F * sf(i, l/2, 1) + Bv * sf(i, l, 1)))
#display(M)
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.2)
ax.set_ylim(-11, 3)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[316], -np.array(M_list) , label='M-lijn')
ax.legend()
plt.savefig('M_par_lijnvoorbeeld1', dpi=450);
```



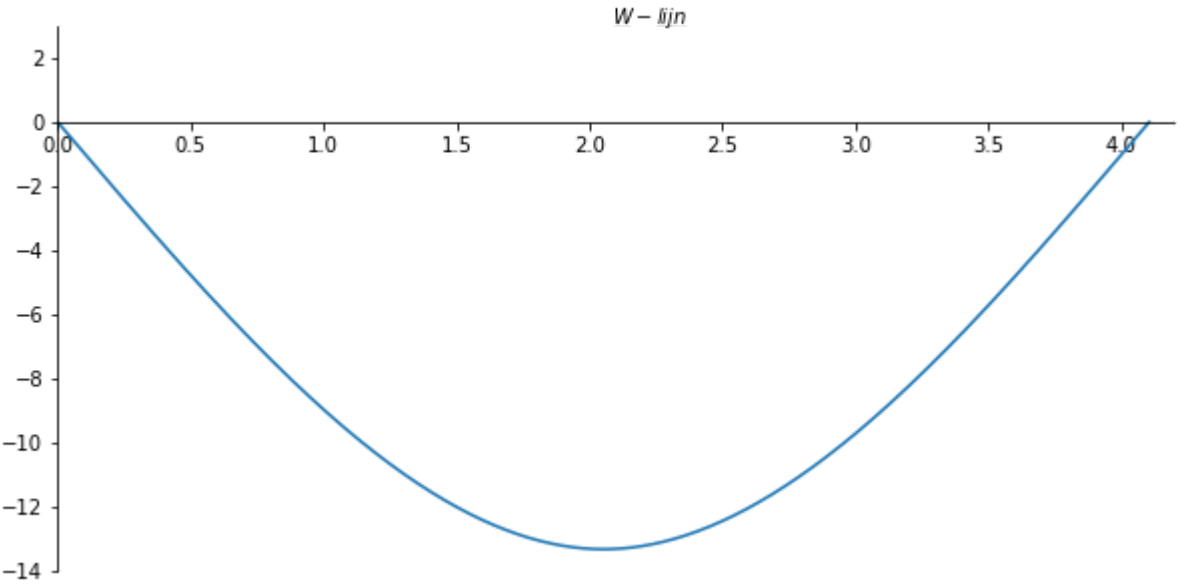

```
x_val = np.linspace(0, l, 550)
phi_list = []
for i in x_val:
    phi_list.append((Av/2) * sf(i, 0, 2) - (F/2) * sf(i, l/2, 2) + (Bv/2) * sf(i, l, 2) +
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.2)
ax.set_ylim(-11, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$phi-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[316], np.array(phi_list), label='phi-lijn')
ax.legend()
plt.savefig('phi_par_lijnvoorbeeld1', dpi=450);
```




```
x_val = np.linspace(0, l, 550)
W_list = []
for i in x_val:
    W_list.append((-Av/6) * sf(i, 0, 3) + (F/6) * sf(i, l/2, 3) + (Bv/6) * sf(i, l, 3) -
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 4.2)
ax.set_ylim(-14, 3)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$W-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[316], - np.array(W_list), label='W-lijn')
plt.savefig('W_par_lijnvoorbeeld1', dpi=450);
```



Kromme ligger voorbeeld 2

```
l = 4
x = np.linspace(0, l, 552)
dx = (x[1]-x[0])
print(dx)
```

```
0.007259528130671506
```

```
y = x**2/ 4
#print(y)
```

```
dy = []
for i in range(len(y)):
    dy.append(y[i] - y[i-1])
```

```
theta = np.arctan(dy/dx)
```

```
cos = np.cos(theta)[1:]
sin = np.sin(theta)[1:]
```

```
E, I = symbols('E, I')
x = symbols('x')
```

```
## Voorbeeld 1: opgelegd op twee steunpunten

Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, theta = sp.symbols('Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn, t
# Define F and l
F = 10  ## KN

# Define qz and qx
qz = -Av * sf(x, 0, -1) + F * sf(x, l/2, -1) - Bv * sf(x, l, -1)
qx = +Ah * sf(x, 0, -1)
```

```
# Define V as a function of x
V = sp.sin(theta) * sp.integrate(-qx, x) + sp.cos(theta) * sp.integrate(-qz, x) + Cv
# N definiëren als een functie van x
N = sp.cos(theta) * sp.integrate(-qx, x) + sp.sin(theta) * sp.integrate(qz, x) + Cn

# N definiëren als een functie van x
M = sp.integrate(V, x) / sp.cos(theta) + Cm

# Define phi as an integral of M
phi = sp.integrate(M, x) + Cphi

# Define W as an integral of -phi
W = sp.integrate(-phi, x) + Cw


# Display the expressions
display("V:", V)
display("M:", M)
display("phi:", phi)
display("W:", W)
display( N)
```

```
'V:'
```

$$-Ah\sin\left(\theta\right)\langle x\rangle^{0} + Cv + \left(Av\langle x\rangle^{0} + Bv\langle x - 4\rangle^{0} - 10\langle x - 2.0\rangle^{0}\right)\cos\left(\theta\right)$$

```
'M:'
```

$$Cm + \frac{-Ah\sin{(\theta)}\langle x\rangle^{1} + Cvx + \left(Av\langle x\rangle^{1} + Bv\langle x-4\rangle^{1} - 10\langle x-2.0\rangle^{1}\right)\cos{(\theta)}}{\cos{(\theta)}}$$

```
'phi:'
```

$$Cmx + Cphi + \frac{-\frac{Ah\sin{(\theta)}\langle x\rangle^{2}}{2} + \frac{Av\cos{(\theta)}\langle x\rangle^{2}}{2} + \frac{Bv\cos{(\theta)}\langle x-4\rangle^{2}}{2} + \frac{Cvx^{2}}{2} - 5\cos{(\theta)}\langle x -}{\cos{(\theta)}}$$

```
'W:'
```

$$-\frac{Cmx^{2}}{2} - Cphix + Cw - \frac{-\frac{Ah\sin{(\theta)}\langle x\rangle^{3}}{6} + \frac{Av\cos{(\theta)}\langle x\rangle^{3}}{6} + \frac{Bv\cos{(\theta)}\langle x-4\rangle^{3}}{6} + \frac{Cvx^{3}}{6} - \frac{5}{\cos{(\theta)}}$$

$$-Ah\cos{(\theta)}\langle x\rangle^{0} + Cn + \left(-Av\langle x\rangle^{0} - Bv\langle x-4\rangle^{0} + 10\langle x-2.0\rangle^{0}\right)\sin{(\theta)}$$

```
eq1 = V.subs(x, -1)
eq2 = V.subs(x, l+1)

eq3 = M.subs(x, 0)
eq4 = M.subs(x, l)

eq5 = W.subs(x, l)
eq6 = W.subs(x, 0)
eq7 = N.subs(x, -1)
eq8 = N.subs(x, l+1)
equations = [eq1 -0, eq2-0,eq3-0,eq4-0,eq5-0,eq6-0, eq7-0, eq8 -0]
solutions = sp.solve(equations, (Cv, Cm, Cphi, Cw, Av, Bv, Ah, Cn))
```

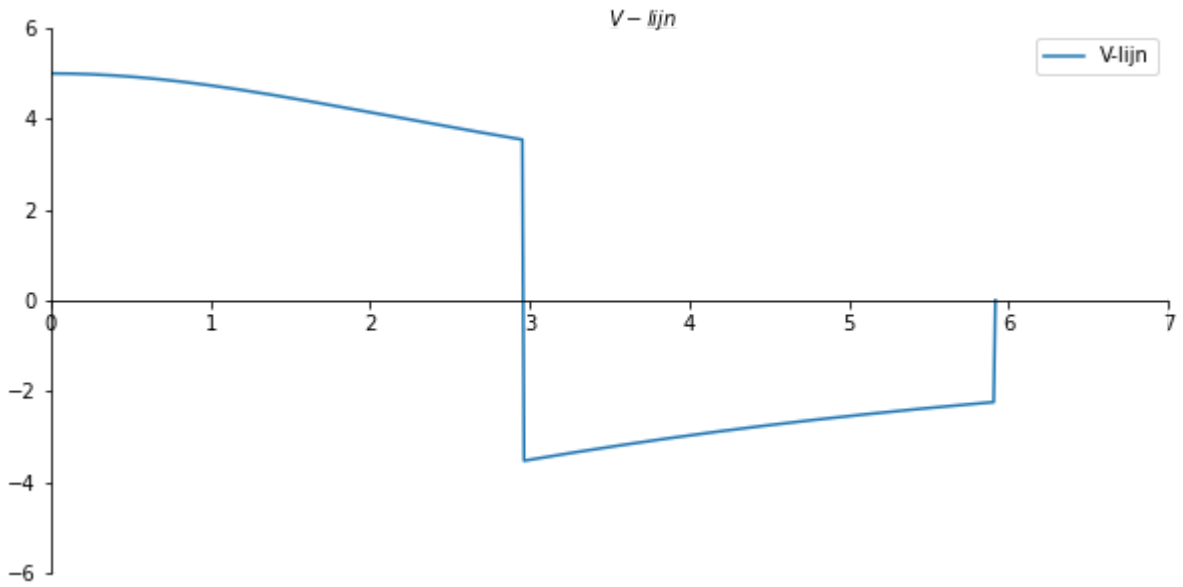
```
print(solutions)
```

```
{Cv: 0.0, Cm: 0.0, Cphi: -10.0000000000000, Cw: 0.0, Av: 5.00000000000000, Bv: 5.00000000
```

```
Cv= 0.0
Cm= 0.0
Cphi= -10
Cw= 0.0,
Av= 5.00000000000000
Bv= 5.00000000000000
Ah= 0.0
Cn= 0.0
```

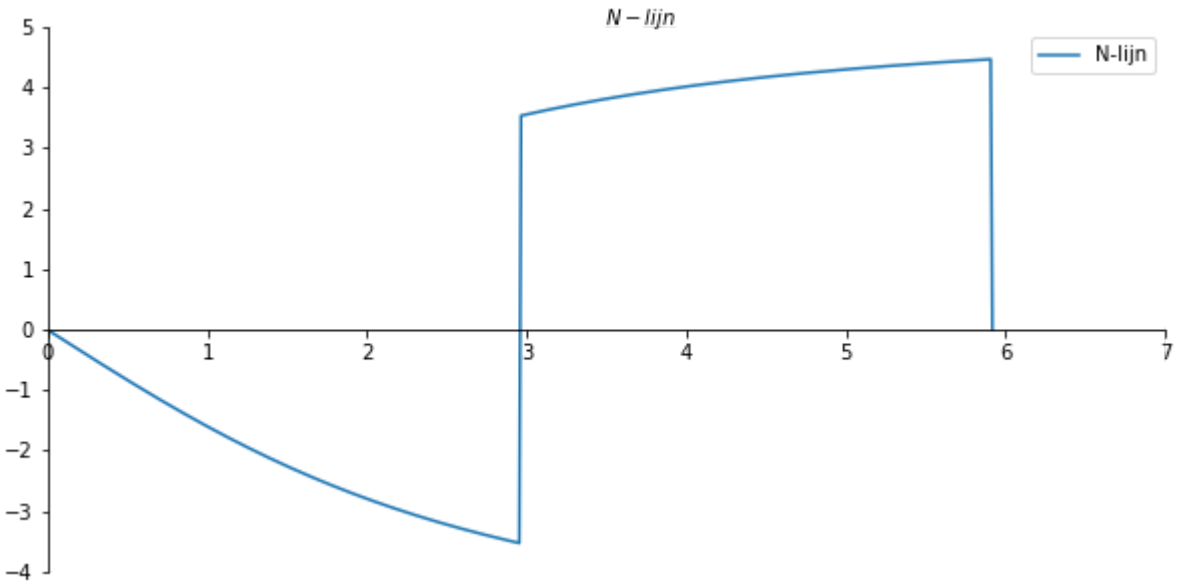
```
x_val = np.linspace(0, l, 550)
V_list = []
for i in range(len(x_val)):
    V_list.append((Av*sf(x_val[i], 0, 0)+ Bv*sf(x_val[i], 4, 0) - 10*sf(x_val[i], 2, 0))
#print(V_list)
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-6, 6)
ax.set_xlim(0, 7)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$V-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[300], np.array(V_list), label='V-lijn')
ax.legend();
x_val[-1]/cos[300] #De lengte van de kromme ligger.
plt.savefig('V_par_lijnvoorbeeld2', dpi=450);
```



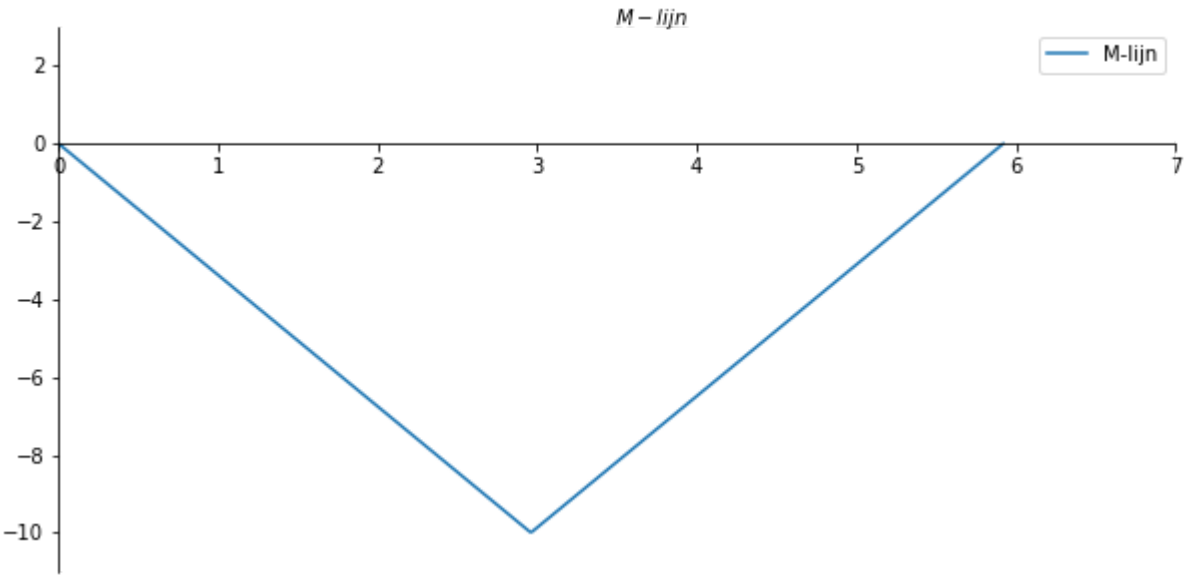
```
x_val = np.linspace(0, l, 550)
N_list = []
for i in range(len(x_val)):
    N_list.append((-Av*sf(x_val[i], 0,0) -Bv*sf(x_val[i],4,0)+ 10*sf(x_val[i], 2, 0)) * s
# print(N_list)
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-4, 5)
ax.set_xlim(0, 7)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$N-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[300], np.array(N_list), label='N-lijn')
ax.legend()
plt.savefig('N_par_lijnvoorbeeld2', dpi=450);
```



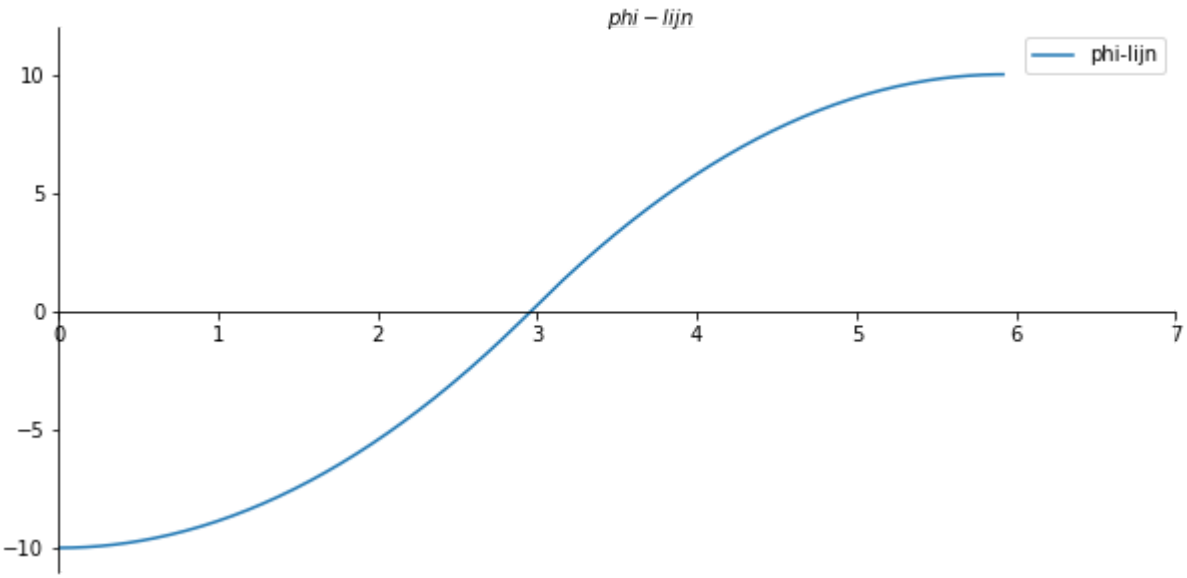

```
x_val = np.linspace(0, 4, 550)
M_list = []
for i in x_val:
    M_list.append((Av * sf(i, 0, 1) - F * sf(i, l/2, 1) + Bv * sf(i, l, 1)))
#display(M)
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 7)
ax.set_ylim(-11, 3)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$M-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[300], -np.array(M_list) , label='M-lijn')
ax.legend()
plt.savefig('M_par_lijnvoorbeeld2', dpi=450);
```



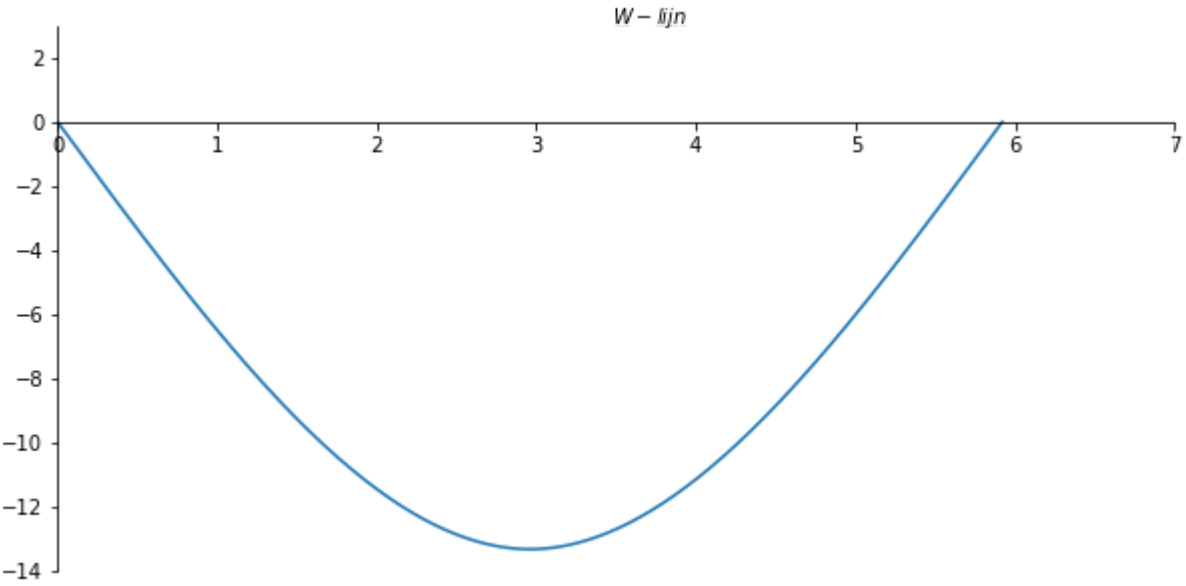

```
x_val = np.linspace(0, l, 550)
phi_list = []
for i in x_val:
    phi_list.append((Av/2) * sf(i, 0, 2) - (F/2) * sf(i, l/2, 2) + (Bv/2) * sf(i, l, 2) +
```

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 7)
ax.set_ylim(-11, 12)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$phi-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[300], np.array(phi_list), label='phi-lijn')
ax.legend()
plt.savefig('phi_par_lijnvoorbeeld2', dpi=450);
```




```
x_val = np.linspace(0, l, 550)
W_list = []
for i in x_val:
    W_list.append((-Av/6) * sf(i, 0, 3) + (F/6) * sf(i, l/2, 3) + (Bv/6) * sf(i, l, 3) -
```

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_xlim(0, 7)
ax.set_ylim(-14, 3)

ax.spines["left"].set_position("zero")
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_position("zero")
ax.spines["top"].set_visible(False)


ax.set_xlabel("$W-lijn$")
ax.xaxis.set_label_coords(0.53, 1.04)
plt.gca()
ax.plot(x_val/cos[300], - np.array(W_list), label='W-lijn')
plt.savefig('W_par_lijnvoorbeeld2', dpi=450);
```




```
x = np.linspace(0, 4, 21)
print(x)
```

```
[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2.  2.2 2.4 2.6 2.8 3.  3.2 3.4
 3.6 3.8 4. ]
```

```
y = x**2/20
y1 = x**2/ 4
```

```
print(y)
```

```
[0.    0.002 0.008 0.018 0.032 0.05  0.072 0.098 0.128 0.162 0.2   0.242
 0.288 0.338 0.392 0.45  0.512 0.578 0.648 0.722 0.8  ]
```

```
print(y1)
```

```
[0.   0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.   1.21 1.44 1.69
 1.96 2.25 2.56 2.89 3.24 3.61 4.  ]
```

Alex: Twodimensional structures

Contents

> In dit rapport wordt uitgelegd hoe de Macaulay methode kan worden uitgebreid zodat het toepasbaar is op geknikte, vertakte en gesloten constructies, inclusief gescharnierde. Het model is robuust, systematisch toepasbaar, kan de snedekrachten en globale verplaatsingen op elk punt berekenen en de oplossingsvoorwaarden zijn eenvoudiger en geringer in aantal in vergelijking met de directe integratie methode.
>
> Het model maakt gebruik van een globaal en een lokaal assenstelsel ((h, v) en (x, z)). In figuur 1 wordt het verloop van het lokaal assenstelsel aangegeven met de nummering van de blauwe pijlen. De belastingen en de verplaatsingen worden globaal genoteerd en de snedekrachtdiagrammen lokaal. Er zijn twee vergelijkingen die de lokale as volgen. $q_z(x)$ is voor de krachten loodrecht op de staaf en integreert naar de dwarskrachtenvergelijking, enz. en $q_x(x)$ is voor de krachten parallel aan de staaf en integreert naar de normaalkrachtenvergelijking, enz.
>
> De belastingen worden in 2D anders gemodelleerd dan in 1D. Er wordt in het model onderscheid gemaakt tussen een koppel (T), verticale en horizontale puntlast (F_v en F_h), verticale en horizontale gelijkmatig verdeelde belasting (q_v en q_h) en scharnier ($EI\varphi_s$). De vergelijking voor elke belasting is anders voor $q_z(x)$ en $q_x(x)$ en bestaan telkens uit een beginterm (b) voor het aangrijppunt van de belasting en een systematische reeks hoektermen (a_i) voor elke hoek (θ_i) waar de functie van x langs gaat na het aangrijppunt van de belasting. De vergelijkingen zijn als volgt:
>
> **koppel**
>
> $$q_z(x) = T\langle x - b\rangle^{-2}$$

$$q_x(x) = 0$$

Verticale Puntlast

$$q_z(x) = F_v \left(\langle x - b \rangle^{-1} \cos(\theta_b) + \sum \left(\langle x - a_i \rangle^{-1} (\cos(\theta_i) - \cos(\theta_{i-1})) \right) \right) ; \quad v$$

$$q_x(x) = -F_v \left(\langle x - b \rangle^{-1} \sin(\theta_b) + \sum \left(\langle x - a_i \rangle^{-1} (\sin(\theta_i) - \sin(\theta_{i-1})) \right) \right) ;$$

Horizontale puntlast

$$q_z(x) = F_h \left(\langle x - b \rangle^{-1} \sin(\theta_b) + \sum \left(\langle x - a_i \rangle^{-1} (\sin(\theta_i) - \sin(\theta_{i-1})) \right) \right) ; \quad vc$$

$$q_x(x) = F_h \left(\langle x - b \rangle^{-1} \cos(\theta_b) + \sum \left(\langle x - a_i \rangle^{-1} (\cos(\theta_i) - \cos(\theta_{i-1})) \right) \right) ; \quad v$$

Verticale Gelijkmatig Verdeelde Belasting

$$q_z(x) = q_v \left(\begin{array}{c} \langle x - b \rangle^{0} \cos(\theta_b) + \\ \sum \left(\left(\left(\langle x - a_i \rangle^{0} + \langle x - a_i \rangle^{-1} (a_i - b) \right) (\cos(\theta_i) - \cos(\theta_{i-1})) \right) \right) \end{array} \right)$$

$$q_x(x) = -q_v \left(\begin{array}{c} \langle x - b \rangle^{0} \sin(\theta_b) + \\ \sum \left(\left(\left(\langle x - a_i \rangle^{0} + \langle x - a_i \rangle^{-1} (a_i - b) \right) (\sin(\theta_i) - \sin(\theta_{i-1})) \right) \right) \end{array} \right)$$

Horizontaal Gelijkmatig Verdeelde Belasting

$$q_z(x) = q_h \left(\begin{array}{c} \langle x-b \rangle^0 \sin(\theta_b) + \\ \sum \left(\left(\left(\langle x-a_i \rangle^0 + \langle x-a_i \rangle^{-1} (a_i - b) \right) (\sin(\theta_i) - \sin(\theta_{i-1})) \right) \right) \end{array} \right.$$

$$q_x(x) = q_h \left(\begin{array}{c} \langle x-b \rangle^0 \cos(\theta_b) + \\ \sum \left(\left(\left(\langle x-a_i \rangle^0 + \langle x-a_i \rangle^{-1} (a_i - b) \right) (\cos(\theta_i) - \cos(\theta_{i-1})) \right) \right) \end{array} \right)$$

Scharnier

$$q_z(x) = EI\varphi_s \langle x-b \rangle^{-3}$$

$$q_x(x) = 0$$

De doorbuiging ($u_z(x)$) en extensie ($u_x(x)$) zijn voor tweedimensionale constructies hele abstracte termen die niet bruikbaar zijn voor de oplossingsvoorwaarden. Daarom moeten ze worden gebruikt als invoer voor de vergelijkingen van de horizontale en verticale verplaatsing van de constructie ($u_h(x)$ en $u_v(x)$). Deze vergelijkingen zijn als volgt:

$$u_v(x) = u_z(a_0)\cos(\theta_0) - u_x(a_0)\sin(\theta_0) + \sum_{i=0}^{i=n} \left(\begin{array}{c} \left((u_z(x) - u_z(a_i)) \langle x-a_i \rangle^{(} \right. \\ - \left((u_x(x) - u_x(a_i)) \langle x-a_i \right. \end{array} \right.$$

$$u_h(x) = u_z(a_0)\sin(\theta_0) + u_x(a_0)\cos(\theta_0) + \sum_{i=0}^{i=n} \left(\begin{array}{c} \left((u_z(x) - u_z(a_i)) \langle x-a_i \rangle \right. \\ + \left((u_x(x) - u_x(a_i)) \langle x-a_i \right. \end{array} \right.$$

Hierbij is a_i het beginpunt, a_j het eindpunt en θ_i de hoek van staaf ij.

Voor vertakte en gesloten constructies bestaan er drie bijzondere punten met unieke eigenschappen. Dit zijn het knooppunt (a_k), sprongpunt (a_s) en aansluitpunt (a_a).

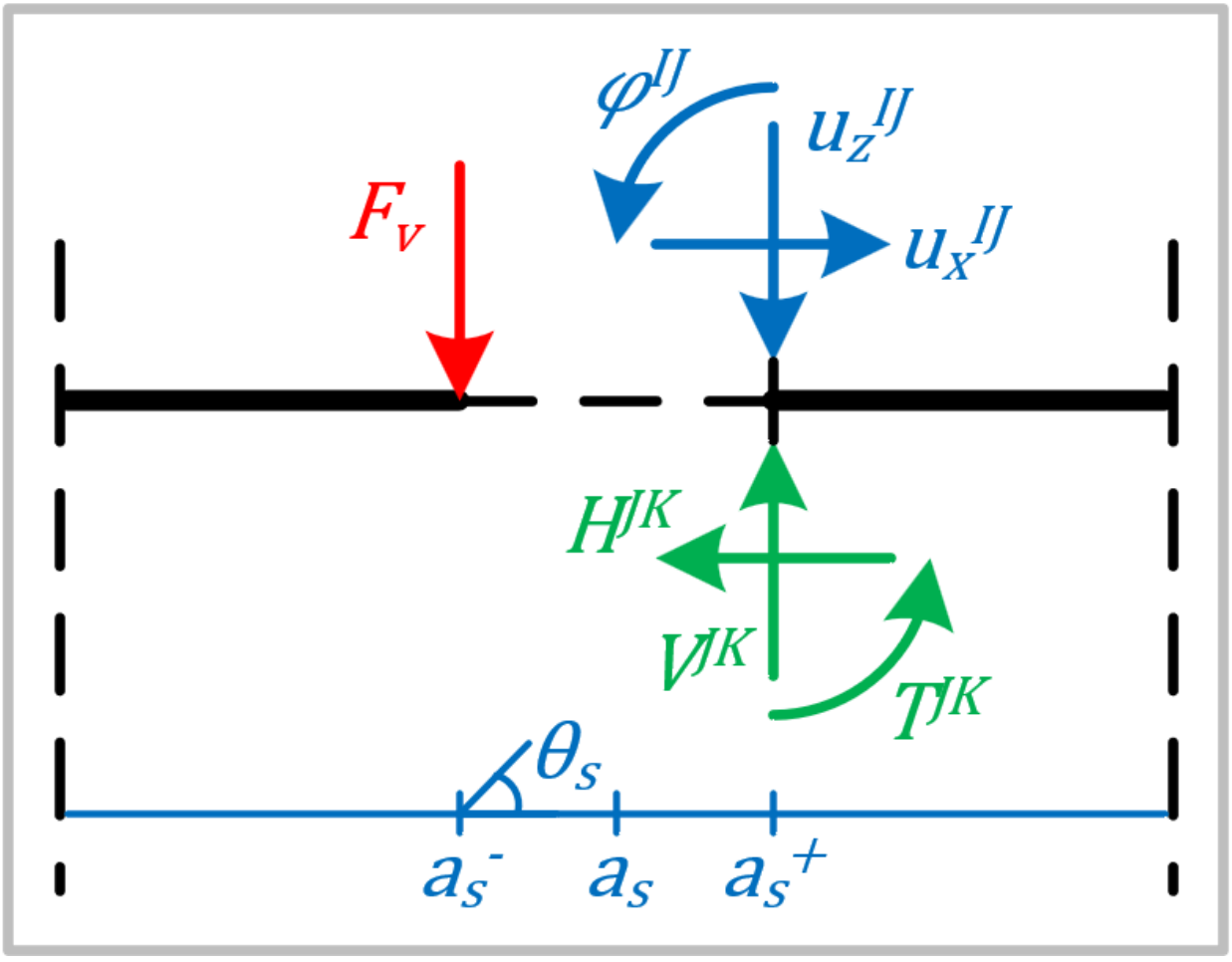
Het knooppunt is het punt waarbij de functie van x voor het eerst langs een knoop gaat. Op a_k moet voor elke staaf IJ waarnaar de functie niet direct de weg vervolgd drie onbekende krachten worden toegevoegd V^{IJ}, H^{IJ} en T^{IJ}. Hierin

vertegenwoordigd V^{IJ} de verticale kracht, H^{IJ} de horizontale kracht en T^{IJ} de koppel voortkomend uit staaf IJ werkend op knoop I.

Het sprongpunt, (zie figuur 1) is het punt waar de functie van x in het globale assenstelsel een sprong maakt van punt I, wat een uiteinde of afsluiting is van een vertakking, naar punt J, wat een knoopverbinding is waar de functie van x al eerder langs is geweest. Belastingen werkende op punt I moeten gemodelleerd worden op a_s^- en op punt J op a_s^+. Op a_s^+ moeten ook de onbekende krachten $-V^{JK}$, $-H^{JK}$ en $-T^{JK}$ worden toegevoegd voor staaf JK waarnaar de functie na de sprong de weg vervolgd. Verder moeten op a_s^+ de sprongconstanten φ^{IJ}, u_z^{IJ} en u_x^{IJ} toegevoegd worden voor de sprong in vervormingen tussen punt I en J. Deze worden als volgt gemodelleerd:

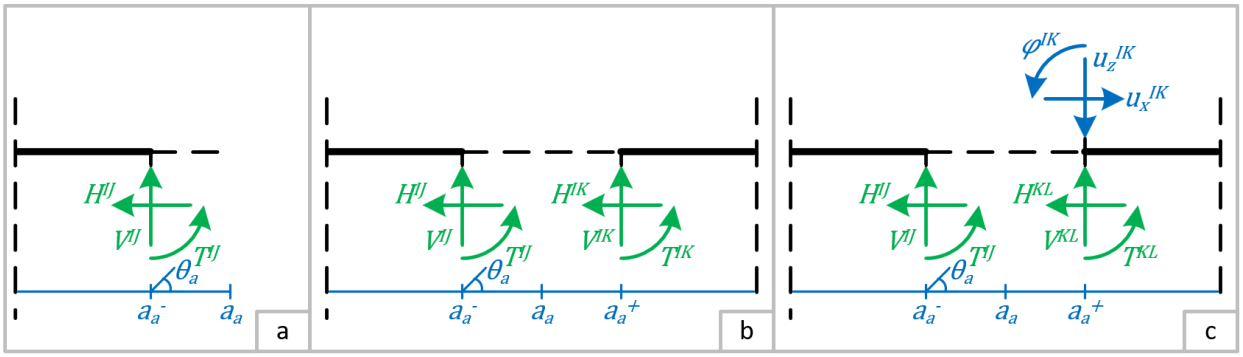
$$q_z(x) = EI\varphi^{ij}\left\langle x - a_s^+ \right\rangle^{-3} + EIu_z^{ij}\left\langle x - a_s^+ \right\rangle^{-4}$$

$$q_x(x) = EAu_x^{ij}\left\langle x - a_s^+ \right\rangle^{-2}$$



figuur 1: modelleren van een sprongpunt.

Het aansluitpunt is het punt waarbij de functie van x opnieuw langs een knoop gaat. Op a_a^- moeten de drie onbekende krachten $-V^{IJ}$, $-H^{IJ}$ en $-T^{IJ}$ voor de staaf IJ dat aansluit op punt I. Het aansluitpunt kan het einde zijn van de functie, maar de functie kan ook direct de weg vervolgen of een sprong maken. Het verschil in modelleren is te zien in figuur 2.



figuur 2: Modelleren van een aansluitpunt als het ook het eindpunt is (a), als de functie erna direct vervolgd (b) of het als het ook een sprongpunt is (c).

In Tabel 1 wordt voorgeschreven welke oplossingsvoorwaarden gelden voor starre tweedimensionale constructies als lokale afstand a_i de benoemde eigenschap heeft. Afstand a_i kan meerdere eigenschappen tegelijk hebben. Hierbij is a_k het knooppunt waarbij de functie voor het eerst langs dezelfde knoop komt als afstand a_i of a_i^+.

Tabel 1: Oplossingsvoorwaarden voor tweedimensionale starre constructies.

Situatie op a_i	**Oplossingsvoorwaarden**
Lokaal beginpunt	$N\left(a_i^-\right)=0$ $V\left(a_i^-\right)=0$ $M\left(a_i^-\right)=0$
Lokaal eindpunt	$N\left(a_i\right)=0$ $V\left(a_i\right)=0$ $M\left(a_i\right)=0$
Globaal eindpunt / sprongpunt (vertakte constructie)	Zie lokaal eindpunt + $\varphi\left(a_i^+\right)-\varphi\left(a_k\right)=0$ $u_v\left(a_i^+\right)-u_v\left(a_k\right)=0$ $u_h\left(a_i^+\right)-u_h\left(a_k\right)=0$
Aansluitpunt (gesloten constructie)	Zie lokaal eindpunt $\varphi\left(a_i\right)-\varphi\left(a_k\right)=0$ $u_v\left(a_i\right)-u_v\left(a_k\right)=0$ $u_h\left(a_i\right)-u_h\left(a_j\right)=0$
Scharnierverbinding	$M\left(a_i^+\right)=0$
Verticale roloplegging	$u_v\left(a_i\right)=0$
Horizontale roloplegging	$u_h\left(a_i\right)=0$
Scharnierende oplegging	$u_v\left(a_i\right)=0$ $u_h\left(a_i\right)=0$
Inklemming	$\varphi\left(a_i\right)=0$ $u_v\left(a_i\right)=0$ $u_h\left(a_i\right)=0$

Een staaf JK kan ook scharnierend verbonden zijn aan een knoop. Het modelleren ervan verschilt voor een knoop-, sprong en aansluitpunt. In Tabel 2 staat een overzicht van welke onbekenden en voorwaarden moeten worden toegevoegd en/of wegelaten voor elke situatie t.o.v. een star verbonden staaf. Als alle staven scharnieren verbonden zijn in de knoop, dan moet voor n staven $n-1$ staven een scharnierverbinding worden gemodelleerd in de knoop.

Baudoin (2024)

Documenten

- TU Delft repository
- GitHub repository , also shown in this book

Example 1

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 10**-15

# gegevens constructie
a0, a1, a2, a3 = 0, 4, 9, 14
aa = np.array([a0, a1, a2, a3])
o0, o1, o2 = 0, sym.atan(3/4), sym.atan(-4/3)
oo = np.array([o0, o1, o2])
L = 14

# gegevens belastingen
Fh, qv, Fv = 15, 6, 16
Rv, Rh, Tr = sym.symbols('R_v R_h T_R')
B = np.array([Fh, qv, -qv, Fv, Rv, Rh, Tr])
b1, b2, b3, b4 = 0, 2, 6.5, 14
bb = np.array([b1, b1, b3, b2, b4, b4, b4])
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5
nn = np.array([3, 4, 4, 2, 2, 3, 1])
```

```

```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(sym.symbols('{q_z}='), qz)
```

$$q_z =$$

$$-0.8R_h\langle x-14.0\rangle^{-1}+0.6R_v\langle x-14.0\rangle^{-1}+T_R\langle x-14.0\rangle^{-2}+6\langle x\rangle^{0}+16\langle x-2.0\rangle^{-}$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(sym.symbols('{q_x}='), qx)
```

$$q_x =$$

$$0.6R_h\langle x-14.0\rangle^{-1} + 0.8R_v\langle x-14.0\rangle^{-1} + 15\langle x\rangle^{-1} - 27.0\langle x-4\rangle^{-1} - 3.6\langle x-4\rangle^{0} +$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - 0.6R_h\langle x - 14.0\rangle^0 - 0.8R_v\langle x - 14.0\rangle^0 - 15\langle x\rangle^0 + 27.0\langle x - 4\rangle^0 + 3.6\langle x - 4\rangle^1 -$$

$$V =$$

$$C_V + 0.8R_h\langle x - 14.0\rangle^0 - 0.6R_v\langle x - 14.0\rangle^0 - T_R\langle x - 14.0\rangle^{-1} - 6\langle x\rangle^1 - 16\langle x - 2.0\rangle$$

$$M =$$

$$C_M + C_V x + 0.8R_h\langle x - 14.0\rangle^1 - 0.6R_v\langle x - 14.0\rangle^1 - T_R\langle x - 14.0\rangle^0 - 3\langle x\rangle^2 - 16\langle x$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - 0.6 R_h \langle x - 14.0 \rangle^{1} - 0.8 R_v \langle x - 14.0 \rangle^{1} - 15 \langle x \rangle^{1} + 27.0 \langle x - 4 \rangle^{1} + 1.8 \langle}{EA}$$

$$\phi =$$

$$C_{\phi} + \frac{C_M x + \frac{C_V x^2}{2} + 0.4 R_h \langle x - 14.0 \rangle^{2} - 0.3 R_v \langle x - 14.0 \rangle^{2} - T_R \langle x - 14.0 \rangle^{1} - \langle x \rangle^{3} -}$$

$$u_z =$$

$$-C_{\phi} x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} + 0.133333333333333 R_h \langle x - 14.0 \rangle^{3} - 0.1 R_v \langle x - 14.0}$$

$$u_v =$$

$$C_{uz} + \left(-C_{\phi} x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} + 0.133333333333333 R_h \langle x - 14.0 \rangle^{3} - 0.1 R_v \langle x - 14}$$

$$u_h =$$

$$C_{ux} - 0.2 \left(-\frac{4 C_N - 60.0}{EA} + \frac{C_N x - 0.6 R_h \langle x - 14.0 \rangle^{1} - 0.8 R_v \langle x - 14.0 \rangle^{1} - 15 \langle x \rangle^{1}}$$

```
# 3 reactiekrachten + 6 integratieconstanten = 9 voorwaarden
Eq1 = sym.Eq(N.subs(x,0-dx),0)
Eq2 = sym.Eq(N.subs(x,L+dx),0)
Eq3 = sym.Eq(V.subs(x,0-dx),0)
Eq4 = sym.Eq(V.subs(x,L+dx),0)
Eq5 = sym.Eq(M.subs(x,0),0)
Eq6 = sym.Eq(M.subs(x,L+dx),0)
Eq7 = sym.Eq(phi.subs(x,L),0)
Eq8 = sym.Eq(uv.subs(x,L),0)
Eq9 = sym.Eq(uh.subs(x,L),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9),(Rv,Rh,Tr,CN,CV,CM,Cphi,Cuz,Cux))
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: 3008.5/EI,
 C_ux: 0.0625*(56875.0*EA + 2304.0*EI)/(EA*EI),
 C_uz: 0.0833333333333333*(300259.0*EA + 3219.0*EI)/(EA*EI),
 R_h: -15.0000000000000,
 R_v: -55.0000000000000,
 T_R: -435.000000000000}
```

```
ea = 10**4
ei = 3 * 10**4

display(f'{Rv} = {Rv.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Rh} = {Rh.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Tr} = {Tr.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
```

```
'R_v = -55.00'
```

```
'R_h = -15.00'
```

```
'T_R = -435.00'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = 0.1003'
```

```
'C_ux = 0.1329'
```

```
'C_uz = 0.8609'
```

```
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
```

```
x_np = np.linspace(0-dx,L+dx,10000)
N_np = sym.lambdify(x, N.subs(sol).rewrite(sym.Piecewise))
display(N.subs(sol))
display(f'N(0) = {N.subs(x,0).subs(sol)}')
display(f'N(4+) = {N.subs(x,4).subs(sol)}')
display(f'N(9-) = {N.subs(x,9-dx).subs(sol)}')
display(f'N(9+) = {N.subs(x,9).subs(sol)}')

plt.figure()
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

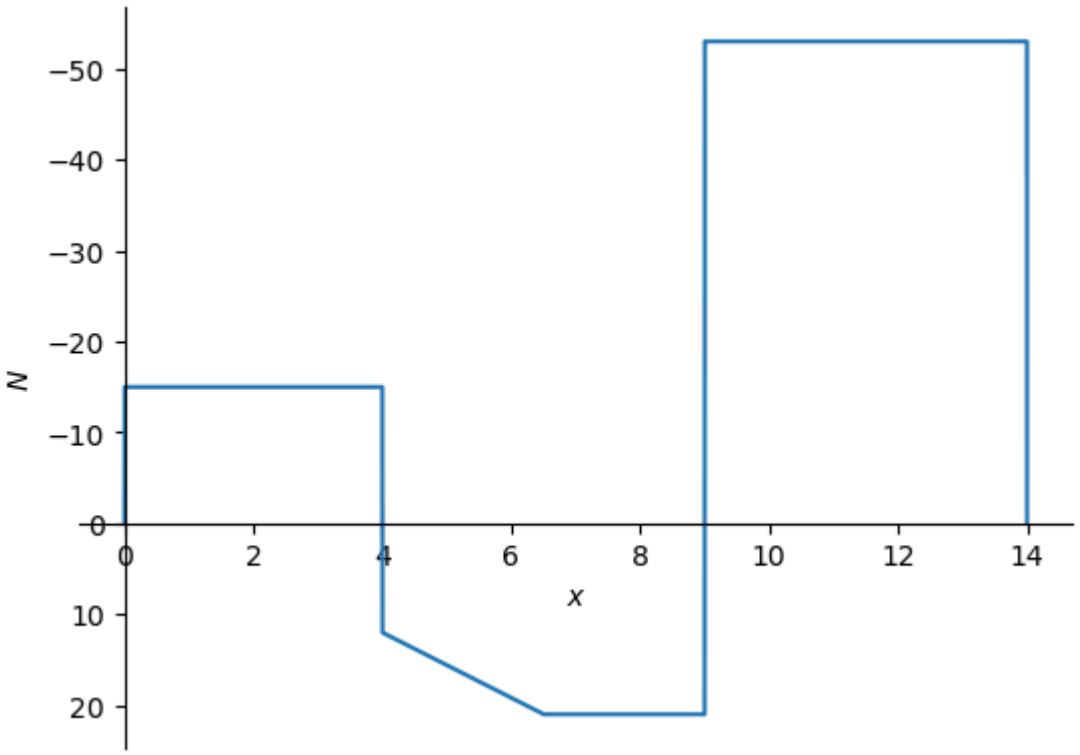
$$-15\langle x\rangle^{0} + 27.0\langle x-4\rangle^{0} + 3.6\langle x-4\rangle^{1} - 3.6\langle x-6.5\rangle^{1} - 74.0\langle x-9\rangle^{0} + 53.0\langle x-1$$

```
'N(0) = -15.0000000000000'
```

```
'N(4+) = 12.0000000000000'
```

```
'N(9-) = 21.0000000000000'
```

```
'N(9+) = -53.0000000000000'
```




```
V_np = sym.lambdify(x, V.subs(sol).rewrite(sym.Piecewise))
display(V.subs(sol))
display(f'V(2-) = {V.subs(x,b2-dx).subs(sol)}')
display(f'V(2+) = {V.subs(x,b2).subs(sol)}')
display(f'V(4-) = {V.subs(x,a1-dx).subs(sol)}')
display(f'V(4+) = {V.subs(x,a1).subs(sol)}')
display(f'V(9-) = {V.subs(x,a2-dx).subs(sol)}')
display(f'V(9+) = {V.subs(x,a2).subs(sol)}')

plt.figure()
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-6\langle x\rangle^{1} - 16\langle x-2.0\rangle^{0} - 1.0\langle x-4\rangle^{0} + 1.2\langle x-4\rangle^{1} + 4.8\langle x-6.5\rangle^{1} + 32.0\langle x-9\rangle^{0} -$$

```
'V(2-) = -12.0000000000000'
```

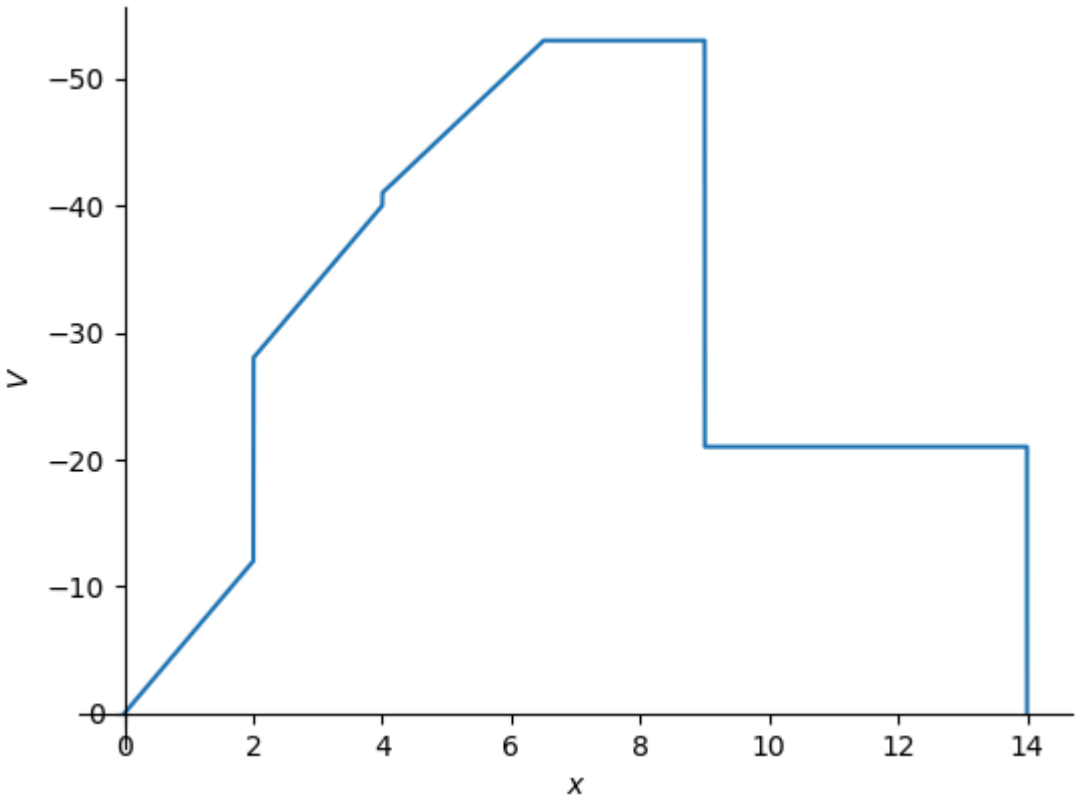
```
'V(2+) = -28.0000000000000'
```

```
'V(4-) = -40.0000000000000'
```

```
'V(4+) = -41.0000000000000'
```

```
'V(9-) = -53.0000000000000'
```

```
'V(9+) = -21.0000000000000'
```



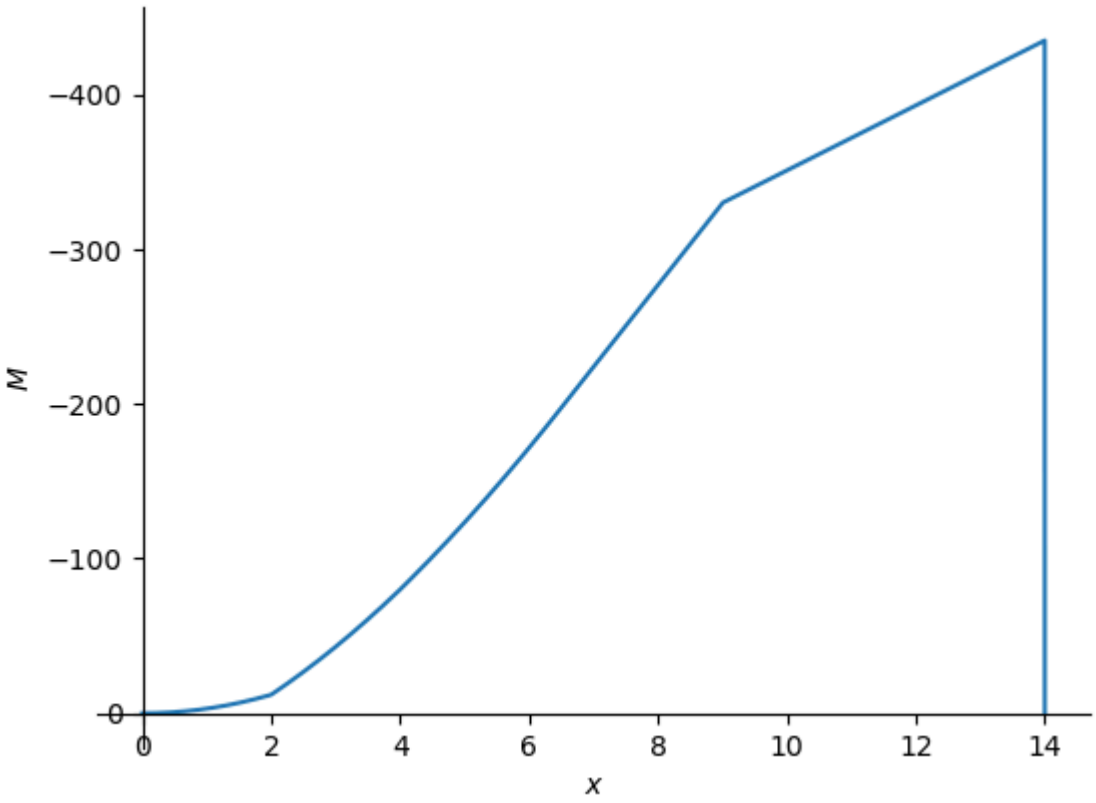

```
M_np = sym.lambdify(x, M.subs(sol).rewrite(sym.Piecewise))
display(M.subs(sol))
display(f'M(4) = {M.subs(x,4).subs(sol)}')
display(f'M(9) = {M.subs(x,9).subs(sol)}')
display(f'M(14) = {M.subs(x,14-dx).subs(sol)}')

plt.figure()
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-3\langle x\rangle^2 - 16\langle x-2.0\rangle^1 - 1.0\langle x-4\rangle^1 + 0.6\langle x-4\rangle^2 + 2.4\langle x-6.5\rangle^2 + 32.0\langle x-9\rangle^1 -$$

```
'M(4) = -80.0000000000000'
```

```
'M(9) = -330.000000000000'
```

```
'M(14) = -435.000000000000'
```



```
v_np = sym.lambdify(x, v.rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,3*10**4).subs(EA,10**4).rewrite(sym.Piecewis
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,3*10**4).subs(EA,10**4).rewrite(sym.Piecewis

#display(uv.subs(sol))
display(f'uv(0) = {uv.subs(x,0).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')
display(f'uv(4) = {uv.subs(x,4).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')
display(f'uv(9) = {uv.subs(x,9).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')

#display(uh.subs(sol))
display(f'uh(0) = {uh.subs(x,0).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')
display(f'uh(4) = {uh.subs(x,4).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')
display(f'uh(9) = {uh.subs(x,9).subs(sol).subs(EI,3*10**4).subs(EA,10**4):.4f}')

plt.figure()
plt.plot(h_np(x_np),v_np(x_np), linewidth=2, color='black', label='constructie')
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), label='vervormde constructie'
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0) = 0.8609'
```

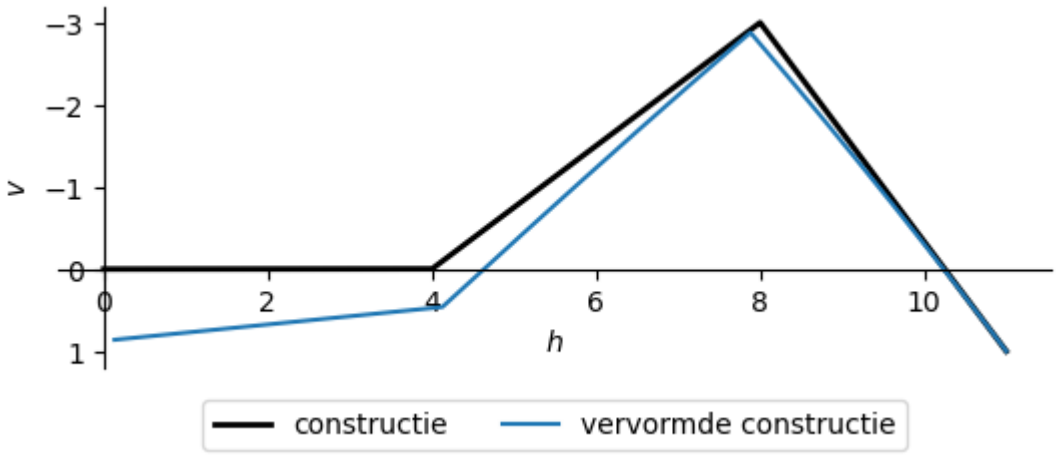
```
'uv(4) = 0.4626'
```

```
'uv(9) = 0.1212'
```

```
'uh(0) = 0.1329'
```

```
'uh(4) = 0.1269'
```

```
'uh(9) = -0.1174'
```


−3
−2
−1
0
1
v
0
2
4
6
8
10
h
constructie
vervormde constructie

Example 2

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 10**-15

# gegevens constructie
a0, a1, a2 = 0, 5, 10
aa = np.array([a0, a1, a2])
o0, o1 = sym.atan(4/3), sym.atan(-4/3)
oo = np.array([o0, o1])
L = 10

# gegevens belastingen
qh = 60
RvA, RhA, RvC, RhC = sym.symbols('R_v^A R_h^A R_v^C R_h^C')
phisB = sym.symbols('phi_s^B')
B = np.array([RvA, RhA, qh, phisB, RvC, RhC])
b1, b2, b3 = 0, 5, 10
bb = np.array([b1, b1, b1, b2, b3, b3])
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5
nn = np.array([2, 3, 5, 6, 2, 3])
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$EI\phi_s^B\langle x-5\rangle^{-3} + 0.8R_h^A\langle x\rangle^{-1} - 1.6R_h^A\langle x-5\rangle^{-1} - 0.8R_h^C\langle x-10\rangle^{-1} + 0.6R_v^A\langle x\rangle^{-}$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$0.6R_h^A\langle x\rangle^{-1} + 0.6R_h^C\langle x-10\rangle^{-1} - 0.8R_v^A\langle x\rangle^{-1} + 1.6R_v^A\langle x-5\rangle^{-1} + 0.8R_v^C\langle x-10\rangle$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - 0.6R_h^A\langle x\rangle^0 - 0.6R_h^C\langle x-10\rangle^0 + 0.8R_v^A\langle x\rangle^0 - 1.6R_v^A\langle x-5\rangle^0 - 0.8R_v^C\langle x-10$$

$$V =$$

$$C_V - EI\phi_s^B\langle x-5\rangle^{-2} - 0.8R_h^A\langle x\rangle^0 + 1.6R_h^A\langle x-5\rangle^0 + 0.8R_h^C\langle x-10\rangle^0 - 0.6R_v^A\langle x$$

$$M =$$

$$C_M + C_V x - EI\phi_s^B\langle x-5\rangle^{-1} - 0.8R_h^A\langle x\rangle^1 + 1.6R_h^A\langle x-5\rangle^1 + 0.8R_h^C\langle x-10\rangle^1 - 0$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - 0.6R_h^A\langle x\rangle^1 - 0.6R_h^C\langle x-10\rangle^1 + 0.8R_v^A\langle x\rangle^1 - 1.6R_v^A\langle x-5\rangle^1 - 0.8R}{EA}$$

$$\phi =$$

$$C_\phi + \frac{C_M x + \frac{C_V x^2}{2} - EI\phi_s^B\langle x-5\rangle^0 - 0.4R_h^A\langle x\rangle^2 + 0.8R_h^A\langle x-5\rangle^2 + 0.4R_h^C\langle x-1}{I}$$

$$u_z =$$

$$-C_\phi x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi_s^B\langle x-5\rangle^1 - 0.133333333333333R_h^A\langle x\rangle^3 + 0.2(}{}$$

$$u_v =$$

$$-0.8C_{ux} + 0.6C_{uz} + 0.6\left(-C_\phi x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi_s^B\langle x-5\rangle^1 - 0.1333333333}{}\right.$$

$$u_h =$$

$$0.6C_{ux} + 0.8C_{uz} + 0.8\left(-C_\phi x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi_s^B\langle x-5\rangle^1 - 0.13333333333}{}\right.$$

```
# 4 reactiekrachten + 1 scharnier + 6 integratieconstanten = 11 voorwaarden
Eq1 = sym.Eq(N.subs(x,0-dx),0)
Eq2 = sym.Eq(N.subs(x,L+dx),0)
Eq3 = sym.Eq(V.subs(x,0-dx),0)
Eq4 = sym.Eq(V.subs(x,L+dx),0)
Eq5 = sym.Eq(M.subs(x,0),0)
Eq6 = sym.Eq(M.subs(x,a1+dx),0)
Eq7 = sym.Eq(M.subs(x,L),0)
Eq8 = sym.Eq(uv.subs(x,0),0)
Eq9 = sym.Eq(uv.subs(x,L),0)
Eq10 = sym.Eq(uh.subs(x,0),0)
Eq11 = sym.Eq(uh.subs(x,L),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11),(RvA,RhA,RvC,RhC,phisB,CN
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: 0.333333333333333*(-750.0*EA - 1000.0*EI)/(EA*EI),
 C_ux: 0.0,
 C_uz: 0.0,
 R_h^A: -300.000000000000,
 R_h^C: -300.000000000000,
 R_v^A: 200.000000000000,
 R_v^C: -200.000000000000,
 phi_s^B: 0.0}
```

```
ea = 10**4
ei = 10**4

display(f'{RvA} = {RvA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhA} = {RhA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvC} = {RvC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhC} = {RhC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{phisB} = {phisB.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
```

```
'R_v^A = 200.00'
```

```
'R_h^A = -300.00'
```

```
'R_v^C = -200.00'
```

```
'R_h^C = -300.00'
```

```
'phi_s^B = 0.0000'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = -0.0583'
```

```
'C_ux = 0.0000'
```

```
'C_uz = 0.0000'
```

```
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
```

```
L = 10
x_np = np.linspace(0-dx,L+dx,10000)
ab = aa
```

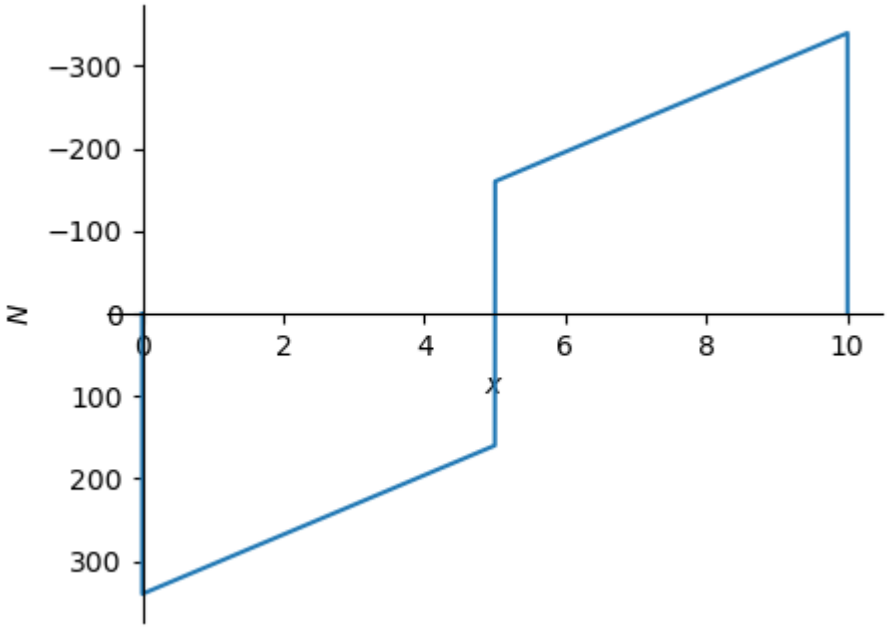
```
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'N({ab[i]}) = {N.subs(x,ab[i]+dx).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure(figsize=(5,4))
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$340.0\langle x\rangle^{0} - 36.0\langle x\rangle^{1} - 320.0\langle x-5\rangle^{0} + 340.0\langle x-10\rangle^{0}$$

```
'N(0) = 340.000000000000'
```

'N(5) = -160.000000000000'

'N(10) = -5.68434188608080E-14'



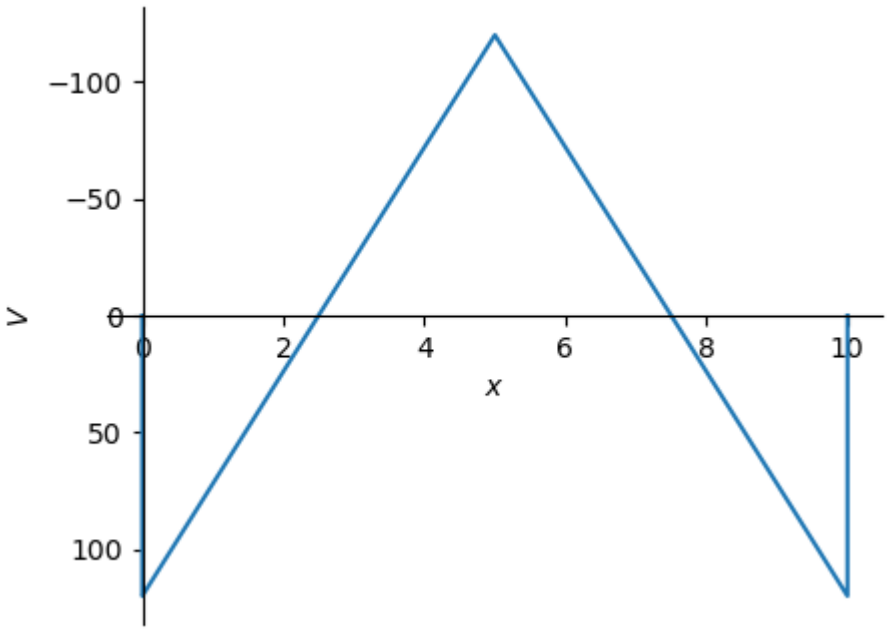

```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'V({ab[i]}) = {V.subs(x,ab[i]+dx).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure(figsize=(5,4))
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$120.0\langle x\rangle^{0} - 48.0\langle x\rangle^{1} + 96.0\langle x-5\rangle^{1} - 120.0\langle x-10\rangle^{0}$$

'V(0) = 120.000000000000'

'V(5) = -120.000000000000'

'V(10) = 5.68434188608080E-14'



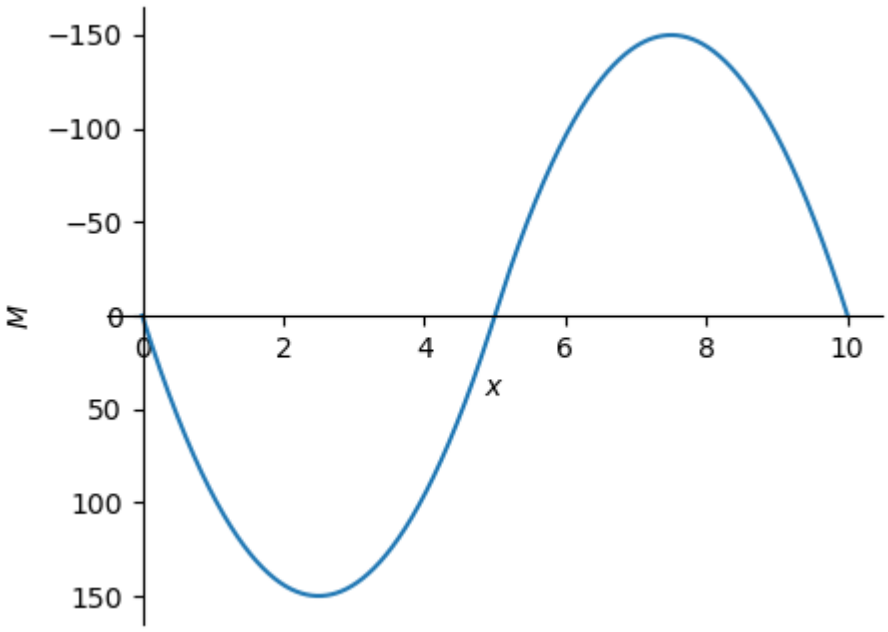

```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(M.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'M({ab[i]}) = {M.subs(x,ab[i]-dx).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure(figsize=(5,4))
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$120.0\langle x\rangle^{1} - 24.0\langle x\rangle^{2} + 48.0\langle x-5\rangle^{2} - 120.0\langle x-10\rangle^{1}$$

```
'M(0) = 0'
```

```
'M(5) = 0'
```

```
'M(10) = -4.54747350886464E-13'
```




```
v_np = sym.lambdify(x, v.rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
for i in range(len(ab)):
    display(f'uv({ab[i]}) = {uv.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)}')

for i in range(len(ab)):
    display(f'uh({ab[i]}) = {uh.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure()
plt.plot(h_np(x_np),v_np(x_np), marker='.',markersize=1, linewidth=0, color='black', labe
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), marker='.',markersize=0.5, li
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0) = 0'
```

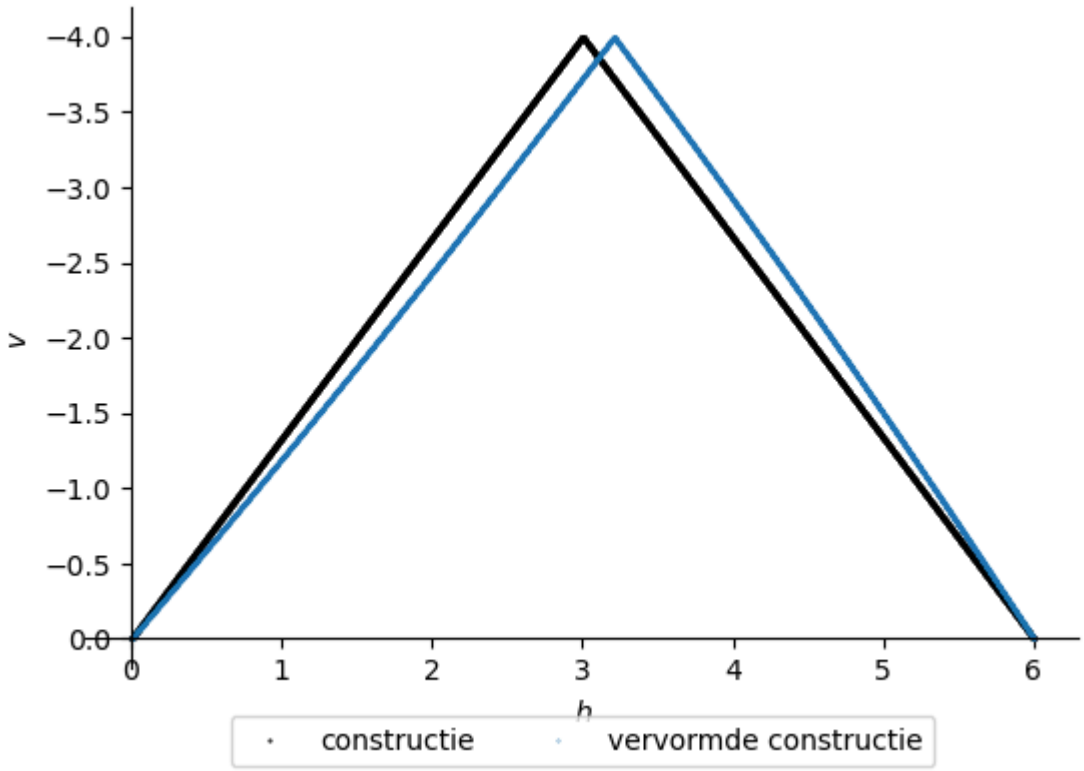
```
'uv(5) = 8.32667268468867E-17'
```

```
'uv(10) = 0'
```

```
'uh(0) = 0'
```

```
'uh(5) = 0.208333333333333'
```

```
'uh(10) = 1.22781784739345E-16'
```



Example 3

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 10**-15

# gegevens constructie
a0, a1, a2 = 0, 5, 10
aa = np.array([a0, a1, a2])
o0, o1 = sym.atan(4/3), sym.atan(-3/4)
oo = np.array([o0, o1])
L = 10

# gegevens belastingen
Fv, qh = 60, 18
RvA, RhA, TrA, RvC = sym.symbols('R_v^A R_h^A T_R^A R_v^C')
B = np.array([RvA, RhA, TrA, Fv, qh, RvC])
b1, b2, b3 = 0, 5, 10
bb = np.array([b1, b1, b1, b2, b2, b3])
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5
nn = np.array([2, 3, 1, 2, 5, 2])
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$0.8R_h^A\langle x\rangle^{-1} - 1.4R_h^A\langle x-5\rangle^{-1} + 0.6R_v^A\langle x\rangle^{-1} + 0.2R_v^A\langle x-5\rangle^{-1} + 0.8R_v^C\langle x-10\rangle^{-}$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$0.6R_h^A\langle x\rangle^{-1} + 0.2R_h^A\langle x-5\rangle^{-1} - 0.8R_v^A\langle x\rangle^{-1} + 1.4R_v^A\langle x-5\rangle^{-1} + 0.6R_v^C\langle x-10\rangle^{-}$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - 0.6R_h^A\langle x\rangle^0 - 0.2R_h^A\langle x-5\rangle^0 + 0.8R_v^A\langle x\rangle^0 - 1.4R_v^A\langle x-5\rangle^0 - 0.6R_v^C\langle x-10\rangle$$

$$V =$$

$$C_V - 0.8R_h^A\langle x\rangle^0 + 1.4R_h^A\langle x-5\rangle^0 - 0.6R_v^A\langle x\rangle^0 - 0.2R_v^A\langle x-5\rangle^0 - 0.8R_v^C\langle x-10\rangle$$

$$M =$$

$$C_M + C_V x - 0.8R_h^A\langle x\rangle^1 + 1.4R_h^A\langle x-5\rangle^1 - 0.6R_v^A\langle x\rangle^1 - 0.2R_v^A\langle x-5\rangle^1 - 0.8R_v^C\langle$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - 0.6R_h^A\langle x\rangle^1 - 0.2R_h^A\langle x-5\rangle^1 + 0.8R_v^A\langle x\rangle^1 - 1.4R_v^A\langle x-5\rangle^1 - 0.6R_v^C}{EA}$$

$$\phi =$$

$$C_\phi + \frac{C_M x + \frac{C_V x^2}{2} - 0.4R_h^A\langle x\rangle^2 + 0.7R_h^A\langle x-5\rangle^2 - 0.3R_v^A\langle x\rangle^2 - 0.1R_v^A\langle x-5\rangle^2 -}{EI}$$

$$u_z =$$

$$-C_\phi x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 0.133333333333333R_h^A\langle x\rangle^3 + 0.233333333333333I}$$

$$u_v =$$

$$-0.8C_{ux} + 0.6C_{uz} + 0.6\left(-C_\phi x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 0.133333333333333R_h^A\langle x\rangle^3 + (\right.$$

$$u_h =$$

$$0.6C_{ux} + 0.8C_{uz} + 0.8\left(-C_\phi x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 0.133333333333333R_h^A\langle x\rangle^3 + 0.2\right.$$

```
# 4 reactiekrachten + 6 integratieconstanten = 10 voorwaarden
Eq1 = sym.Eq(N.subs(x,0-dx),0)
Eq2 = sym.Eq(N.subs(x,L+dx),0)
Eq3 = sym.Eq(V.subs(x,0-dx),0)
Eq4 = sym.Eq(V.subs(x,L+dx),0)
Eq5 = sym.Eq(M.subs(x,0-dx),0)
Eq6 = sym.Eq(M.subs(x,L),0)
Eq7 = sym.Eq(phi.subs(x,0),0)
Eq8 = sym.Eq(uv.subs(x,0),0)
Eq9 = sym.Eq(uv.subs(x,L),0)
Eq10 = sym.Eq(uh.subs(x,0),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10),(RvA,RhA,TrA,RvC,CN,CV,CM,Cphi
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: 0.0,
 C_ux: 0.0,
 C_uz: 0.0,
 R_h^A: -90.0000000000000,
 R_v^A: (-43125.0*EA - 1296.0*EI)/(1090.0*EA + 30.0*EI),
 R_v^C: (-22275.0*EA - 504.0*EI)/(1090.0*EA + 30.0*EI),
 T_R^A: (285525.0*EA + 8622.0*EI)/(1090.0*EA + 30.0*EI)}
```

```
ea = 10**4
ei = 10**4

display(f'{RvA} = {RvA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhA} = {RhA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{TrA} = {TrA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvC} = {RvC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
```

```
'R_v^A = -39.66'
```

```
'R_h^A = -90.00'
```

```
'T_R^A = 262.63'
```

```
'R_v^C = -20.34'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = 0.0000'
```

```
'C_ux = 0.0000'
```

```
'C_uz = 0.0000'
```

```
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
```

```
x_np = np.linspace(0-dx,L+dx,10000)
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))
display(f'N(0) = {N.subs(x,0).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'N(5-) = {N.subs(x,5-dx).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'N(5+) = {N.subs(x,5).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'N(10) = {N.subs(x,10-dx).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure()
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

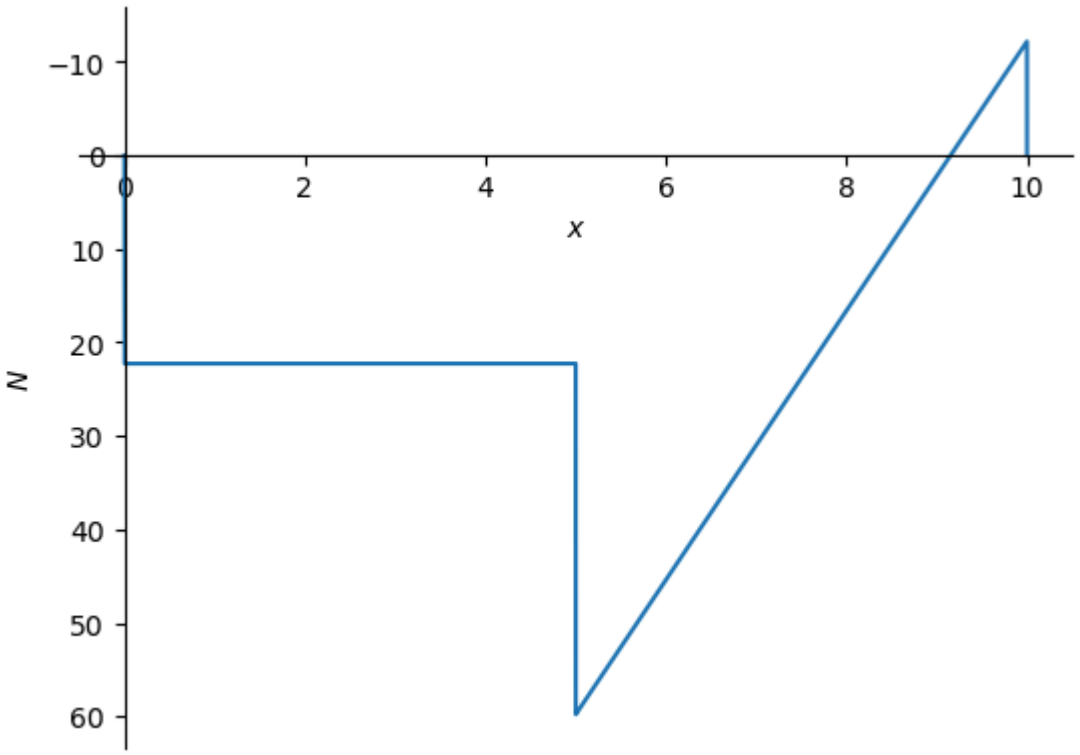
$$22.2707142857143\langle x\rangle^{0} + 37.52625\langle x-5\rangle^{0} - 14.4\langle x-5\rangle^{1} + 12.2030357142857\langle x$$

```
'N(0) = 22.2707142857143'
```

```
'N(5-) = 22.2707142857143'
```

```
'N(5+) = 59.7969642857143'
```

```
'N(10) = -12.2030357142857'
```




```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))
display(f'V(0) = {V.subs(x,0+dx).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'V(5-) = {V.subs(x,5-dx).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'V(5+) = {V.subs(x,5).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'V(10) = {V.subs(x,10-dx).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure()
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

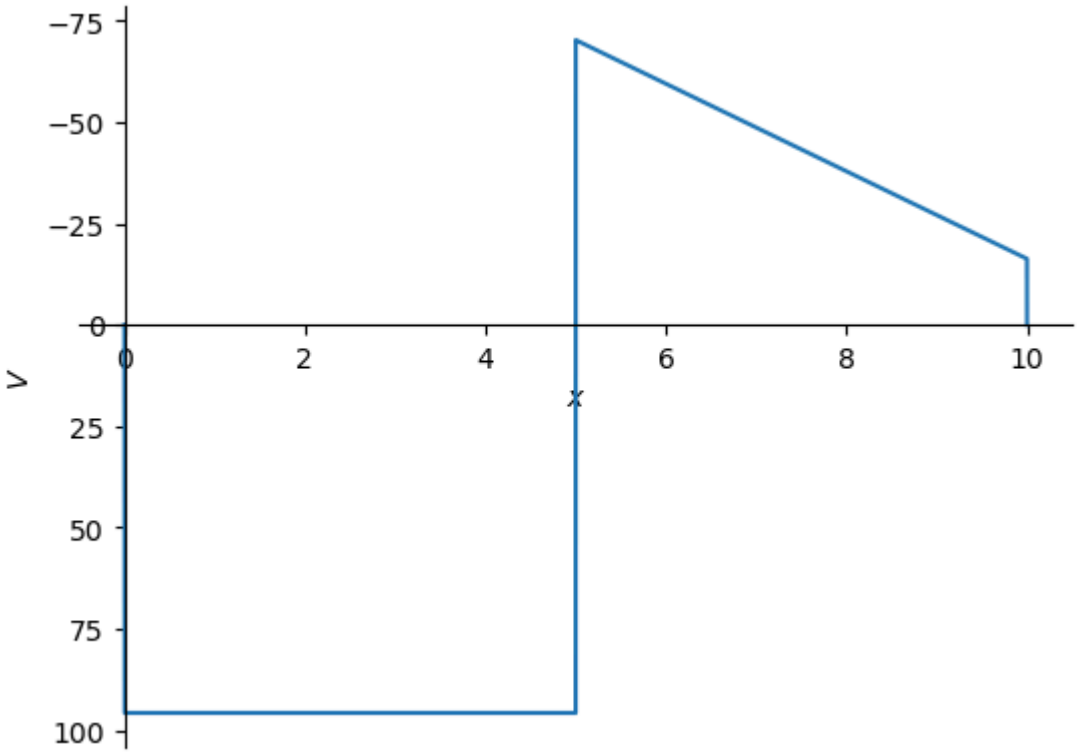
$$-262.63125\langle x\rangle^{-1} + 95.7969642857143\langle x\rangle^{0} - 166.067678571429\langle x-5\rangle^{0} + 10.8\langle x$$

```
'V(0) = 95.7969642857143'
```

```
'V(5-) = 95.7969642857143'
```

```
'V(5+) = -70.2707142857143'
```

```
'V(10) = -16.2707142857143'
```



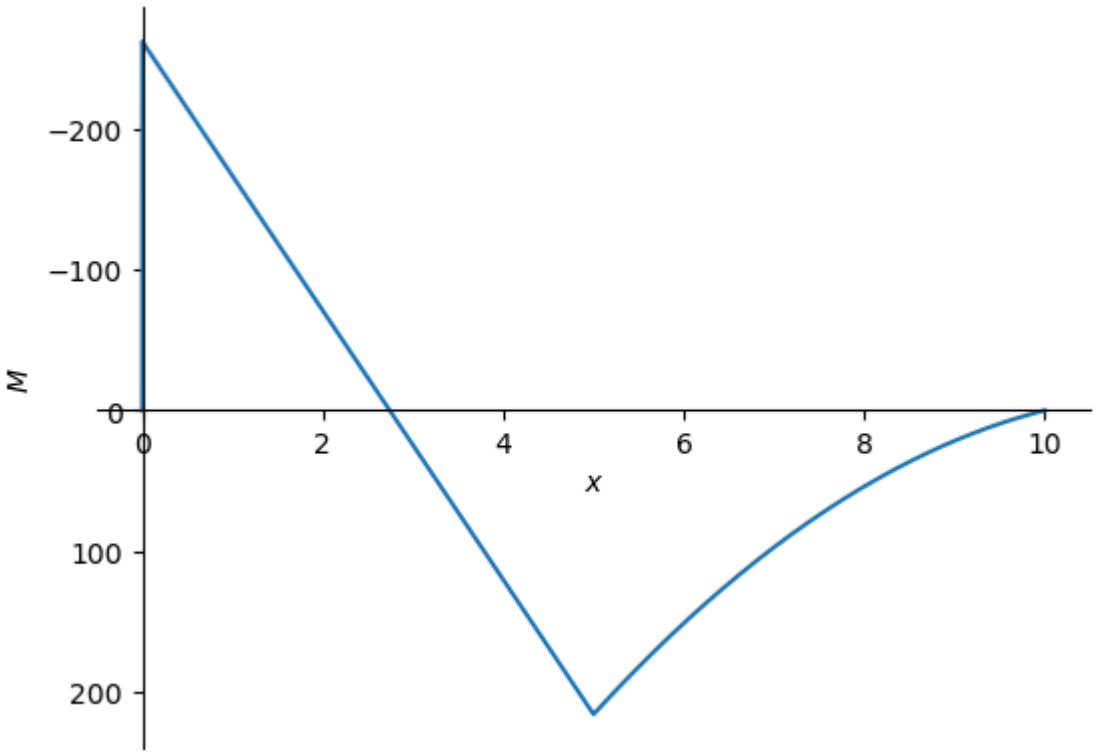

```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(M.subs(sol).subs(EI,ei).subs(EA,ea))
display(f'M(0) = {M.subs(x,0).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'M(5) = {M.subs(x,5).subs(sol).subs(EI,ei).subs(EA,ea)}')
display(f'M(10) = {M.subs(x,10).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure()
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-262.63125\langle x\rangle^{0} + 95.7969642857143\langle x\rangle^{1} - 166.067678571429\langle x - 5\rangle^{1} + 5.4\langle x -$$

```
'M(0) = -262.631250000000'
```

```
'M(5) = 216.353571428571'
```

```
'M(10) = -1.13686837721616E-13'
```




```
v_np = sym.lambdify(x, v.rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
display(f'uv(0) = {uv.subs(x,0).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'uv(5) = {uv.subs(x,5).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'uv(10) = {uv.subs(x,10).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

#display(uh.subs(sol))
display(f'uh(0) = {uh.subs(x,0).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'uh(5) = {uh.subs(x,5).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'uh(10) = {uh.subs(x,10).subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

plt.figure()
plt.plot(h_np(x_np),v_np(x_np), linewidth=2, color='black', label='constructie')
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), label='vervormde constructie'
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0) = 0.0000'
```

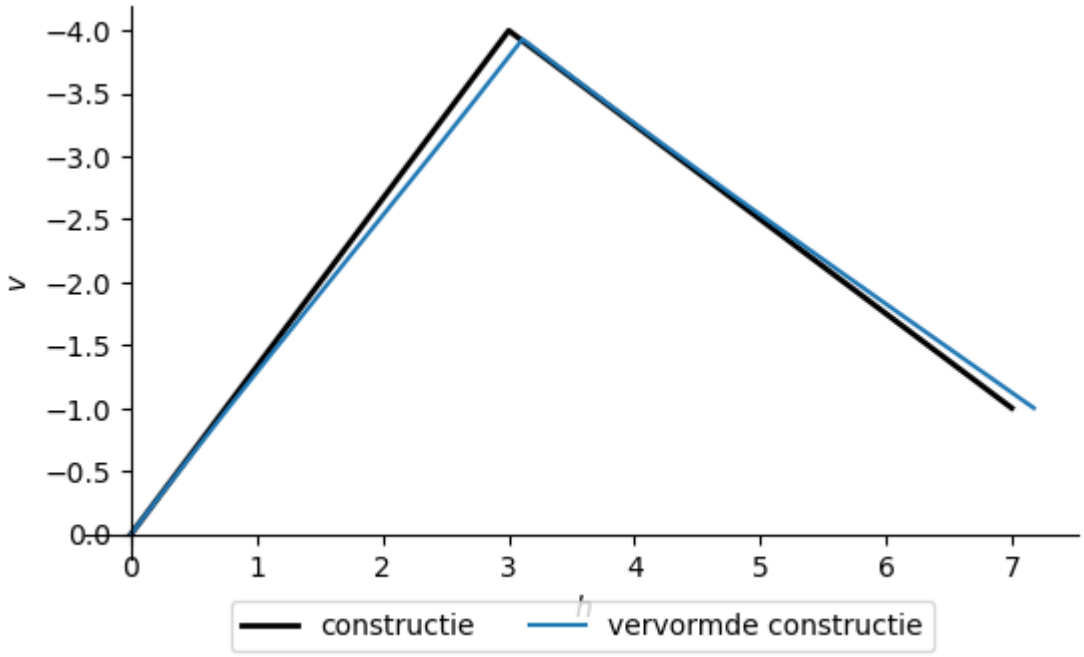
```
'uv(5) = 0.0683'
```

```
'uv(10) = 0.0000'
```

```
'uh(0) = 0.0000'
```

```
'uh(5) = 0.1097'
```

```
'uh(10) = 0.1758'
```



Example 4

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 2*10**-15

# gegevens constructie
a0, a1, a2, a3, a4, a5 = 0, 2, 4.5, 7, 12, 14.5
aa = [a0, a1, a2, a3, a4, a5]
o0, o1, o2, o3, o4 = 0, sym.atan(3/4), sym.atan(3/4), sym.atan(-3/4), sym.atan(-3/4)
oo = [o0, o1, o2, o3, o4]

# sprongen in de x functie
phiBD, uzBD, uxBD, phiCF, uzCF, uxCF = sym.symbols('phi^BD u_z^BD u_x^BD phi^CF u_z^CF u_
s1, s2 = 7+2*dx, 12+2*dx
ss = [s1, s2]

# gegevens belastingen
Fv = 20
Rv, Rh, Tr = sym.symbols('R_v R_h T_R')
VBF, HBF, TBF, VCE, HCE, TCE = sym.symbols('V^BF H^BF T^BF V^CE H^CE T^CE')
B = [Rv, Rh, Tr, VBF, HBF, TBF, VCE, HCE, TCE, Fv, -VBF, -HBF, -TBF, phiBD, uzBD, uxBD, F
b1, b2, b3, b4, b5, b6, b7, b8 = 0, 2, 4.5, 7, 7+2*dx, 12, 12+2*dx, 14.5
bb = [b1, b1, b1, b2, b2, b2, b3, b3, b3, b4, b5, b5, b5, s1, s1, s1, b6, b7, b7, b7, s2,
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5, phi = 6, uz = 7, ux = 8
nn = [2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 2, 3, 1, 6, 7, 8, 2, 2, 3, 1, 6, 7, 8, 2]
```

```
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 7:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-4) * EI
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$EI\phi^{BD}\langle x-7.0\rangle^{-3} + EI\phi^{CF}\langle x-12.0\rangle^{-3} + EIu_z^{BD}\langle x-7.0\rangle^{-4} + EIu_z^{CF}\langle x-12.0\rangle$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 8:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-2) * EA
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$EAu_x^{BD}\langle x-7.0\rangle^{-2} + EAu_x^{CF}\langle x-12.0\rangle^{-2} + 0.8H^{BF}\langle x-2\rangle^{-1} - 0.8H^{BF}\langle x-7.0\rangle$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - EAu_x^{BD}\langle x - 7.0\rangle^{-1} - EAu_x^{CF}\langle x - 12.0\rangle^{-1} - 0.8H^{BF}\langle x - 2\rangle^{0} + 0.8H^{BF}\langle x -$$

$$V =$$

$$C_V - EI\phi^{BD}\langle x - 7.0\rangle^{-2} - EI\phi^{CF}\langle x - 12.0\rangle^{-2} - EIu_z^{BD}\langle x - 7.0\rangle^{-3} - EIu_z^{CF}\langle x -$$

$$M =$$

$$C_M + C_V x - EI\phi^{BD}\langle x - 7.0\rangle^{-1} - EI\phi^{CF}\langle x - 12.0\rangle^{-1} - EIu_z^{BD}\langle x - 7.0\rangle^{-2} - EI$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - EAu_x^{BD}\langle x - 7.0\rangle^{0} - EAu_x^{CF}\langle x - 12.0\rangle^{0} - 0.8H^{BF}\langle x - 2\rangle^{1} + 0.8H^{l}$$

$$\phi =$$

$$C_{\phi} + \frac{C_M x + \frac{C_V x^2}{2} - EI\phi^{BD}\langle x - 7.0\rangle^{0} - EI\phi^{CF}\langle x - 12.0\rangle^{0} - EIu_z^{BD}\langle x - 7.0\rangle^{-1}$$

$$u_z =$$

$$-C_{\phi} x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi^{BD}\langle x - 7.0\rangle^{1} - EI\phi^{CF}\langle x - 12.0\rangle^{1} - EIu_z^{BD}\langle x$$

$$u_v =$$

$$C_{uz} + \left(-C_{\phi} x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi^{BD}\langle x - 7.0\rangle^{1} - EI\phi^{CF}\langle x - 12.0\rangle^{1} - EIu_z^{BD}\right.$$

$$u_h =$$

$$C_{ux} - 0.2\left(-\frac{2C_N - 2R_h}{EA} + \frac{C_N x - EAu_x^{BD}\langle x - 7.0\rangle^{0} - EAu_x^{CF}\langle x - 12.0\rangle^{0} - 0.8\right.$$

```
# 3 reactiekrachten + 6 knoopkrachten + 6 vervormingssprongen + 6 integratieconstanten =
Eq1 = sym.Eq(N.subs(x,0-dx),0)
Eq2 = sym.Eq(N.subs(x,a3),0)
Eq3 = sym.Eq(N.subs(x,a4),0)
Eq4 = sym.Eq(N.subs(x,a5),0)
Eq5 = sym.Eq(V.subs(x,0-dx),0)
Eq6 = sym.Eq(V.subs(x,a3),0)
Eq7 = sym.Eq(V.subs(x,a4),0)
Eq8 = sym.Eq(V.subs(x,a5),0)
Eq9 = sym.Eq(M.subs(x,0-dx),0)
Eq10 = sym.Eq(M.subs(x,a3),0)
Eq11 = sym.Eq(M.subs(x,a4),0)
Eq12 = sym.Eq(M.subs(x,a5),0)
Eq13 = sym.Eq(phi.subs(x,0),0)
Eq14 = sym.Eq(uv.subs(x,0),0)
Eq15 = sym.Eq(uh.subs(x,0),0)
Eq16 = sym.Eq(phi.subs(x,s1+dx)-phi.subs(x,a1),0)
Eq17 = sym.Eq(phi.subs(x,s2+dx)-phi.subs(x,a2),0)
Eq18 = sym.Eq(uv.subs(x,s1)-uv.subs(x,a1),0)
Eq19 = sym.Eq(uv.subs(x,s2)-uv.subs(x,a2),0)
Eq20 = sym.Eq(uh.subs(x,s1)-uh.subs(x,a1),0)
Eq21 = sym.Eq(uh.subs(x,s2)-uh.subs(x,a2),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11,Eq12,Eq13,Eq14,Eq15,Eq16,E
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: 0.0,
 C_ux: 0.0,
 C_uz: 0.0,
 H^BF: 0.0,
 H^CE: 0.0,
 R_h: 0.0,
 R_v: -60.0000000000000,
 T^BF: -80.0000000000000,
 T^CE: -40.0000000000000,
 T_R: 360.000000000000,
 V^BF: 20.0000000000000,
 V^CE: 20.0000000000000,
 phi^BD: -350.0/EI,
 phi^CF: 100.0/EI,
 u_x^BD: 2.5e-27*(1.632e+30*EA - 1.008e+28*EI)/(EA*EI),
 u_x^CF: 1.0e-28*(-1.84e+31*EA + 7.68000000000002e+29*EI)/(EA*EI),
 u_z^BD: 7.96545955566226e-60*(-1.49395021302203e+62*EA - 1.08468317987482e+61*EI)/(EA*EI
 u_z^CF: 8.75811540203011e-88*(-3.57382822253572e+90*EA + 6.57675736798906e+88*EI)/(EA*EI
```

```
#display(sym.simplify(uz.subs(sol).rewrite(sym.Piecewise)))
ea = 10**4
ei = 10**4

display(f'{Rv} = {Rv.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Rh} = {Rh.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Tr} = {Tr.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{VBF} = {VBF.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HBF} = {HBF.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{TBF} = {TBF.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{VCE} = {VCE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HCE} = {HCE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{TCE} = {TCE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{phiBD} = {phiBD.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uzBD} = {uzBD.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uxBD} = {uxBD.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{phiCF} = {phiCF.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uzCF} = {uzCF.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uxCF} = {uxCF.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
```

```
'R_v = -60.00'
```

```
'R_h = 0.00'
```

```
'T_R = 360.00'
```

```
'V^BF = 20.00'
```

```
'H^BF = 0.00'
```

```
'T^BF = -80.00'
```

```
'V^CE = 20.00'
```

```
'H^CE = 0.00'
```

```
'T^CE = -40.00'
```

```
'phi^BD = -0.0350'
```

```
'u_z^BD = -0.1276'
```

```
'u_x^BD = 0.4055'
```

```
'phi^CF = 0.0100'
```

```
'u_z^CF = -0.3072'
```

```
'u_x^CF = -0.1763'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = 0.0000'
```

```
'C_ux = 0.0000'
```

```
'C_uz = 0.0000'
```

```
vBD, hBD, vCF, hCF = sym.symbols('v_BD h_BD v_CF h_CF')
vv = [vBD, vCF]
hh = [hBD, hCF]
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
for i in range(len(ss)):
    v += vv[i] * sym.SingularityFunction(x,ss[i],0)
    h += hh[i] * sym.SingularityFunction(x,ss[i],0)

# 2 x 2 sprongen = 4 onbekenden
Eq1 = sym.Eq(v.subs(x,a1),v.subs(x,s1))
Eq2 = sym.Eq(v.subs(x,a2),v.subs(x,s2))
Eq3 = sym.Eq(h.subs(x,a1),h.subs(x,s1))
Eq4 = sym.Eq(h.subs(x,a2),h.subs(x,s2))

sol2 = sym.solve((Eq1,Eq2,Eq3,Eq4),(vBD, hBD, vCF, hCF))
display(sol2)
```

```
{h_BD: -4.00000000000000,
 h_CF: -2.00000000000000,
 v_BD: 3.00000000000000,
 v_CF: -4.50000000000000}
```

```
L = 14.5
x_np = np.linspace(0-dx,L+dx,10000)
ab = aa
ab.extend(bb)
ab.sort()
ab = list(dict.fromkeys(ab))
```

```
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'N({ab[i]:.1f}) = {N.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)+dx}')

plt.figure(figsize=(4,4))
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-24.0\langle x-2\rangle^{0}+12.0\langle x-4.5\rangle^{0}+12.0\langle x-7\rangle^{0}-4054.8\langle x-7.0\rangle^{-1}+12.0\langle x-7.0$$

```
'N(0.0) = 2.00000000000000E-15'
```

```
'N(2.0) = -24.0000000000000'
```

```
'N(4.5) = -12.0000000000000'
```

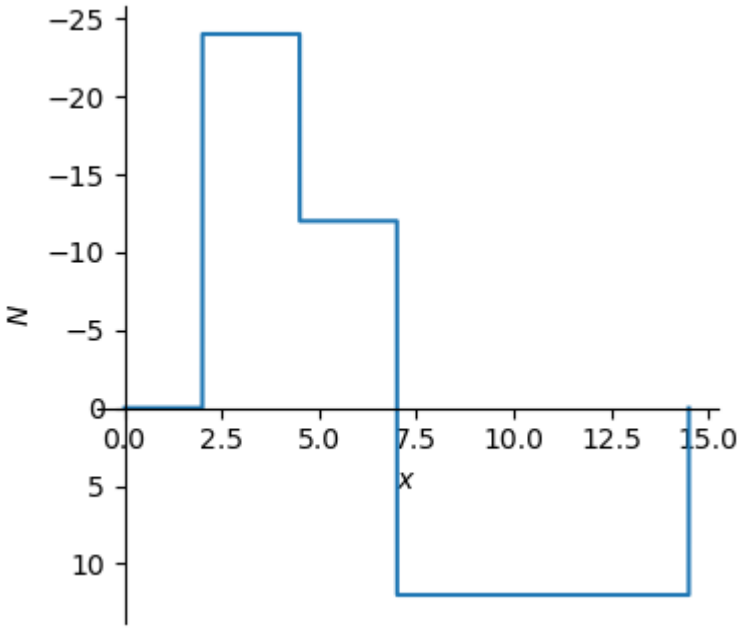
```
'N(7.0) = 2.00000000000000E-15'
```

```
'N(7.0) = -oo'
```

```
'N(12.0) = 2.00000000000000E-15'
```

```
'N(12.0) = oo'
```

```
'N(14.5) = 2.00000000000000E-15'
```



```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'V({ab[i]:.1f}) = {V.subs(x,ab[i]-dx).subs(sol).subs(EI,ei).subs(EA,ea)+dx:.

plt.figure(figsize=(4,4))
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-360.0\langle x\rangle^{-1} + 60.0\langle x\rangle^{0} + 80.0\langle x-2\rangle^{-1} - 28.0\langle x-2\rangle^{0} + 40.0\langle x-4.5\rangle^{-1} - 16.0\langle$$

```
'V(0.0) = 0.00'
```

```
'V(2.0) = 60.00'
```

```
'V(4.5) = 32.00'
```

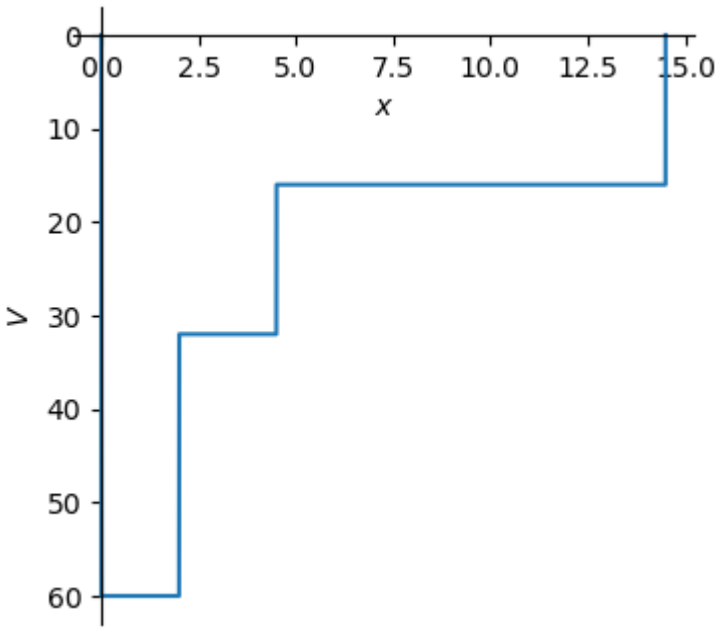
```
'V(7.0) = 16.00'
```

```
'V(7.0) = 0.00'
```

```
'V(12.0) = 16.00'
```

```
'V(12.0) = 0.00'
```

```
'V(14.5) = 16.00'
```




```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

for i in range(len(ab)):
    display(f'M({ab[i]}) = {M.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)}')

plt.figure(figsize=(4,4))
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

```
'M(0) = -360.000000000000'
```

```
'M(2) = -160.000000000000'
```

```
'M(4.5) = -40.0000000000000'
```

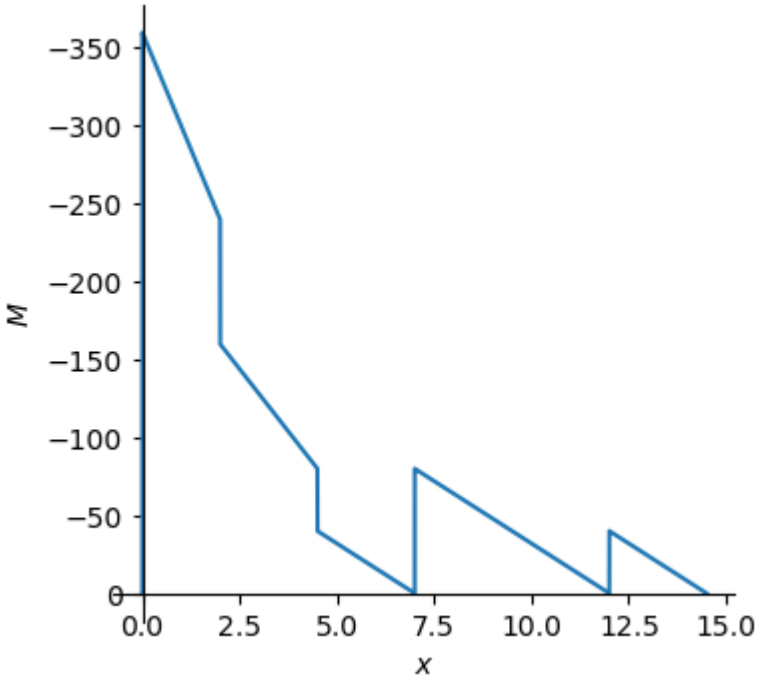
```
'M(7) = 0'
```

```
'M(7.000000000000004) = oo'
```

```
'M(12) = -7.10542735760100E-14'
```

```
'M(12.000000000000004) = nan'
```

```
'M(14.5) = -1.27897692436818E-13'
```




```
v_np = sym.lambdify(x, v.subs(sol2).rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.subs(sol2).rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
for i in range(len(ab)):
    display(f'uv({ab[i]:.1f}) = {uv.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)+dx:.4

for i in range(len(ab)):
    display(f'uh({ab[i]:.1f}) = {uh.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)+dx:.4

plt.figure()
plt.plot(h_np(x_np),v_np(x_np), marker='.',markersize=1, linewidth=0, color='black', labe
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), marker='.',markersize=0.5, li
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0.0) = 0.0000 [m]'
```

```
'uv(2.0) = 0.0640 [m]'
```

```
'uv(4.5) = 0.2209 [m]'
```

```
'uv(7.0) = 0.4094 [m]'
```

```
'uv(7.0) = 0.0640 [m]'
```

```
'uv(12.0) = 0.3609 [m]'
```

```
'uv(12.0) = 0.2209 [m]'
```

```
'uv(14.5) = 0.4094 [m]'
```

```
'uh(0.0) = 0.0000 [m]'
```

```
'uh(2.0) = 0.0000 [m]'
```

```
'uh(4.5) = 0.1102 [m]'
```

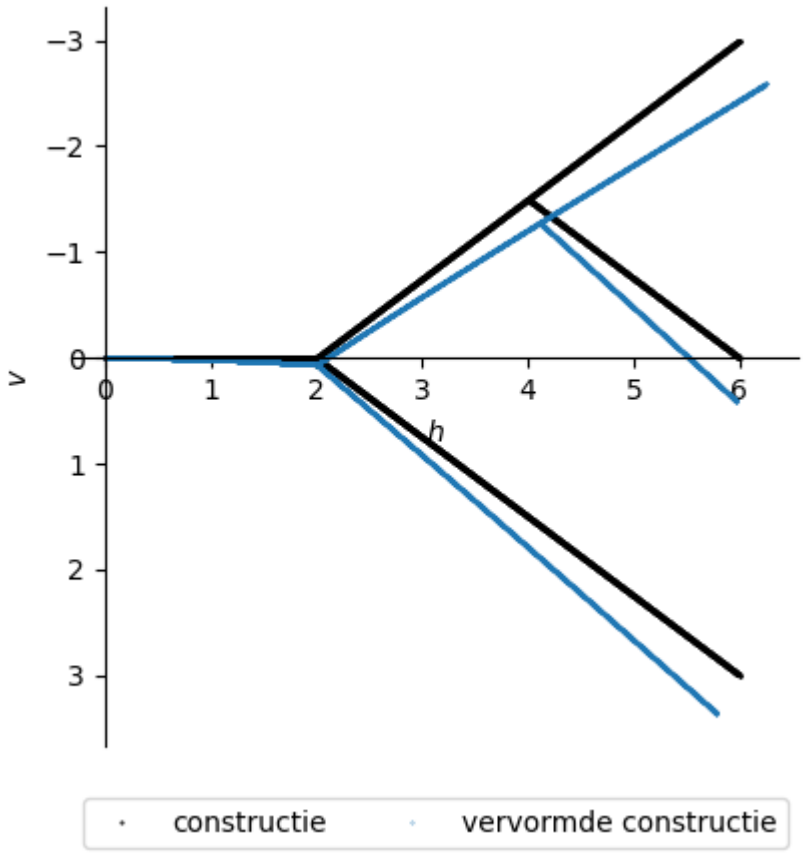
```
'uh(7.0) = 0.2478 [m]'
```

```
'uh(7.0) = 0.0000 [m]'
```

```
'uh(12.0) = -0.2152 [m]'
```

```
'uh(12.0) = 0.1102 [m]'
```

```
'uh(14.5) = -0.0274 [m]'
```


−3
−2
−1
0
1
2
3
v
0
1
2
3
4
5
6
h
constructie
vervormde constructie

Example 5

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 2*10**-15

# gegevens constructie
a0, a1, a2, a3, a4, a5 = 0, 4, 8, 12, 16, 20
aa = [a0, a1, a2, a3, a4, a5]
o0, o1, o2, o3, o4 = np.pi/2, 0, 0, -np.pi/2, -np.pi/2
oo = [o0, o1, o2, o3, o4]

# sprongen in de x functie
uzEC, uxEC = sym.symbols('u_z^EC u_x^EC')
s1 = 16+2*dx
ss = [s1]

# gegevens belastingen
qv, qh = 20, 10
RvA, RhA, RvB, RhB, RvC, RhC = sym.symbols('R_v^A R_h^A R_v^B R_h^B R_v^C R_h^C')
phisEB = sym.symbols('phi_s^EB')
VEB, HEB = sym.symbols('V^EB H^EB')
B = [RvA, RhA, qh, -qh, qv, -qv, VEB, HEB, RvC, RhC, -VEB, -HEB, phisEB, uzEC, uxEC, RvB,
b1, b2, b3, b4, b5, b6, b7 = 0, 4, 8, 12, 16, 16+2*dx, 20
bb = [b1, b1, b1, b2, b2, b4, b3, b3, b5, b5, b6, b6, b6, b6, b6, b7, b7]
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5, phi = 6, uz = 7, ux = 8
nn = [2, 3, 5, 5, 4, 4, 2, 3, 2, 3, 2, 3, 6, 7, 8, 2, 3]
```

```
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 7:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-4) * EI
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$2EI\phi_s^{EB}\langle x-16.0\rangle^{-3} + EIu_z^{EC}\langle x-16.0\rangle^{-4} - 1.0H^{EB}\langle x-12\rangle^{-1} + 1.0H^{EB}\langle x-1$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 8:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-2) * EA
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$EAu_x^{EC}\langle x-16.0\rangle^{-2} + H^{EB}\langle x-8\rangle^{-1} - 1.0H^{EB}\langle x-12\rangle^{-1} - 6.12323399573677\cdot$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - EAu_x^{EC}\langle x - 16.0\rangle^{-1} - H^{EB}\langle x - 8\rangle^{0} + 1.0H^{EB}\langle x - 12\rangle^{0} + 6.123233995736$$

$$V =$$

$$C_V - 2EI\phi_s^{EB}\langle x - 16.0\rangle^{-2} - EIu_z^{EC}\langle x - 16.0\rangle^{-3} + 1.0H^{EB}\langle x - 12\rangle^{0} - 1.0H^{EB}\langle$$

$$M =$$

$$C_M + C_V x - 2EI\phi_s^{EB}\langle x - 16.0\rangle^{-1} - EIu_z^{EC}\langle x - 16.0\rangle^{-2} + 1.0H^{EB}\langle x - 12\rangle^{1} - 1.$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - EAu_x^{EC}\langle x - 16.0\rangle^{0} - H^{EB}\langle x - 8\rangle^{1} + 1.0H^{EB}\langle x - 12\rangle^{1} + 6.123233$$

$$\phi =$$

$$C_{\phi} + \frac{C_M x + \frac{C_V x^2}{2} - 2EI\phi_s^{EB}\langle x - 16.0\rangle^{0} - EIu_z^{EC}\langle x - 16.0\rangle^{-1} + 0.5H^{EB}\langle x - 12$$

$$u_z =$$

$$-C_{\phi} x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 2EI\phi_s^{EB}\langle x - 16.0\rangle^{1} - EIu_z^{EC}\langle x - 16.0\rangle^{0} + 0.1666$$

$$u_v =$$

$$-1.0C_{ux} + 6.12323399573677 \cdot 10^{-17} C_{uz} + 6.12323399573677 \cdot 10^{-17} \left(-C_{\phi} x - \frac{C}{-}\right.$$

$$u_h =$$

$$6.12323399573677 \cdot 10^{-17} C_{ux} + 1.0C_{uz} + 1.0 \left(-C_{\phi} x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 2EI\phi_s^{EB}\langle$$

```
# 6 reactiekrachten + 2 knoopkrachten + 2 vervormingssprongen + 1 scharnier + 6 integrati
Eq1 = sym.Eq(N.subs(x,0-dx),0)
Eq2 = sym.Eq(N.subs(x,a4),0)
Eq3 = sym.Eq(N.subs(x,a5),0)
Eq4 = sym.Eq(V.subs(x,0-dx),0)
Eq5 = sym.Eq(V.subs(x,a4),0)
Eq6 = sym.Eq(V.subs(x,a5),0)
Eq7 = sym.Eq(M.subs(x,0),0)
Eq8 = sym.Eq(M.subs(x,a4),0)
Eq9 = sym.Eq(M.subs(x,a5),0)
Eq10 = sym.Eq(uv.subs(x,0),0)
Eq11 = sym.Eq(uv.subs(x,a4),0)
Eq12 = sym.Eq(uv.subs(x,a5),0)
Eq13 = sym.Eq(uh.subs(x,0),0)
Eq14 = sym.Eq(uh.subs(x,a4),0)
Eq15 = sym.Eq(uh.subs(x,a5),0)
Eq16 = sym.Eq(uv.subs(x,s1)-uv.subs(x,a2),0)
Eq17 = sym.Eq(uh.subs(x,s1)-uh.subs(x,a2),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11,Eq12,Eq13,Eq14,Eq15,Eq16,E
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: (-4.35200000000004e+259*EA**3 - 6.24000000000002e+259*EA**2*EI - 1.4112e+259*EA*E
 C_ux: 0.0,
 C_uz: 0.0,
 H^EB: (-2.82440737464642e+213*EA**2 - 3.45057316053499e+213*EA*EI - 1.44884104713583e+21
 R_h^A: (-5.76000000000005e+228*EA**2 - 6.54e+228*EA*EI - 5.4e+227*EI**2)/(2.24e+227*EA**
 R_h^B: (-1.85250730654591e+275*EA**2 - 2.53045465343985e+275*EA*EI - 1.09035331709877e+2
 R_h^C: (-3.19999999999995e+228*EA**2 - 5.94e+228*EA*EI - 3.65932793530848e+211*EI**2)/(2
 R_v^A: (-5.91999999999986e+228*EA**2 - 1.26e+229*EA*EI - 5.85e+227*EI**2)/(2.24e+227*EA*
 R_v^B: (-1.95200000000003e+291*EA**2 - 1.848e+291*EA*EI - 7.19999999999999e+289*EI**2)/(
 R_v^C: (-1.03999999999999e+229*EA**2 - 1.884e+229*EA*EI - 8.55e+227*EI**2)/(2.24e+227*EA
 V^EB: (-1.95200000000003e+229*EA**2 - 1.848e+229*EA*EI - 7.19999999999999e+227*EI**2)/(2
 phi_s^EB: (-2.80832777782885e+303*EA**3 - 4.15983552090897e+303*EA**2*EI + 1.95485941409
 u_x^EC: (-2.01320441798697e+284*EA**3 - 9.76000000000014e+299*EA**2*EI - 9.2399999999999
 u_z^EC: (-1.13231776002033e+355*EA**3 - 1.67151669336335e+355*EA**2*EI - 1.9967210500358
```

```
#display(sym.simplify(uz.subs(sol).rewrite(sym.Piecewise)))
ea = 10**3
ei = 10**3

display(f'{RvA} = {RvA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhA} = {RhA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvB} = {RvB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhB} = {RhB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvC} = {RvC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhC} = {RhC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{VEB} = {VEB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HEB} = {HEB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{phisEB} = {phisEB.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{uzEC} = {uzEC.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uxEC} = {uxEC.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
```

```
'R_v^A = -34.77'
```

```
'R_h^A = -23.37'
```

```
'R_v^B = -70.46'
```

```
'R_h^B = -0.00'
```

```
'R_v^C = -54.77'
```

```
'R_h^C = -16.63'
```

```
'V^EB = -70.46'
```

```
'H^EB = -0.00'
```

```
'phi_s^EB = -0.0139'
```

```
'u_z^EC = -0.5200'
```

```
'u_x^EC = -0.2819'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = -0.1823'
```

```
'C_ux = 0.0000'
```

```
'C_uz = 0.0000'
```

```
vEC, hEC = sym.symbols('v^EC h^EC')
vv = [vEC]
hh = [hEC]
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
for i in range(len(ss)):
    v += vv[i] * sym.SingularityFunction(x,ss[i],0)
    h += hh[i] * sym.SingularityFunction(x,ss[i],0)

# 2 x 2 sprongen = 4 onbekenden
Eq1 = sym.Eq(v.subs(x,a2),v.subs(x,s1))
Eq2 = sym.Eq(h.subs(x,a2),h.subs(x,s1))

sol2 = sym.solve((Eq1,Eq2),(vEC, hEC))
display(sol2)
```

```
{h^EC: -4.00000000000000, v^EC: -4.00000000000000}
```

```
```

```
L = aa[-1]
x_np = np.linspace(0-dx,L+dx,10000)
ab = aa
ab.extend(bb)
ab.sort()
ab = list(dict.fromkeys(ab))
```

```
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'N({ab[i]}) = {N.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(5,4))
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-34.7679708826203\langle x\rangle^{0} - 6.12323399573677 \cdot 10^{-16}\langle x\rangle^{1} + 18.1346678798907\langle x -$$

'N(0) = -34.7679708826203 [kN]'

'N(4) = -16.6333030027296 [kN]'

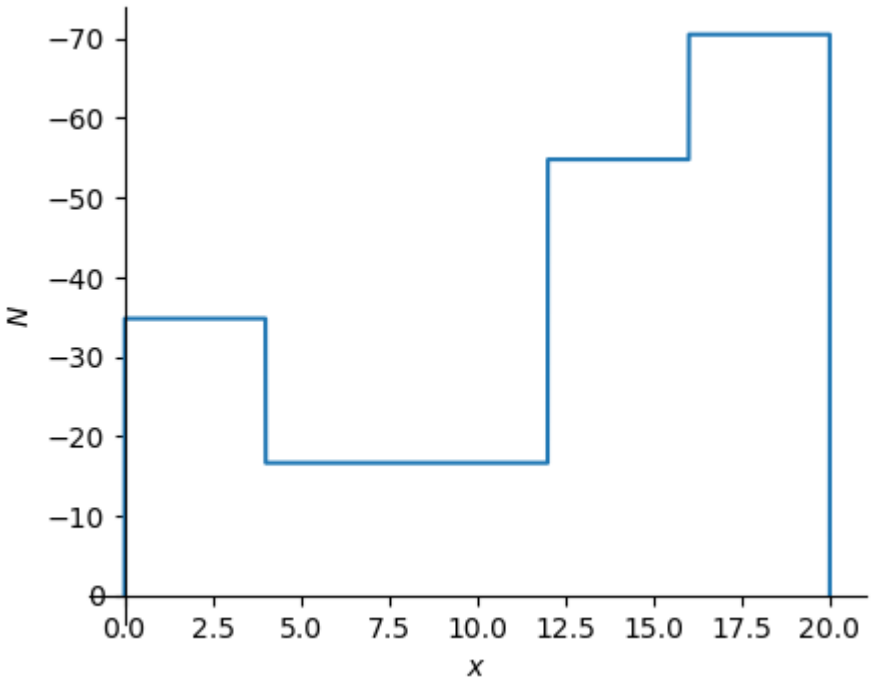
'N(8) = -16.6333030027296 [kN]'

'N(12) = -54.7679708826203 [kN]'

'N(16) = 2.94402054944811E-14 [kN]'

'N(16.000000000000004) = oo [kN]'

'N(20) = 2.94402054944811E-14 [kN]'




```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'V({ab[i]}) = {V.subs(x,ab[i]+dx).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(5,4))
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$23.3666969972704\langle x\rangle^{0} - 10.0\langle x\rangle^{1} + 51.40127388535\langle x-4\rangle^{0} - 10.0\langle x-4\rangle^{1} + 70.$$

```
'V(0) = 23.3666969972703 [kN]'
```

```
'V(4) = 34.7679708826203 [kN]'
```

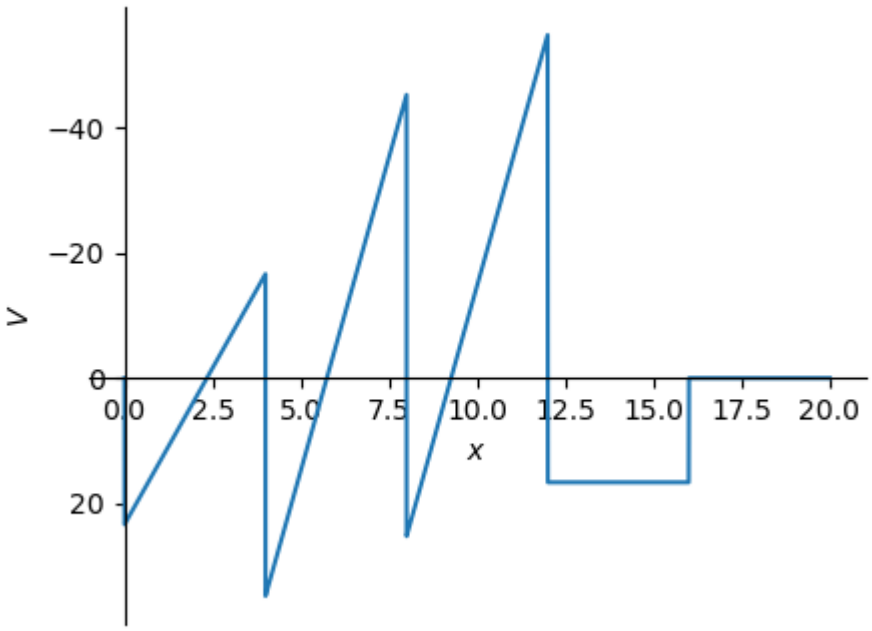
```
'V(8) = 25.2320291173797 [kN]'
```

```
'V(12) = 16.6333030027296 [kN]'
```

```
'V(16) = oo [kN]'
```

```
'V(16.000000000000004) = -1.05499772028045E-14 [kN]'
```

```
'V(20) = -5.59732895374049E-14 [kN]'
```




```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

for i in range(len(ab)):
    display(f'M({ab[i]}) = {M.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kNm]')

plt.figure(figsize=(5,4))
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

```
'M(0) = 0.0 [kNm]'
```

```
'M(4) = 13.4667879890814 [kNm]'
```

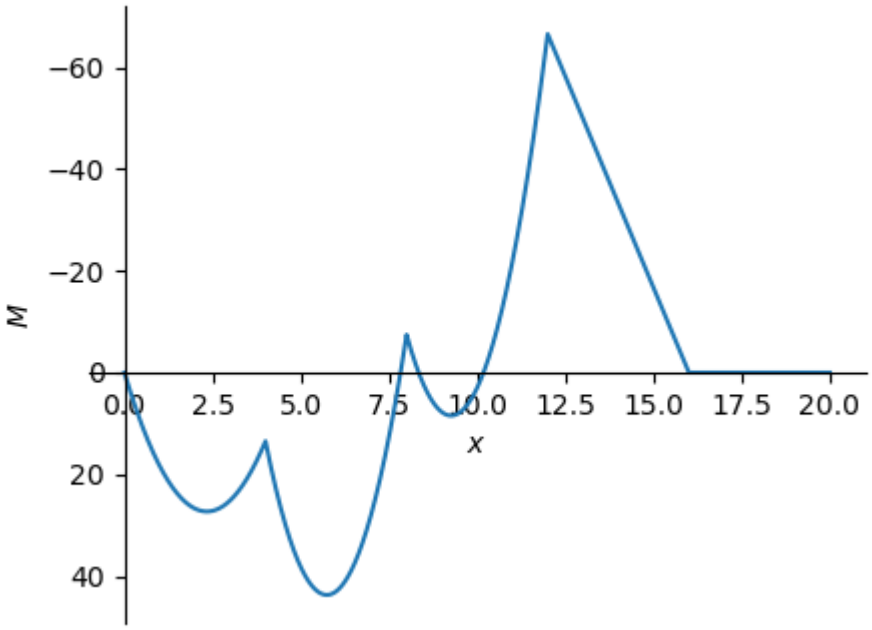
```
'M(8) = -7.46132848043729 [kNm]'
```

```
'M(12) = -66.5332120109185 [kNm]'
```

```
'M(16) = 5.68434188608080E-14 [kNm]'
```

```
'M(16.000000000000004) = oo [kNm]'
```

```
'M(20) = 1.27101121769055E-13 [kNm]'
```



```
v_np = sym.lambdify(x, v.subs(sol2).rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.subs(sol2).rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
for i in range(len(ab)):
    display(f'uv({ab[i]:.1f}) = {uv.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

for i in range(len(ab)):
    display(f'uh({ab[i]:.1f}) = {uh.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

plt.figure(figsize=(5,4))
plt.plot(h_np(x_np),v_np(x_np), marker='.',markersize=1, linewidth=0, color='black', labe
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), marker='.',markersize=0.5, li
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0.0) = 0.0000 [m]'
```

```
'uv(4.0) = 0.1391 [m]'
```

```
'uv(8.0) = 0.2819 [m]'
```

```
'uv(12.0) = 0.2191 [m]'
```

```
'uv(16.0) = 0.0000 [m]'
```

```
'uv(16.0) = 0.2819 [m]'
```

```
'uv(20.0) = 0.0000 [m]'
```

```
'uh(0.0) = 0.0000 [m]'
```

```
'uh(4.0) = 0.5865 [m]'
```

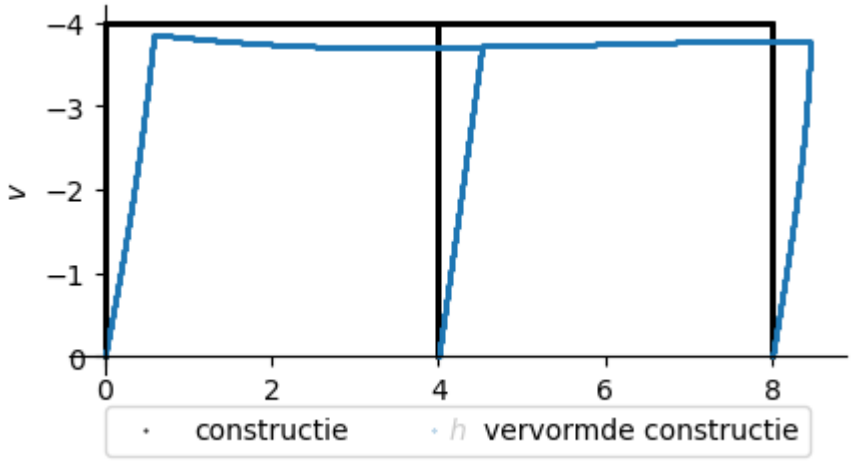
```
'uh(8.0) = 0.5200 [m]'
```

```
'uh(12.0) = 0.4535 [m]'
```

```
'uh(16.0) = 0.0000 [m]'
```

```
'uh(16.0) = 0.5200 [m]'
```

```
'uh(20.0) = 0.0000 [m]'
```



Example 6

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 2*10**-15

# gegevens constructie
a0, a1, a2, a3, a4, a5, a6 = 0, 2, 6, 7, 11, 12, 14
aa = [a0, a1, a2, a3, a4, a5, a6]
o0, o1, o2, o3, o4, o5 = 0, 0, -sym.pi/2, sym.pi, sym.pi/2, 0
oo = [o0, o1, o2, o3, o4, o5]

# sprongen in de x functie
phiBC, uzBC, uxBC = sym.symbols('phi^BC u_z^BC u_x^BC')
s1 = 12+2*dx
ss = [s1]

# gegevens belastingen
Fv, qv = 50, 10
RvA, RhA, RvF = sym.symbols('R_v^A R_h^A R_v^F')
VBE, HBE, TBE, VCD, HCD, TCD = sym.symbols('V^BE H^BE T^BE V^CD H^CD T^CD')
B = [RvA, RhA, qv, -qv, VBE, HBE, TBE, Fv, VCD, HCD, TCD, -VBE, -HBE, -TBE, -VCD, -HCD, -
b1, b2, b3, b4, b5, b6, b7 = 0, 2, 4, 6, 12, 12+2*dx, 14
bb = [b1, b1, b1, b4, b2, b2, b2, b3, b4, b4, b4, b5, b5, b5, b6, b6, b6, b6, b6, b6, b6,
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5, phi = 6, uz = 7, ux = 8
nn = [2, 3, 4, 4, 2, 3, 1, 2, 2, 3, 1, 2, 3, 1, 2, 3, 1, 6, 7, 8, 4, 2]
```

```
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 7:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-4) * EI
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$2EI\phi^{BC}\langle x-12.0\rangle^{-3} + EIu_z^{BC}\langle x-12.0\rangle^{-4} - H^{BE}\langle x-6\rangle^{-1} + H^{BE}\langle x-7\rangle^{-1} + I$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 8:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-2) * EA
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$EAu_x^{BC}\langle x-12.0\rangle^{-2} + H^{BE}\langle x-2\rangle^{-1} - H^{BE}\langle x-6\rangle^{-1} - H^{BE}\langle x-7\rangle^{-1} + H^{BE}\langle x$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - EAu_x^{BC}\langle x - 12.0\rangle^{-1} - H^{BE}\langle x - 2\rangle^{0} + H^{BE}\langle x - 6\rangle^{0} + H^{BE}\langle x - 7\rangle^{0} - H^{BE}$$

$$V =$$

$$C_V - 2EI\phi^{BC}\langle x - 12.0\rangle^{-2} - EIu_z^{BC}\langle x - 12.0\rangle^{-3} + H^{BE}\langle x - 6\rangle^{0} - H^{BE}\langle x - 7\rangle^{0}$$

$$M =$$

$$C_M + C_V x - 2EI\phi^{BC}\langle x - 12.0\rangle^{-1} - EIu_z^{BC}\langle x - 12.0\rangle^{-2} + H^{BE}\langle x - 6\rangle^{1} - H^{BE}\langle$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - EAu_x^{BC}\langle x - 12.0\rangle^{0} - H^{BE}\langle x - 2\rangle^{1} + H^{BE}\langle x - 6\rangle^{1} + H^{BE}\langle x - 7\rangle^{1}$$

$$\phi =$$

$$C_{\phi} + \frac{C_M x + \frac{C_V x^2}{2} - 2EI\phi^{BC}\langle x - 12.0\rangle^{0} - EIu_z^{BC}\langle x - 12.0\rangle^{-1} + \frac{H^{BE}\langle x-6\rangle^{2}}{2} - \frac{H^{E}}{}$$

$$u_z =$$

$$-C_{\phi} x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 2EI\phi^{BC}\langle x - 12.0\rangle^{1} - EIu_z^{BC}\langle x - 12.0\rangle^{0} + \frac{H^{BE}\langle x-}{6}$$

$$u_v =$$

$$C_{uz} + \left(-C_{\phi} x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - 2EI\phi^{BC}\langle x - 12.0\rangle^{1} - EIu_z^{BC}\langle x - 12.0\rangle^{0} + \frac{H^{BE}\langle}{}\right.$$

$$u_h =$$

$$C_{ux} - \left(-\frac{6C_N - 4H^{BE} - 6R_h^A}{EA} + \frac{C_N x - EAu_x^{BC}\langle x - 12.0\rangle^{0} - H^{BE}\langle x - 2\rangle^{1} + H\right.$$

```
# 3 reactiekrachten + 6 knoopkrachten + 3 vervormingssprongen + 6 integratieconstanten =
Eq1 = sym.Eq(N.subs(x,a0-dx),0)
Eq2 = sym.Eq(N.subs(x,a5+dx),0)
Eq3 = sym.Eq(N.subs(x,a6+dx),0)
Eq4 = sym.Eq(V.subs(x,0-dx),0)
Eq5 = sym.Eq(V.subs(x,a5+dx),0)
Eq6 = sym.Eq(V.subs(x,a6+dx),0)
Eq7 = sym.Eq(M.subs(x,a0-dx),0)
Eq8 = sym.Eq(M.subs(x,a5+dx),0)
Eq9 = sym.Eq(M.subs(x,a6+dx),0)
Eq10 = sym.Eq(phi.subs(x,a5+dx)-phi.subs(x,a1),0)
Eq11 = sym.Eq(phi.subs(x,s1+dx)-phi.subs(x,a2),0)
Eq12 = sym.Eq(uv.subs(x,0),0)
Eq13 = sym.Eq(uv.subs(x,a5)-uv.subs(x,a1),0)
Eq14 = sym.Eq(uv.subs(x,s1)-uv.subs(x,a2),0)
Eq15 = sym.Eq(uv.subs(x,a6),0)
Eq16 = sym.Eq(uh.subs(x,0),0)
Eq17 = sym.Eq(uh.subs(x,a5)-uh.subs(x,a1),0)
Eq18 = sym.Eq(uh.subs(x,s1)-uh.subs(x,a2),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11,Eq12,Eq13,Eq14,Eq15,Eq16,E
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: (-4.40975360000001e+45*EA**2 - 3.08877312e+46*EA*EI - 3.2587776e+45*EI**2)/(2.795
 C_ux: 0.0,
 C_uz: 0.0,
 H^BE: 5.696e+31*EA/(4.16e+29*EA + 1.536e+30*EI),
 H^CD: 0.0,
 R_h^A: 0.0,
 R_v^A: -65.0000000000000,
 R_v^F: -65.0000000000000,
 T^BE: (4.465664e+33*EA**2 + 8.133888e+33*EA*EI + 8.20223999999999e+32*EI**2)/(3.4944e+31
 T^CD: 110.000000000000,
 V^BE: -3.69482222595252e+15*EA/(8.96e+29*EA + 9.6e+28*EI),
 V^CD: -45.0000000000000,
 phi^BC: (-1.7088e+31*EA - 2.73408e+32*EI)/(4.16e+29*EA*EI + 1.536e+30*EI**2),
 u_x^BC: (2.848e+58*EA + 1.09139364212751e+43*EI)/(5.2e+55*EA**2 + 1.92e+56*EA*EI),
 u_z^BC: (-9.14720432611493e+29*EA - 9.82168933355715e+28*EI)/(1.792e+41*EA*EI + 1.92e+40
```

```
#display(sym.simplify(uz.subs(sol).rewrite(sym.Piecewise)))
ea = 10**3
ei = 3*10**3

display(f'{RvA} = {RvA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhA} = {RhA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvF} = {RvF.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{VBE} = {VBE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HBE} = {HBE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{TBE} = {TBE.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{VCD} = {VCD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HCD} = {HCD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{TCD} = {TCD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{phiBC} = {phiBC.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uzBC} = {uzBC.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{uxBC} = {uxBC.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
```

```
'R_v^A = -65.00'
```

```
'R_h^A = 0.00'
```

```
'R_v^F = -65.00'
```

```
'V^BE = -0.00'
```

```
'H^BE = 11.34'
```

```
'T^BE = 65.00'
```

```
'V^CD = -45.00'
```

```
'H^CD = 0.00'
```

```
'T^CD = 110.00'
```

```
'phi^BC = -0.0556'
```

```
'u_z^BC = -0.0000'
```

```
'u_x^BC = 0.0454'
```

```
'C_N = 0.00'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = -0.0944'
```

```
'C_ux = 0.00'
```

```
'C_uz = 0.00'
```

```
vBE, hBE = sym.symbols('v^BE h^BE')
vv = [vBE]
hh = [hBE]
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
for i in range(len(ss)):
    v += vv[i] * sym.SingularityFunction(x,ss[i],0)
    h += hh[i] * sym.SingularityFunction(x,ss[i],0)

# 1 x 2 sprongen = 2 onbekenden
Eq1 = sym.Eq(v.subs(x,a2),v.subs(x,s1))
Eq2 = sym.Eq(h.subs(x,a2),h.subs(x,s1))

sol2 = sym.solve((Eq1,Eq2),(vBE, hBE))
display(sol2)
```

```
{h^BE: 4.00000000000000, v^BE: 0.0}
```

```
L = aa[-1]
x_np = np.linspace(0-dx,L+dx,10000)
ab = aa
ab.extend(bb)
ab.sort()
ab = list(dict.fromkeys(ab))
```

```
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'N({ab[i]}) = {N.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(10,4))
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-11.3375796178344\langle x-2\rangle^{0}+11.3375796178344\langle x-6\rangle^{0}+11.3375796178344\langle x$$

```
'N(0) = 0 [kN]'
```

```
'N(2) = -11.3375796178344 [kN]'
```

```
'N(4) = -11.3375796178344 [kN]'
```

```
'N(6) = 3.12062688002747E-15 [kN]'
```

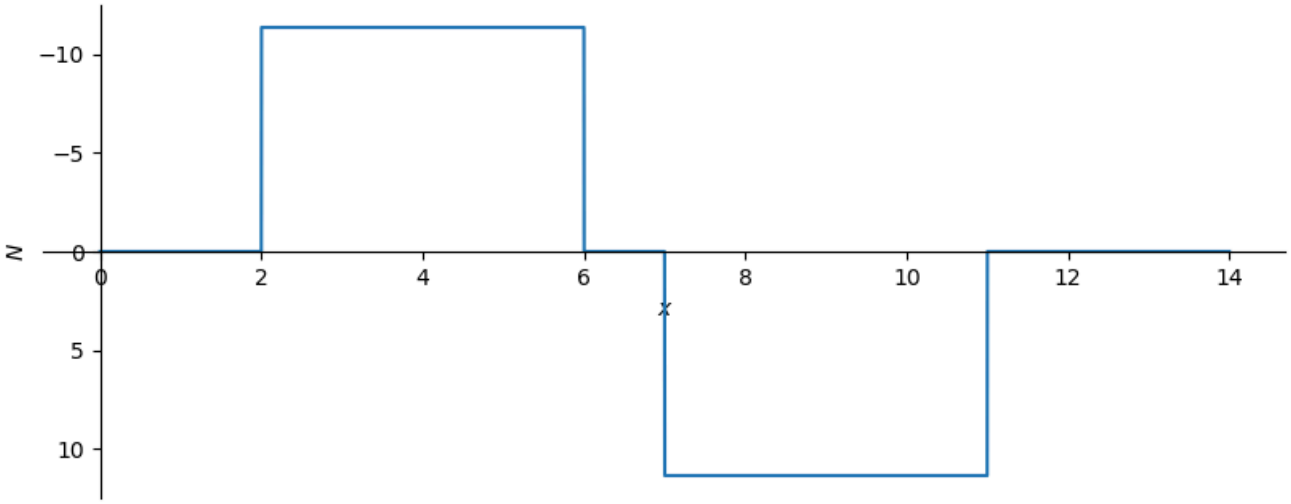
```
'N(7) = 11.3375796178344 [kN]'
```

```
'N(11) = -3.12062688002747E-15 [kN]'
```

```
'N(12) = 0 [kN]'
```

'N(12.000000000000004) = -oo [kN]'

'N(14) = 0 [kN]'




```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'V({ab[i]}) = {V.subs(x,ab[i]+dx).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(10,4))
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$65.0\langle x\rangle^{0} - 10\langle x\rangle^{1} - 65.0021231422505\langle x-2\rangle^{-1} + 3.12062688002747\cdot 10^{-15}\langle x -$$

'V(0) = 65.0000000000000 [kN]'

'V(2) = 45.0000000000000 [kN]'

'V(4) = -25.0000000000000 [kN]'

'V(6) = 11.3375796178344 [kN]'

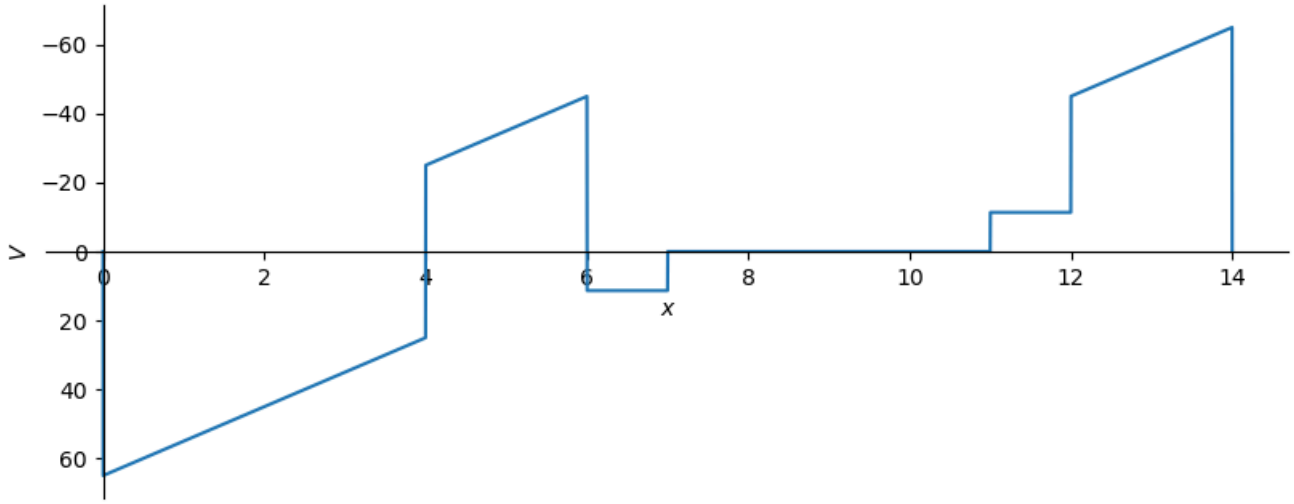
'V(7) = -3.12062688002747E-15 [kN]'

'V(11) = -11.3375796178344 [kN]'

'V(12) = 0 [kN]'

'V(12.000000000000004) = -45.0000000000000 [kN]'

'V(14) = 0 [kN]'




```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

for i in range(len(ab)):
    display(f'M({ab[i]}) = {M.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kNm]')

plt.figure(figsize=(10,4))
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

'M(0) = 0.0 [kNm]'

'M(2) = 44.9978768577495 [kNm]'

'M(4) = 114.997876857749 [kNm]'

'M(6) = -65.0021231422505 [kNm]'

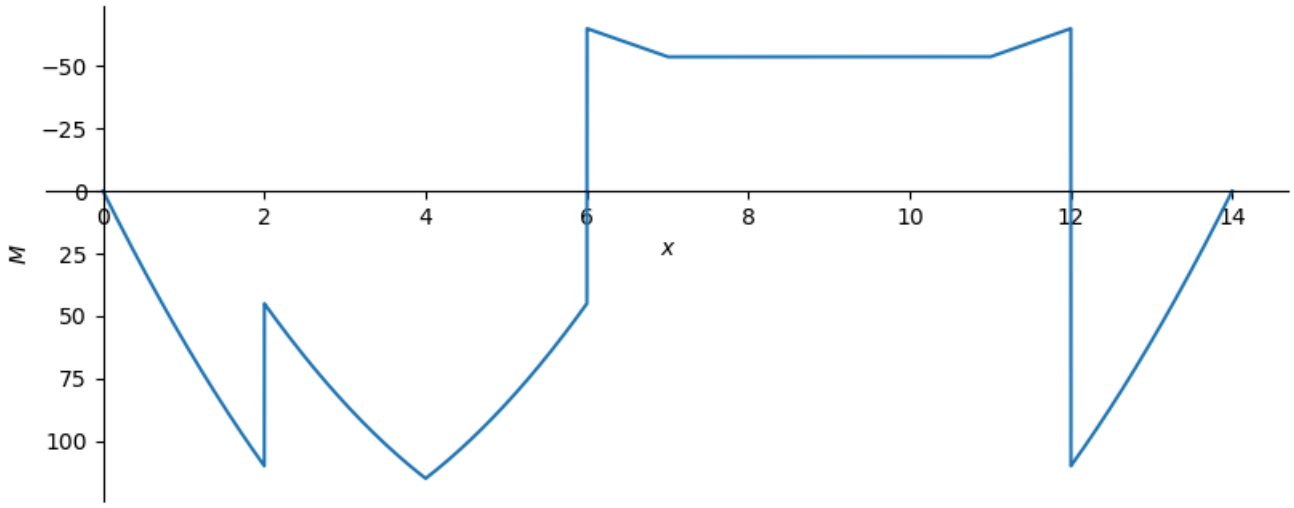
'M(7) = -53.6645435244161 [kNm]'

'M(11) = -53.6645435244161 [kNm]'

'M(12) = 2.84217094304040E-14 [kNm]'

'M(12.000000000000004) = oo [kNm]'

'M(14) = 5.68434188608080E-14 [kNm]'



```
v_np = sym.lambdify(x, v.subs(sol2).rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.subs(sol2).rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
for i in range(len(ab)):
    display(f'uv({ab[i]:.1f}) = {uv.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

for i in range(len(ab)):
    display(f'uh({ab[i]:.1f}) = {uh.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

plt.figure(figsize=(6,6))
plt.plot(h_np(x_np),v_np(x_np), marker='.',markersize=1, linewidth=0, color='black', labe
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), marker='.',markersize=0.5, li
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0.0) = 0.0000 [m]'
```

```
'uv(2.0) = 0.1622 [m]'
```

```
'uv(4.0) = 0.2256 [m]'
```

```
'uv(6.0) = 0.1622 [m]'
```

```
'uv(7.0) = 0.1622 [m]'
```

```
'uv(11.0) = 0.1622 [m]'
```

```
'uv(12.0) = 0.1622 [m]'
```

```
'uv(12.0) = 0.1622 [m]'
```

'uv(14.0) = 0.0000 [m]'

'uh(0.0) = 0.0000 [m]'

'uh(2.0) = 0.0000 [m]'

'uh(4.0) = -0.0227 [m]'

'uh(6.0) = -0.0454 [m]'

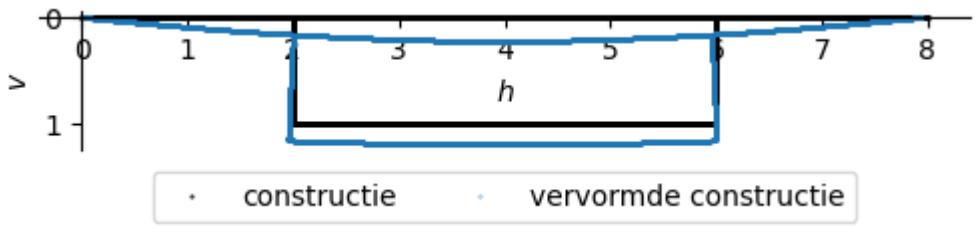
'uh(7.0) = -0.0000 [m]'

'uh(11.0) = -0.0454 [m]'

'uh(12.0) = -0.0000 [m]'

'uh(12.0) = -0.0454 [m]'

'uh(14.0) = -0.0454 [m]'



Example 7

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

```
# algemene gegevens
x = sym.symbols('x')
EI, EA = sym.symbols('EI EA')
CV, CM, Cphi, Cuz, CN, Cux = sym.symbols('C_V C_M C_phi C_uz C_N C_ux')
dx = 2*10**-15

# gegevens constructie
a0, a1, a2, a3, a4, a5 = 0, 3, 7, 12, 16, 19
aa = [a0, a1, a2, a3, a4, a5]
o0, o1, o2, o3, o4 = -sym.pi, sym.pi/2, sym.atan(-4/3), sym.pi/2, -sym.pi
oo = [o0, o1, o2, o3, o4]

# sprongen in de x functie
#= sym.symbols('phi^ij u_v^ij u_h^ij')
#s1 =
ss = []

# gegevens belastingen
Fv, Fh = 40, -30
RvA, RvB, RhB = sym.symbols('R_v^A R_v^B R_h^B')
phisA, phisCB, phisBD, phisD = sym.symbols('phi_s^A phi_s^CB phi_s^BD phi_s^D')
VBC, HBC, VBD, HBD, VCD, HCD = sym.symbols('V^BC H^BC V^BD H^BD V^CD H^CD')
B = [RvB, RhB, VBC, HBC, VBD, HBD, RvA, phisA, Fv, VCD, HCD, phisCB, -VBC, -HBC, -VBD, -H
b1, b2, b3, b4, b5, b6, b7, b8 = 0, 3, 7, 7+2*dx, 12, 12+2*dx, 16, 19
bb = [b1, b1, b1, b1, b1, b1, b2, b2, b3, b3, b3, b4, b5, b5, b6, b6, b6, b7, b7, b8, b8]
# K = 1, Fv = 2, Fh = 3, qv = 4, qh = 5, phi = 6, uz = 7, ux = 8
nn = [2, 3, 2, 3, 2, 3, 2, 6, 2, 2, 3, 6, 2, 3, 2, 3, 6, 3, 6, 2, 3]
```

```
#qz opstellen
qz = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 1:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[-1])
            if nn[i] == 4:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            if nn[i] == 5:
                qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 1:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-2)
                if nn[i] == 2:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 3:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.sin(oo[j-1])
                if nn[i] == 4:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 5:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.sin(oo[j-1])
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 6:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-3) * EI
                if nn[i] == 7:
                    qz += B[i] * sym.SingularityFunction(x,bb[i],-4) * EI
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 3:
                qz += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.sin(oo[j]) - sym.
            if nn[i] == 4:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qz += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qz)
```

$$EI\phi_s^A\langle x-3\rangle^{-3} + EI\phi_s^{BD}\langle x-12.0\rangle^{-3} + EI\phi_s^{CB}\langle x-7.0\rangle^{-3} + EI\phi_s^D\langle x-16\rangle^{-3} + .$$

```
#qx opstellen
qx = 0

#beginpunten
for i in range(len(B)):
    for j in range(len(aa)):
        if bb[i] == aa[-1]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[-1])
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[-1])
            if nn[i] == 4:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[-1])
            if nn[i] == 5:
                qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[-1])
            break
        else:
            if bb[i] < aa[j]:
                if nn[i] == 2:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * -sym.sin(oo[j-1])
                if nn[i] == 3:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-1) * sym.cos(oo[j-1])
                if nn[i] == 4:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * -sym.sin(oo[j-1])
                if nn[i] == 5:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],0) * sym.cos(oo[j-1])
                if nn[i] == 8:
                    qx += B[i] * sym.SingularityFunction(x,bb[i],-2) * EA
                break

# knikpunten
for i in range(len(B)):
    for j in range(len(aa)-1):
        if bb[i] < aa[j]:
            if nn[i] == 2:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (-sym.sin(oo[j]) + sym
            if nn[i] == 3:
                qx += B[i] * sym.SingularityFunction(x,aa[j],-1) * (sym.cos(oo[j]) - sym.
            if nn[i] == 4:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti
            if nn[i] == 5:
                qx += B[i] * ((sym.SingularityFunction(x,aa[j],0) + sym.SingularityFuncti

display(qx)
```

$$-H^{BC}\langle x\rangle^{-1} + H^{BC}\langle x-3\rangle^{-1} + 0.6H^{BC}\langle x-7\rangle^{-1} - 0.6H^{BC}\langle x-12\rangle^{-1} - H^{BD}\langle x$$

```
V = -sym.integrate(qz.expand(), x) + CV
M = sym.integrate(V, x) + CM
kappa = M / EI
phi = sym.integrate(kappa, x) + Cphi
uz = -sym.integrate(phi, x) + Cuz

N = -sym.integrate(qx.expand(), x) + CN
epsilon = N / EA
ux = sym.integrate(epsilon, x) + Cux

uvz = uz.subs(x,0) * sym.cos(o0)
uvx = -ux.subs(x,0) * sym.sin(o0)
for i in range(len(oo)):
    uvz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uvx += -((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs
uv = uvz + uvx

uhz = uz.subs(x,0) * sym.sin(o0)
uhx = ux.subs(x,0) * sym.cos(o0)
for i in range(len(oo)):
    uhz += ((uz - uz.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (uz - uz.subs(
    uhx += ((ux - ux.subs(x,aa[i])) * sym.SingularityFunction(x,aa[i],0) - (ux - ux.subs(
uh = uhz + uhx

display(sym.symbols('{N}='), N)
display(sym.symbols('{V}='), V)
display(sym.symbols('{M}='), M)
display(sym.symbols('{u_x}='), ux)
display(sym.symbols('{\phi}='), phi)
display(sym.symbols('{u_z}='), uz)
display(sym.symbols('{u_v}='), uv)
display(sym.symbols('{u_h}='), uh)
```

$$N =$$

$$C_N - EAu_x^{BD}\langle x - 7.0\rangle^{-1} - EAu_x^{CF}\langle x - 12.0\rangle^{-1} - 0.8H^{BF}\langle x - 2\rangle^{0} + 0.8H^{BF}\langle x -$$

$$V =$$

$$C_V - EI\phi^{BD}\langle x - 7.0\rangle^{-2} - EI\phi^{CF}\langle x - 12.0\rangle^{-2} - EIu_z^{BD}\langle x - 7.0\rangle^{-3} - EIu_z^{CF}\langle x -$$

$$M =$$

$$C_M + C_V x - EI\phi^{BD}\langle x - 7.0\rangle^{-1} - EI\phi^{CF}\langle x - 12.0\rangle^{-1} - EIu_z^{BD}\langle x - 7.0\rangle^{-2} - EI$$

$$u_x =$$

$$C_{ux} + \frac{C_N x - EAu_x^{BD}\langle x - 7.0\rangle^{0} - EAu_x^{CF}\langle x - 12.0\rangle^{0} - 0.8H^{BF}\langle x - 2\rangle^{1} + 0.8H^{l}$$

$$\phi =$$

$$C_{\phi} + \frac{C_M x + \frac{C_V x^2}{2} - EI\phi^{BD}\langle x - 7.0\rangle^{0} - EI\phi^{CF}\langle x - 12.0\rangle^{0} - EIu_z^{BD}\langle x - 7.0\rangle^{-1}$$

$$u_z =$$

$$-C_{\phi} x + C_{uz} - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi^{BD}\langle x - 7.0\rangle^{1} - EI\phi^{CF}\langle x - 12.0\rangle^{1} - EIu_z^{BD}\langle x$$

$$u_v =$$

$$C_{uz} + \left(-C_{\phi} x - \frac{\frac{C_M x^2}{2} + \frac{C_V x^3}{6} - EI\phi^{BD}\langle x - 7.0\rangle^{1} - EI\phi^{CF}\langle x - 12.0\rangle^{1} - EIu_z^{BD}\right.$$

$$u_h =$$

$$C_{ux} - 0.2\left(-\frac{2C_N - 2R_h}{EA} + \frac{C_N x - EAu_x^{BD}\langle x - 7.0\rangle^{0} - EAu_x^{CF}\langle x - 12.0\rangle^{0} - 0.8\right.$$

```
# 3 reactiekrachten + 6 knoopkrachten + 4 scharnieren + 6 integratieconstanten = 19 voorw
Eq1 = sym.Eq(N.subs(x,a0-dx),0)
Eq2 = sym.Eq(N.subs(x,a3+dx),0)
Eq3 = sym.Eq(N.subs(x,a5+dx),0)
Eq4 = sym.Eq(V.subs(x,0-dx),0)
Eq5 = sym.Eq(V.subs(x,a3+dx),0)
Eq6 = sym.Eq(V.subs(x,a5+dx),0)
Eq7 = sym.Eq(M.subs(x,a0-dx),0)
Eq8 = sym.Eq(M.subs(x,a1+dx),0)
Eq9 = sym.Eq(M.subs(x,a2+dx),0)
Eq10 = sym.Eq(M.subs(x,a3+dx),0)
Eq11 = sym.Eq(M.subs(x,a4+dx),0)
Eq12 = sym.Eq(M.subs(x,a5+dx),0)
Eq13 = sym.Eq(uv.subs(x,a0),0)
Eq14 = sym.Eq(uv.subs(x,a1),0)
Eq15 = sym.Eq(uv.subs(x,a3)-uv.subs(x,a0),0)
Eq16 = sym.Eq(uv.subs(x,a5)-uv.subs(x,a2),0)
Eq17 = sym.Eq(uh.subs(x,a0),0)
Eq18 = sym.Eq(uh.subs(x,a3)-uh.subs(x,a0),0)
Eq19 = sym.Eq(uh.subs(x,a5)-uh.subs(x,a2),0)
```

```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11,Eq12,Eq13,Eq14,Eq15,Eq16,E
display(sol)
```

```
{C_M: 0.0,
 C_N: 0.0,
 C_V: 0.0,
 C_phi: -5.25486924121805e-45/EI,
 C_uh: 0.0,
 C_uv: 0.0,
 H^BC: -30.0000000000000,
 H^BD: -4.21884749357561e-14,
 H^CD: -30.0000000000000,
 R_h^B: 30.0000000000000,
 R_v^A: -80.0000000000001,
 R_v^B: 40.0000000000001,
 V^BC: -40.0000000000000,
 V^BD: -9.53311503811469e-14,
 V^CD: 9.53311503811469e-14,
 phi_s^A: 1.25e-201*(-1.00974195868289e+172*EA - 1.68666666666666e+203*EI)/(EA*EI),
 phi_s^BD: 5.62500000000001e-201*(1.47371826676167e+187*EA - 1.06666666666666e+202*EI)/(E
 phi_s^CB: 3.12500000000001e-175*(1.42108547152019e+161*EA + 1.2e+176*EI)/(EA*EI),
 phi_s^D: 6.25000000000001e-202*(-2.37794968901048e+188*EA + 2.02666666666666e+203*EI)/(E
```

```
ea = 10**3
ei = 3*10**3

display(f'{RvA} = {RvA.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RvB} = {RvB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{RhB} = {RhB.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{VBC} = {VBC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HBC} = {HBC.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{VBD} = {VBD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HBD} = {HBD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{VCD} = {VCD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{HCD} = {HCD.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')

display(f'{phisA} = {phisA.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{phisCB} = {phisCB.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{phisBD} = {phisBD.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{phisD} = {phisD.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')

display(f'{CN} = {CN.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CV} = {CV.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{CM} = {CM.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cphi} = {Cphi.subs(sol).subs(EI,ei).subs(EA,ea):.4f}')
display(f'{Cux} = {Cux.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
display(f'{Cuz} = {Cuz.subs(sol).subs(EI,ei).subs(EA,ea):.2f}')
```

```
'R_v^A = -80.00'
```

```
'R_v^B = 40.00'
```

```
'R_h^B = 30.00'
```

```
'V^BC = -40.00'
```

```
'H^BC = -30.00'
```

```
'V^BD = -0.00'
```

```
'H^BD = -0.00'
```

```
'V^CD = 0.00'
```

```
'H^CD = -30.00'
```

```
'phi_s^A = -0.2108'
```

```
'phi_s^CB = 0.0375'
```

```
'phi_s^BD = -0.0600'
```

```
'phi_s^D = 0.1267'
```

```
'C_V = 0.00'
```

```
'C_M = 0.00'
```

```
'C_phi = -0.0000'
```

```
'C_uv = 0.00'
```

```
'C_uh = 0.00'
```

```
'C_N = 0.00'
```

```
vBE, hBE = sym.symbols('v^BE h^BE')
vv = [vBE]
hh = [hBE]
v = 0
h = 0
for i in range(len(oo)):
    v += -(sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * s
    h += (sym.SingularityFunction(x,aa[i],1) - sym.SingularityFunction(x,aa[i+1],1)) * sy
for i in range(len(ss)):
    v += vv[i] * sym.SingularityFunction(x,ss[i],0)
    h += hh[i] * sym.SingularityFunction(x,ss[i],0)

# 0 X 0 sprongen = 0 onbekenden
```

```
L = aa[-1]
x_np = np.linspace(0-dx,L+dx,10000)
ab = aa
ab.extend(bb)
ab.sort()
ab = list(dict.fromkeys(ab))
```

```
N_np = sym.lambdify(x, N.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(N.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'N({ab[i]}) = {N.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(10,4))
plt.plot(x_np,N_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$N$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-80.0000000000001\langle x-3\rangle^{0}+130.0\langle x-7\rangle^{0}-49.9999999999999\langle x-12\rangle^{0}+9.53$$

```
'N(0) = 0 [kN]'
```

```
'N(3) = -80.0000000000001 [kN]'
```

```
'N(7) = 49.9999999999999 [kN]'
```

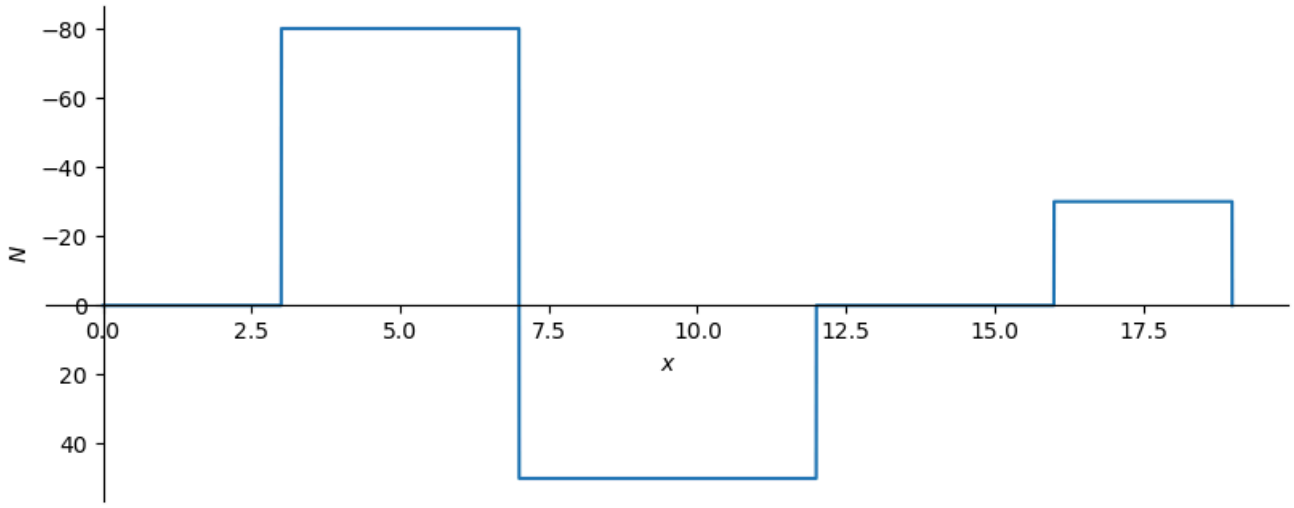
```
'N(7.000000000000004) = 49.9999999999999 [kN]'
```

```
'N(12) = -1.26217744835362E-29 [kN]'
```

```
'N(12.000000000000004) = 9.53311503811469E-14 [kN]'
```

```
'N(16) = -30.0000000000000 [kN]'
```

'N(19) = 3.55271367880049E-15 [kN]'




```
V_np = sym.lambdify(x, V.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
display(V.subs(sol).subs(EI,ei).subs(EA,ea))

for i in range(len(ab)):
    display(f'V({ab[i]}) = {V.subs(x,ab[i]+dx).subs(sol).subs(EI,ei).subs(EA,ea)} [kN]')

plt.figure(figsize=(10,4))
plt.plot(x_np,V_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$V$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

$$-2.96059473233392 \cdot 10^{-15}\langle x\rangle^{0} + 632.5\langle x-3\rangle^{-2} + 2.96059473233392 \cdot 10^{-15}\langle x -$$

'V(0) = -2.96059473233392E-15 [kN]'

'V(3) = 0 [kN]'

'V(7) = 7.10542735760100E-15 [kN]'

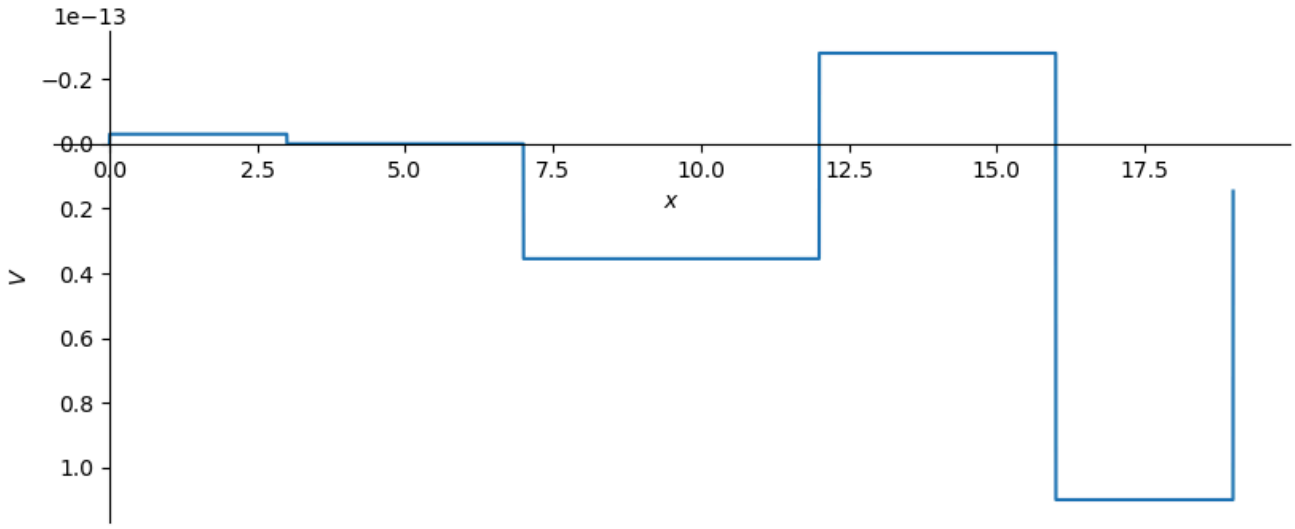
'V(7.000000000000004) = 7.10542735760100E-15 [kN]'

```
'V(12) = -3.55271367880050E-15 [kN]'
```

```
'V(12.000000000000004) = -4.61852778244065E-14 [kN]'
```

```
'V(16) = 9.53311503811469E-14 [kN]'
```

```
'V(19) = 0 [kN]'
```




```
M_np = sym.lambdify(x, M.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

for i in range(len(ab)):
    display(f'M({ab[i]}) = {M.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea)} [kNm]')

plt.figure(figsize=(10,4))
plt.plot(x_np,M_np(x_np))
plt.xlabel('$x$')
plt.ylabel('$M$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
```

```
'M(0) = 0.0 [kNm]'
```

```
'M(3) = oo [kNm]'
```

```
'M(7) = -1.77635683940076E-15 [kNm]'
```

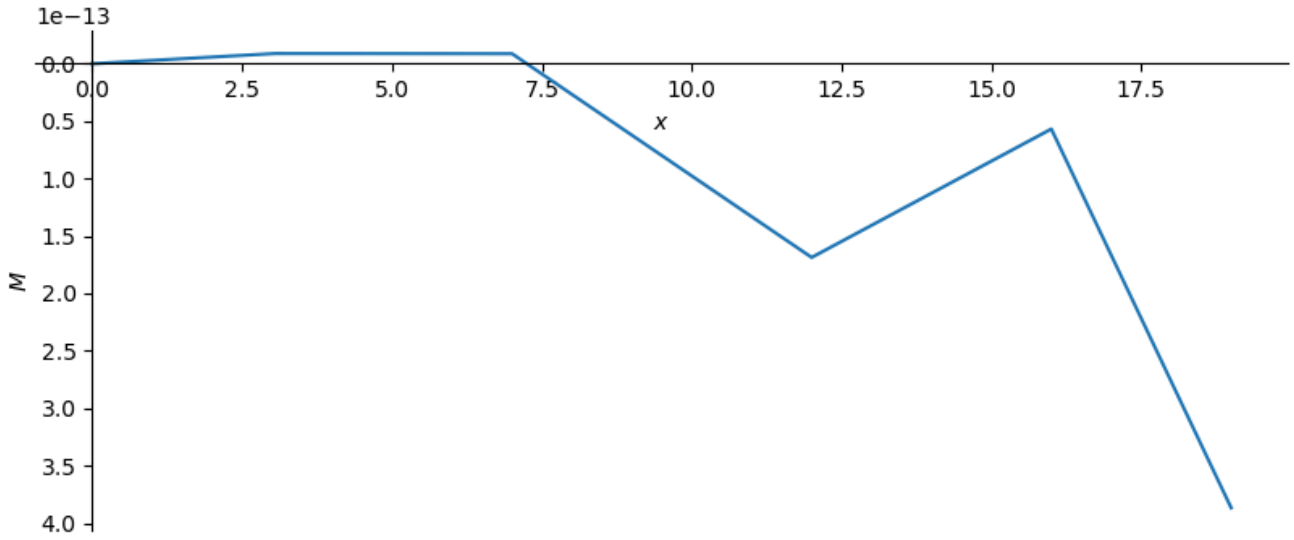
```
'M(7.000000000000004) = -oo [kNm]'
```

```
'M(12) = 8.34887714518113E-14 [kNm]'
```

```
'M(12.000000000000004) = oo [kNm]'
```

```
'M(16) = -oo [kNm]'
```

```
'M(19) = 8.52651282912120E-14 [kNm]'
```



```
v_np = sym.lambdify(x, v.rewrite(sym.Piecewise))
h_np = sym.lambdify(x, h.rewrite(sym.Piecewise))

uv_np = sym.lambdify(x, uv.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))
uh_np = sym.lambdify(x, uh.subs(sol).subs(EI,ei).subs(EA,ea).rewrite(sym.Piecewise))

#display(uv.subs(sol))
for i in range(len(ab)):
    display(f'uv({ab[i]:.1f}) = {uv.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

for i in range(len(ab)):
    display(f'uh({ab[i]:.1f}) = {uh.subs(x,ab[i]).subs(sol).subs(EI,ei).subs(EA,ea):.4f}

plt.figure(figsize=(6,6))
plt.plot(h_np(x_np),v_np(x_np), marker='.',markersize=1, linewidth=0, color='black', labe
plt.plot((h_np(x_np)+uh_np(x_np)),(v_np(x_np)+uv_np(x_np)), marker='.',markersize=0.5, li
plt.xlabel('$h$')
plt.ylabel('$v$');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.invert_yaxis()
plt.axis('scaled')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=False, nc
```

```
'uv(0.0) = 0.0000 [m]'
```

```
'uv(3.0) = -0.0000 [m]'
```

```
'uv(7.0) = 0.3200 [m]'
```


```
'uv(7.0) = 0.3200 [m]'
```


```
'uv(12.0) = 0.0000 [m]'
```


```
'uv(12.0) = 0.0000 [m]'
```


```
'uv(16.0) = -0.0000 [m]'
```

```
'uv(19.0) = 0.3200 [m]'
```

```
'uh(0.0) = 0.0000 [m]'
```

```
'uh(3.0) = 0.0000 [m]'
```

```
'uh(7.0) = -0.8433 [m]'
```

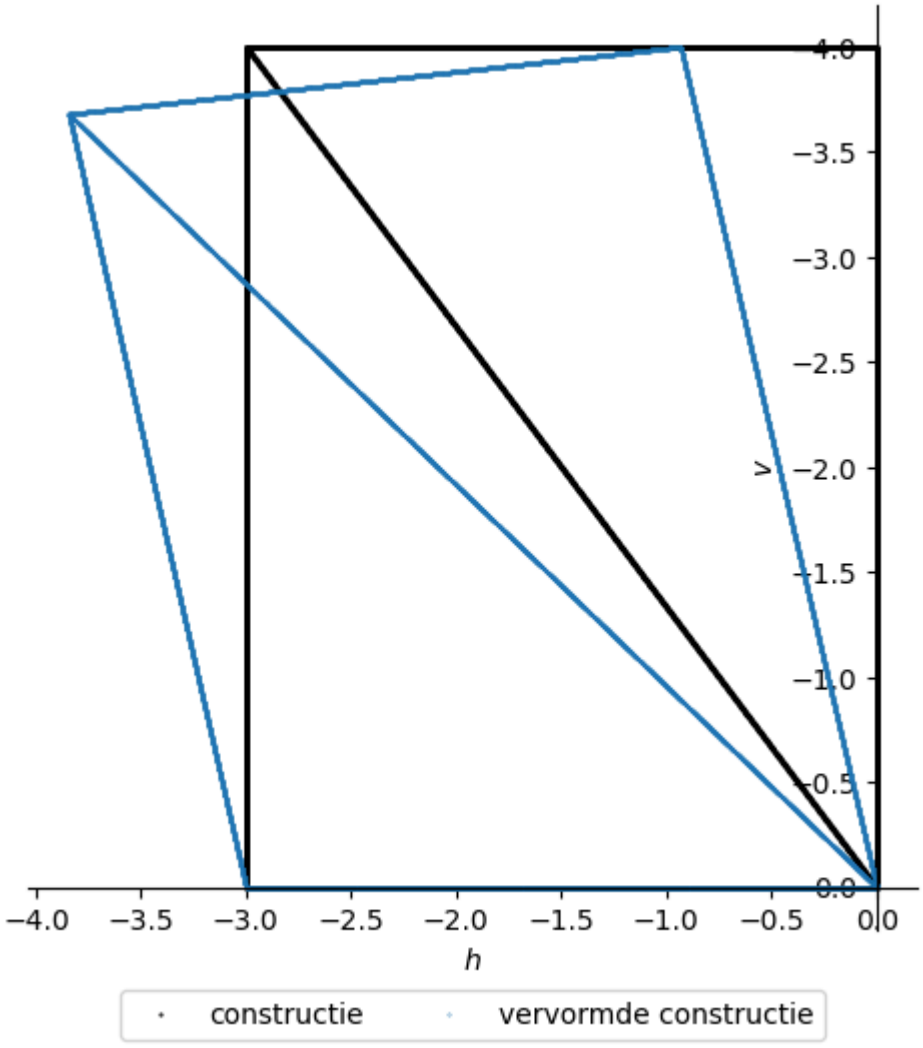
```
'uh(7.0) = -0.8433 [m]'
```

```
'uh(12.0) = 0.0000 [m]'
```

```
'uh(12.0) = -0.0000 [m]'
```

```
'uh(16.0) = -0.9333 [m]'
```

```
'uh(19.0) = -0.8433 [m]'
```


4.0
−3.5
−3.0
−2.5
v
−2.0
−1.5
−1.0
−0.5
0.0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
h
constructie
vervormde constructie

Mark: Hinges and influence lines in SymPy

Contents

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.

van Gelder (2024)

Documenten

- TU Delft Education repository
- GitHub repository old code, also shown in this book
- GitHub repository new code, also shown in this book
- GitHub respoitory other example, also shown in this book

Equations 1

```
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(10, E, I)
r0 = b.apply_support(0, type="pin")
r10 = b.apply_support(10, type="pin")
b.solve_for_ild_reactions(-1, r0, r10)
```

```
b.ild_reactions
```

```
{R_0: SingularityFunction(a, 0, 0) - SingularityFunction(a, 0, 1)/10 + SingularityFunctio
 R_10: SingularityFunction(a, 0, 1)/10 - SingularityFunction(a, 10, 0) - SingularityFunct
```

```
b.ild_reactions[r0]
```

$$\langle a \rangle^{0} - \frac{\langle a \rangle^{1}}{10} + \frac{\langle a - 10 \rangle^{1}}{10}$$

```
b.ild_reactions[r10]
```

$$\frac{\langle a \rangle^{1}}{10} - \langle a - 10 \rangle^{0} - \frac{\langle a - 10 \rangle^{1}}{10}$$

Example 1

```
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
F = sm.Symbol('F')
x = sm.Symbol('x')
b = Beam(10, E, I)
r0, m0 = b.apply_support(0, type="fixed")
r10, m10 = b.apply_support(10, type="fixed")
b.apply_rotation_hinge(5)
b.apply_load(-F, 5, -1)
```

```
b.load
```

$$EIP_5\langle x-5\rangle^{-3} - F\langle x-5\rangle^{-1} + M_0\langle x\rangle^{-2} + M_{10}\langle x-10\rangle^{-2} + R_0\langle x\rangle^{-1} + R_{10}\langle x-10$$

```
b.bending_moment()
```

$$-EIP_5\langle x-5\rangle^{-1} + F\langle x-5\rangle^{1} - M_0\langle x\rangle^{0} - M_{10}\langle x-10\rangle^{0} - R_0\langle x\rangle^{1} - R_{10}\langle x-10\rangle^{1}$$

```
b.bending_moment().subs(x, 5)
```

$$-\infty EIP_5 - M_0 - 5R_0$$

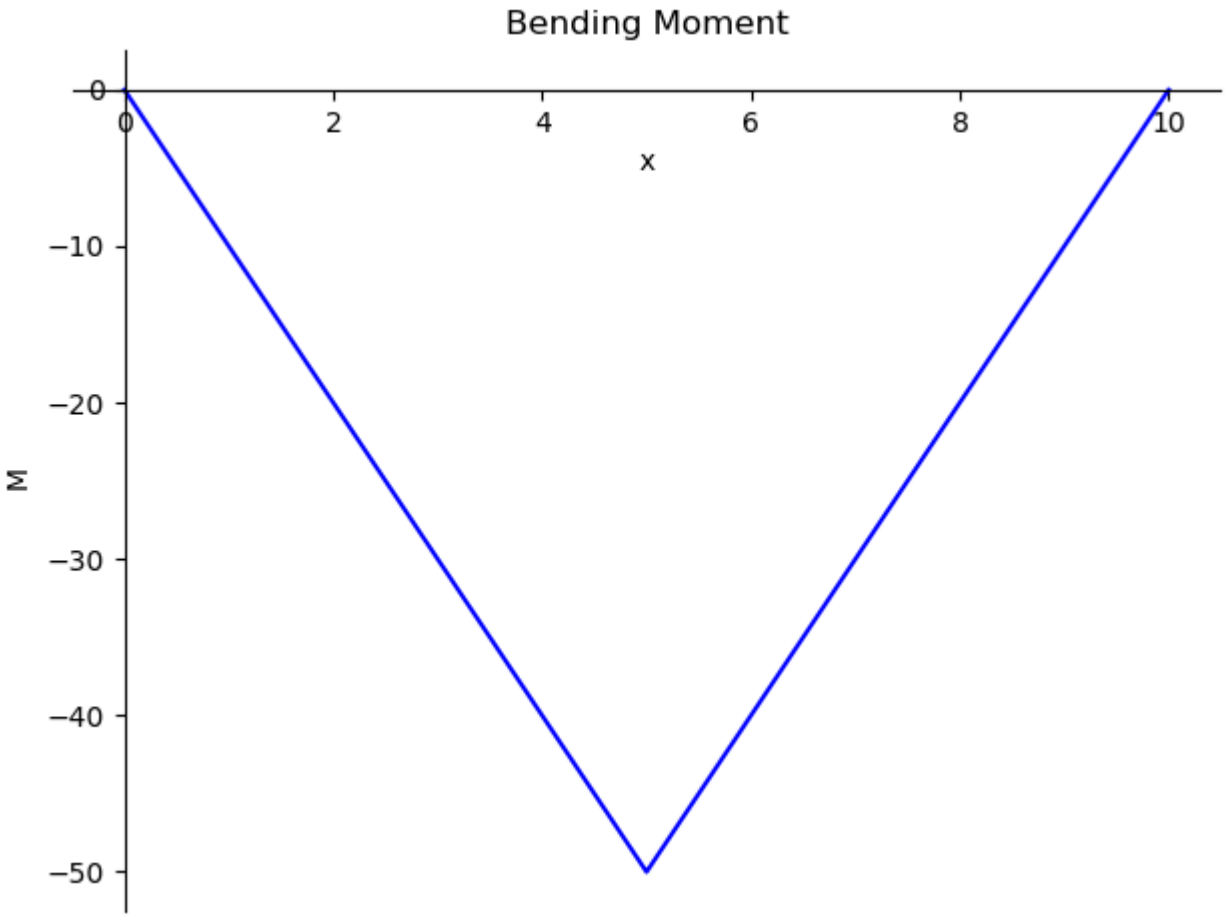
Example 2

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

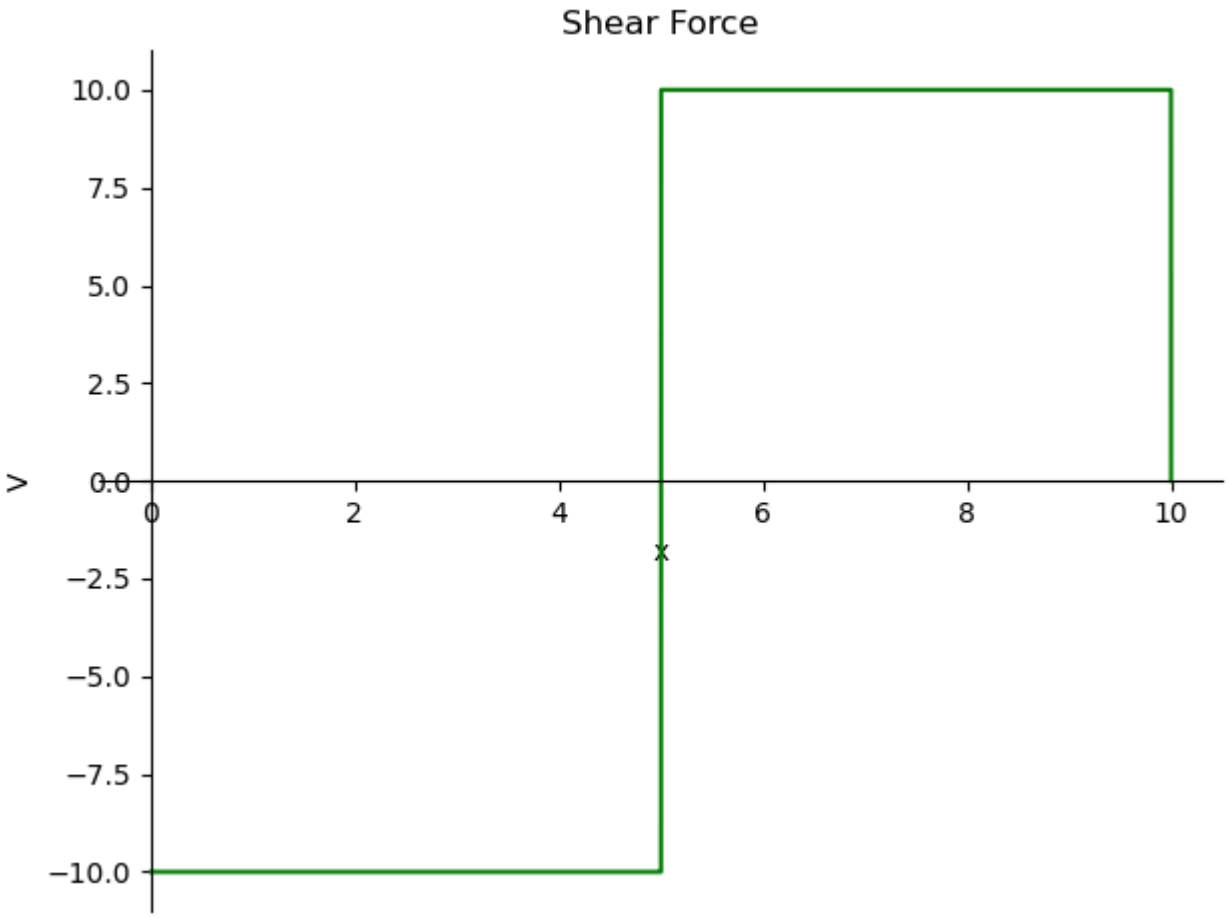
```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(10, E, I)
b.apply_support(0, type="pin")
b.apply_support(10, type="pin")
b.apply_load(-20, 5, -1)
R_0, R_10 = sm.symbols('R_0, R_10')
b.solve_for_reaction_loads(R_0, R_10)
b.reaction_loads
```

```
{R_0: 10, R_10: 10}
```

```
b.plot_bending_moment();
```



```
b.plot_shear_force();
```

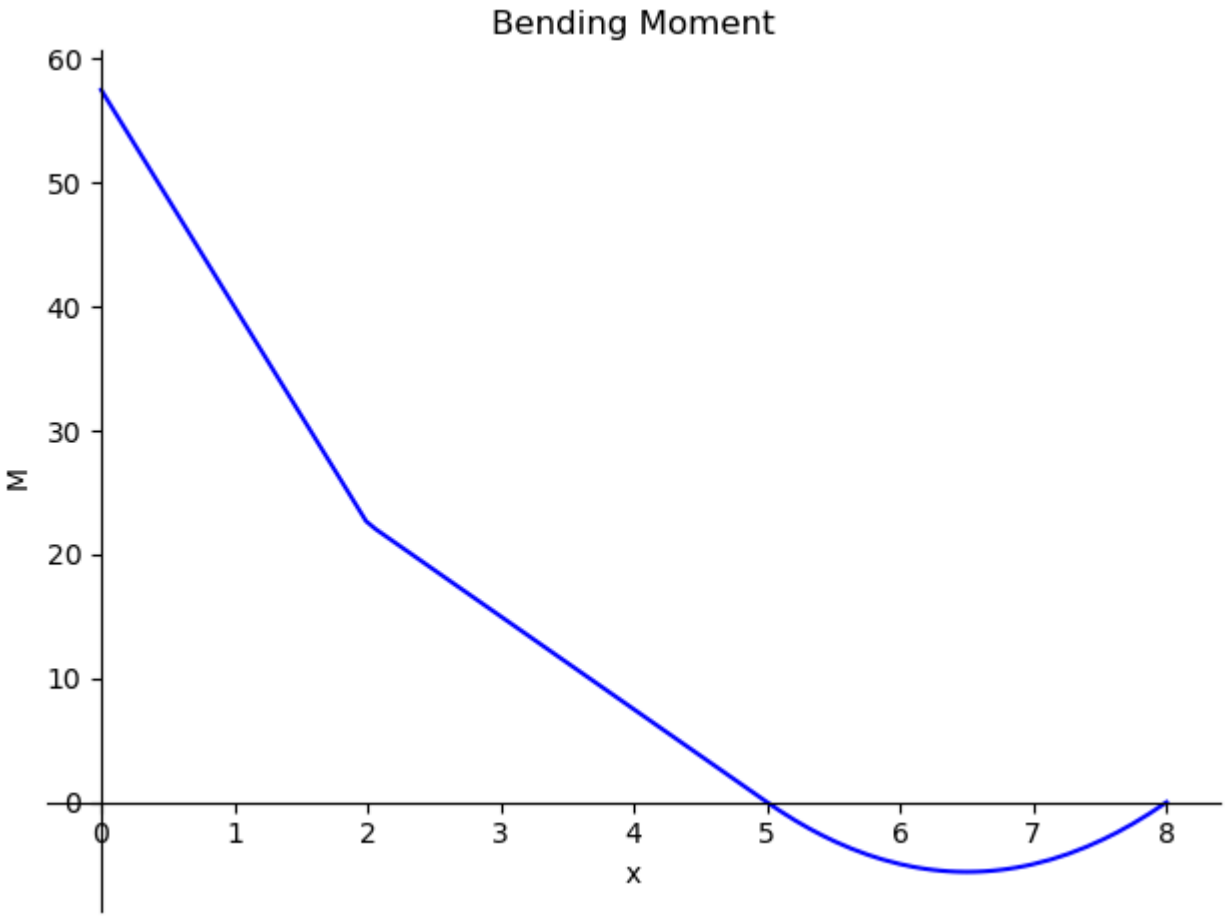


Example 3

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b1 = Beam(5, E, I)
b2 = Beam(3, E, I)
b = b1.join(b2, via="hinge")
b.apply_support(0, type='fixed')
b.apply_support(8, type='pin')
b.apply_load(-10, 2, -1)
b.apply_load(-5, 5, 0, 8)
r0, m0, r8 = sm.symbols('R_0, M_0, R_8')
b.solve_for_reaction_loads(r0, m0, r8)
```

```
b.plot_bending_moment();
```

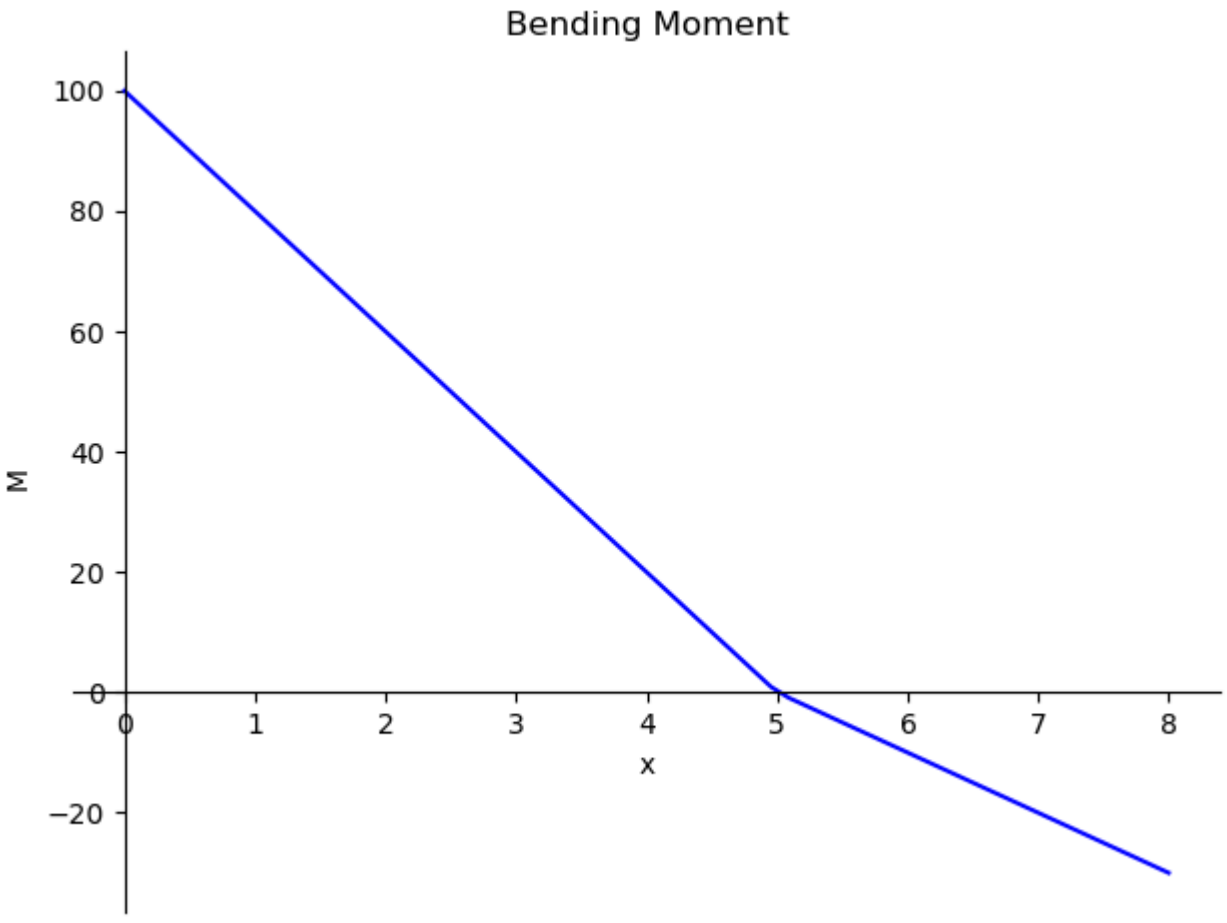


Example 4

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b1 = Beam(5, E, I)
b2 = Beam(3, E, I)
b = b1.join(b2, via="hinge")
b.apply_support(0, type='fixed')
b.apply_support(8, type='pin')
b.apply_load(-10, 5, -1)
r0, m0, r8 = sm.symbols('R_0, M_0, R_8')
b.solve_for_reaction_loads(r0, m0, r8)
```

```
b.plot_bending_moment();
```

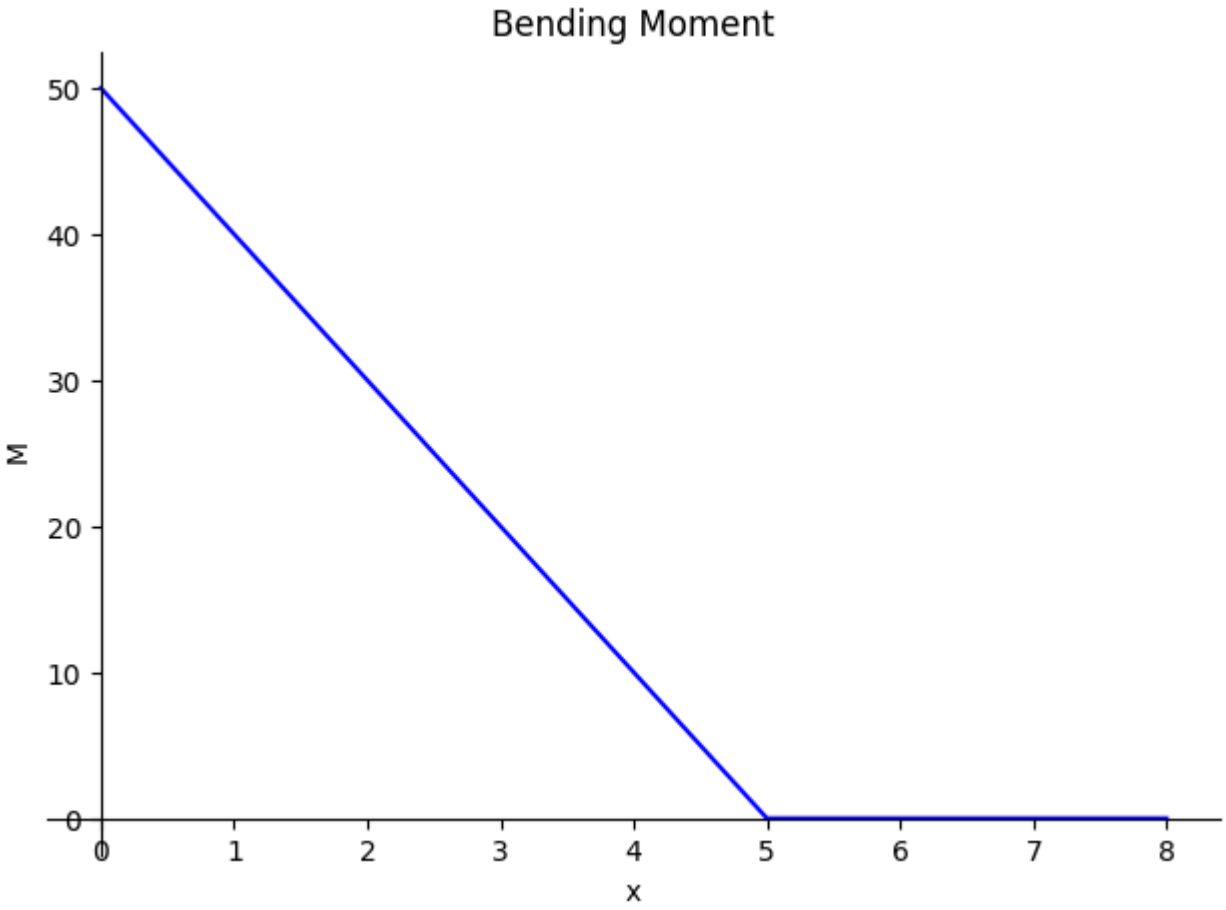


Example 4

```
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(8, E, I)
r0, m0 = b.apply_support(0, type='fixed')
r8 = b.apply_support(8, type='pin')
b.apply_rotation_hinge(5)
b.apply_load(-10, 5, -1)
b.solve_for_reaction_loads(r0, m0, r8)
```

```
b.plot_bending_moment();
```



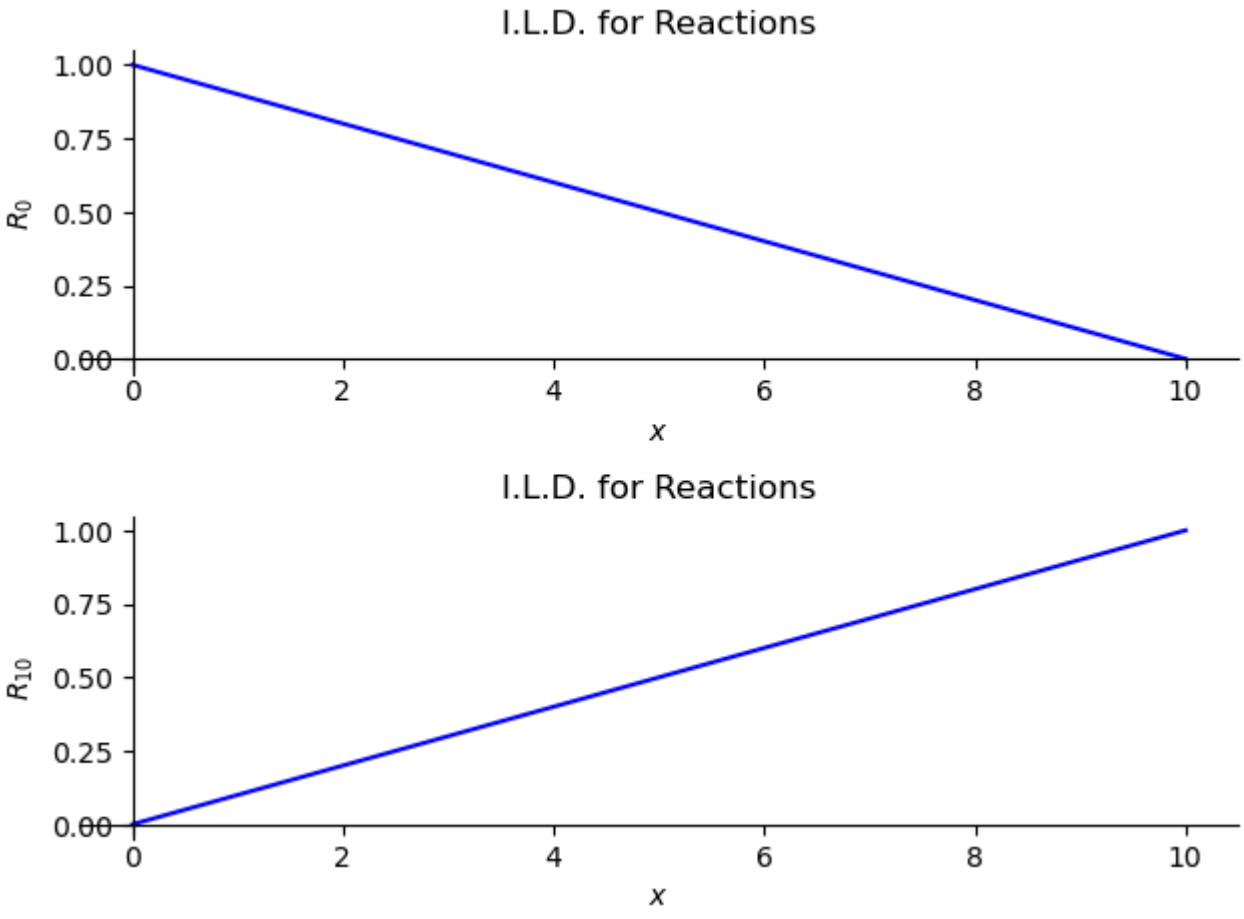
Example 5

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(10, E, I)
b.apply_support(0, type="pin")
b.apply_support(10, type="pin")
R_0, R_10 = sm.symbols('R_0, R_10')
b.solve_for_ild_reactions(-1, R_0, R_10)
b.ild_reactions
```

```
{R_0: 1 - x/10, R_10: x/10}
```

```
b.plot_ild_reactions();
```



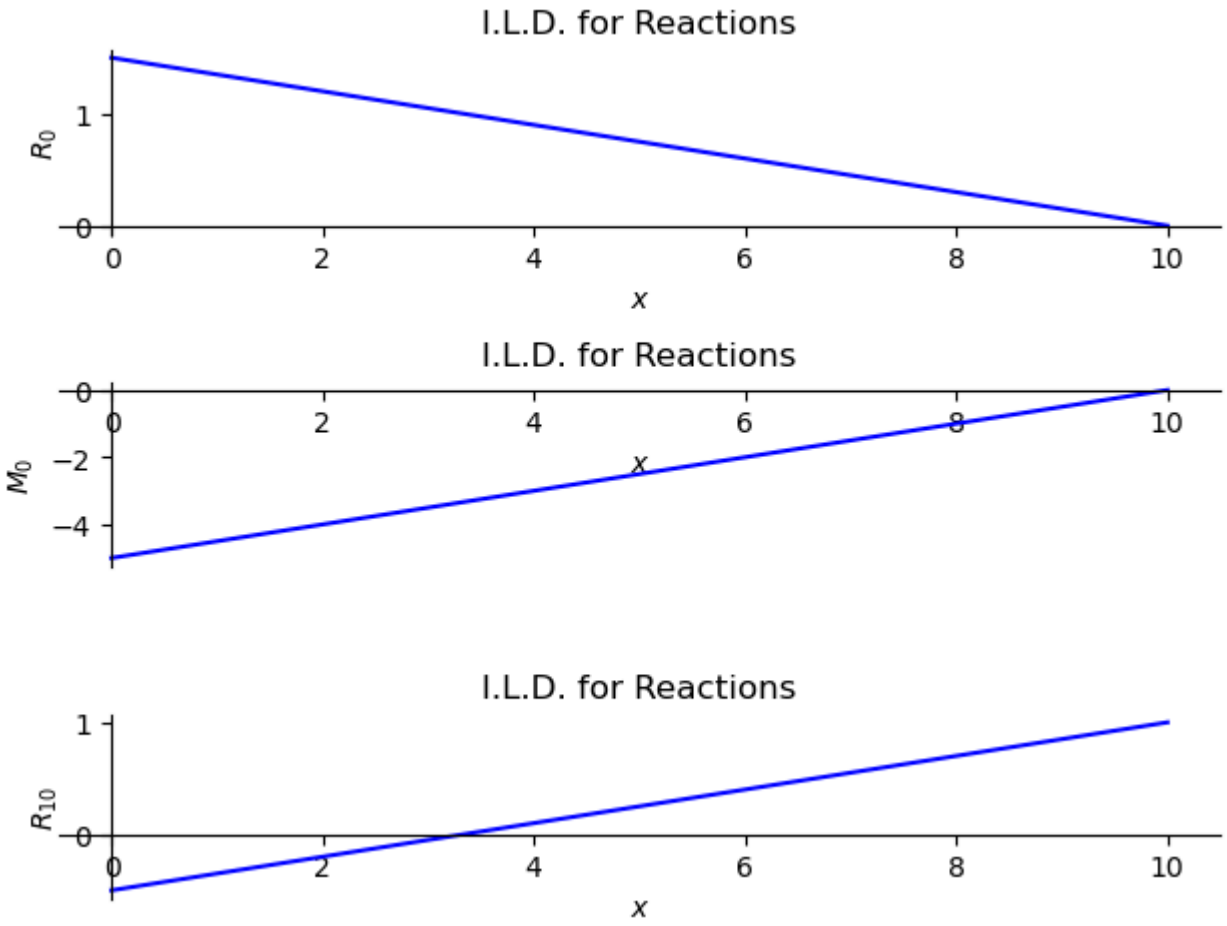
Example 6

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
x = sm.Symbol('x')
b = Beam(10, E, I)
b.apply_support(0, type="fixed")
b.apply_support(10, type="pin")
R_0, M_0, R_10 = sm.symbols('R_0, M_0, R_10')
b.solve_for_ild_reactions(-1, R_0, M_0, R_10)
b.ild_reactions
```

```
{R_0: 3/2 - 3*x/20, M_0: x/2 - 5, R_10: 3*x/20 - 1/2}
```

```
b.plot_ild_reactions();
```

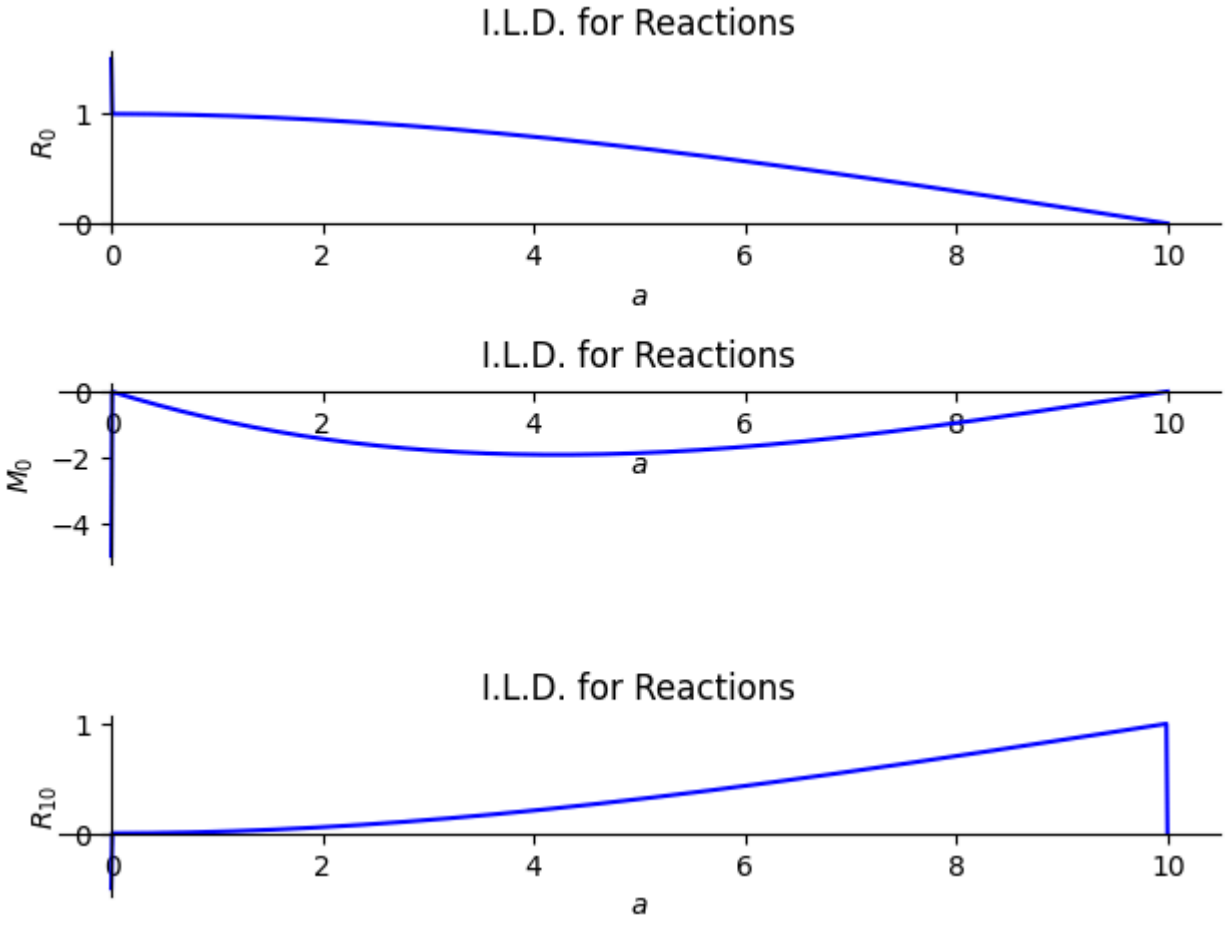


Example 6

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(10, E, I)
r0, m0 = b.apply_support(0, type="fixed")
r10 = b.apply_support(10, type="pin")
b.solve_for_ild_reactions(-1, r0, m0, r10)
```

```
b.plot_ild_reactions();
```

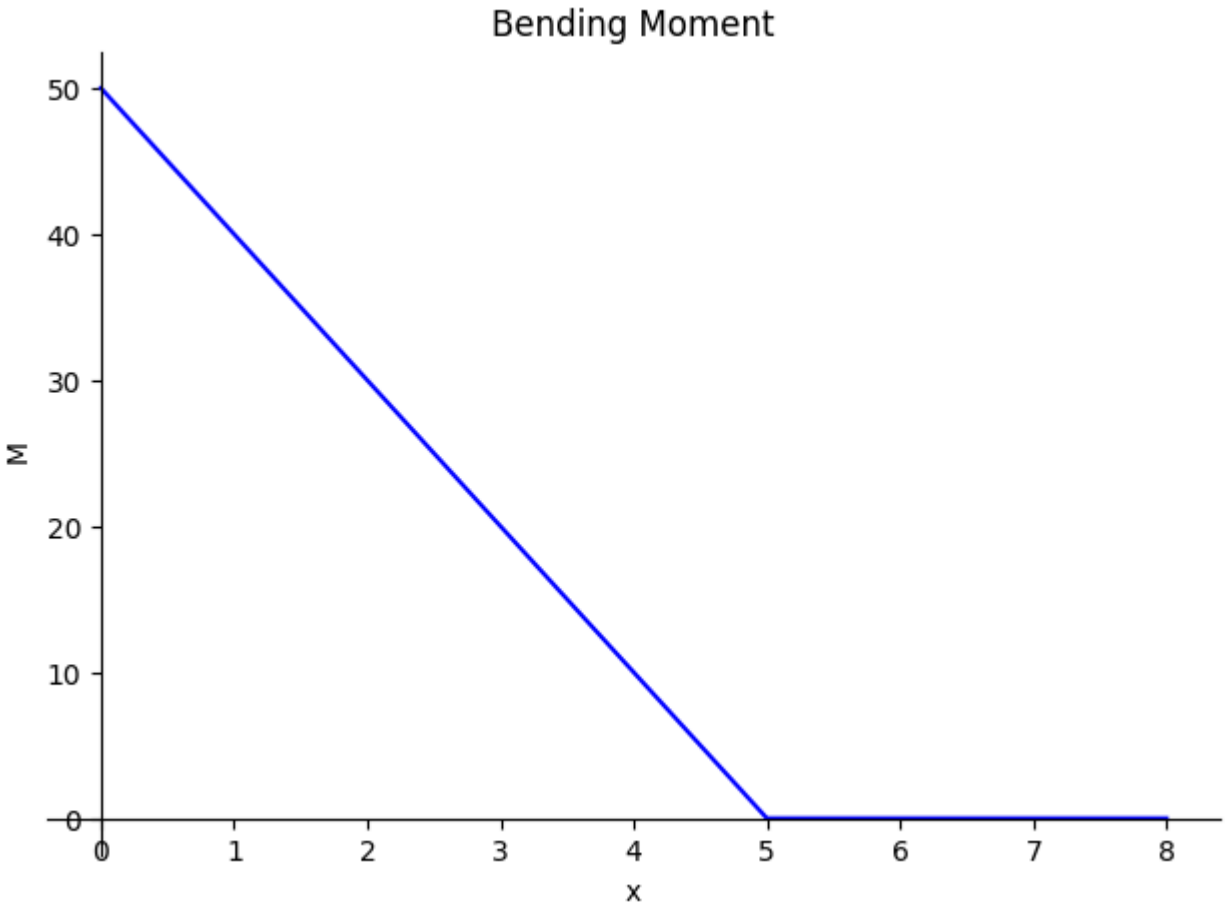


Example 7

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(8, E, I)
r0, m0 = b.apply_support(0, type='fixed')
r8 = b.apply_support(8, type='pin')
b.apply_rotation_hinge(5)
b.apply_load(-10, 5, -1)
b.solve_for_reaction_loads(r0, m0, r8)
```

```
b.plot_bending_moment();
```

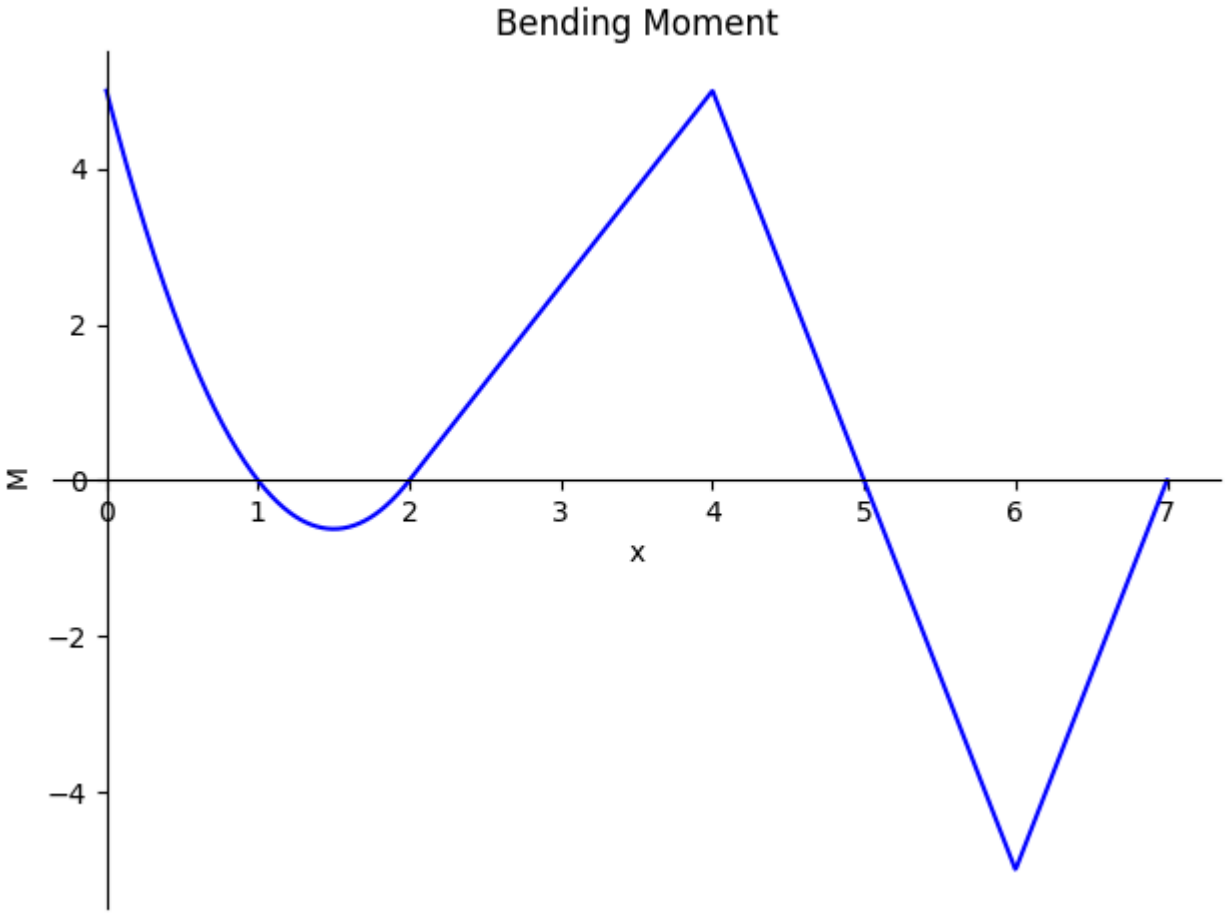


Example 8

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(7, E, I)
r0, m0 = b.apply_support(0, type='fixed')
r4 = b.apply_support(4, type='pin')
r7 = b.apply_support(7, type='pin')
b.apply_rotation_hinge(2)
b.apply_rotation_hinge(5)
b.apply_load(-5, 0, 0, 2)
b.apply_load(-10, 6, -1)
b.solve_for_reaction_loads(r0, m0, r4, r7)
```

```
b.plot_bending_moment();
```

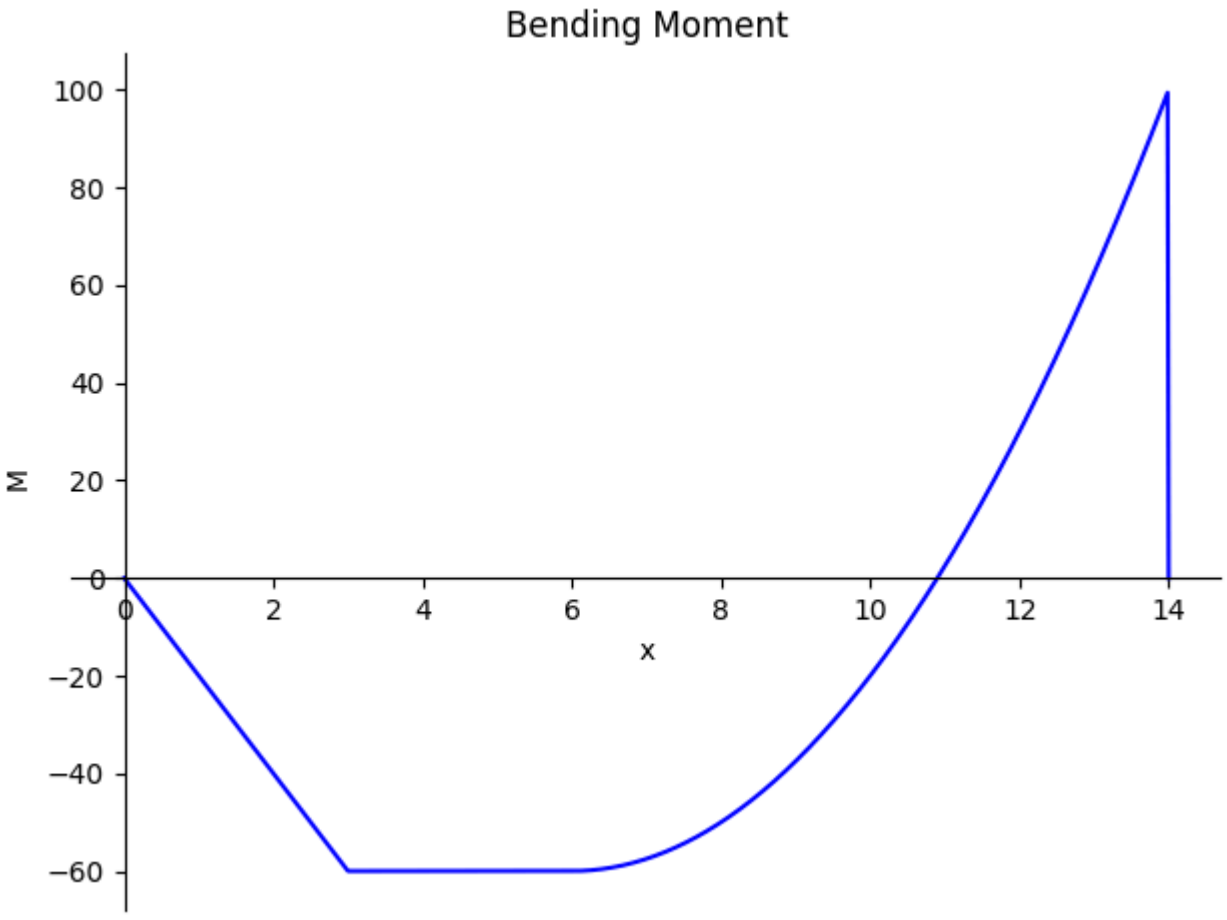


Example 9

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(14, E, I)
r0 = b.apply_support(0, type="pin")
r14, m14 = b.apply_support(14, type="fixed")
b.apply_sliding_hinge(6)
b.apply_load(-20, 3, -1)
b.apply_load(-5, 6, 0, 14)
b.solve_for_reaction_loads(r0, r14, m14)
```

```
b.plot_bending_moment();
```

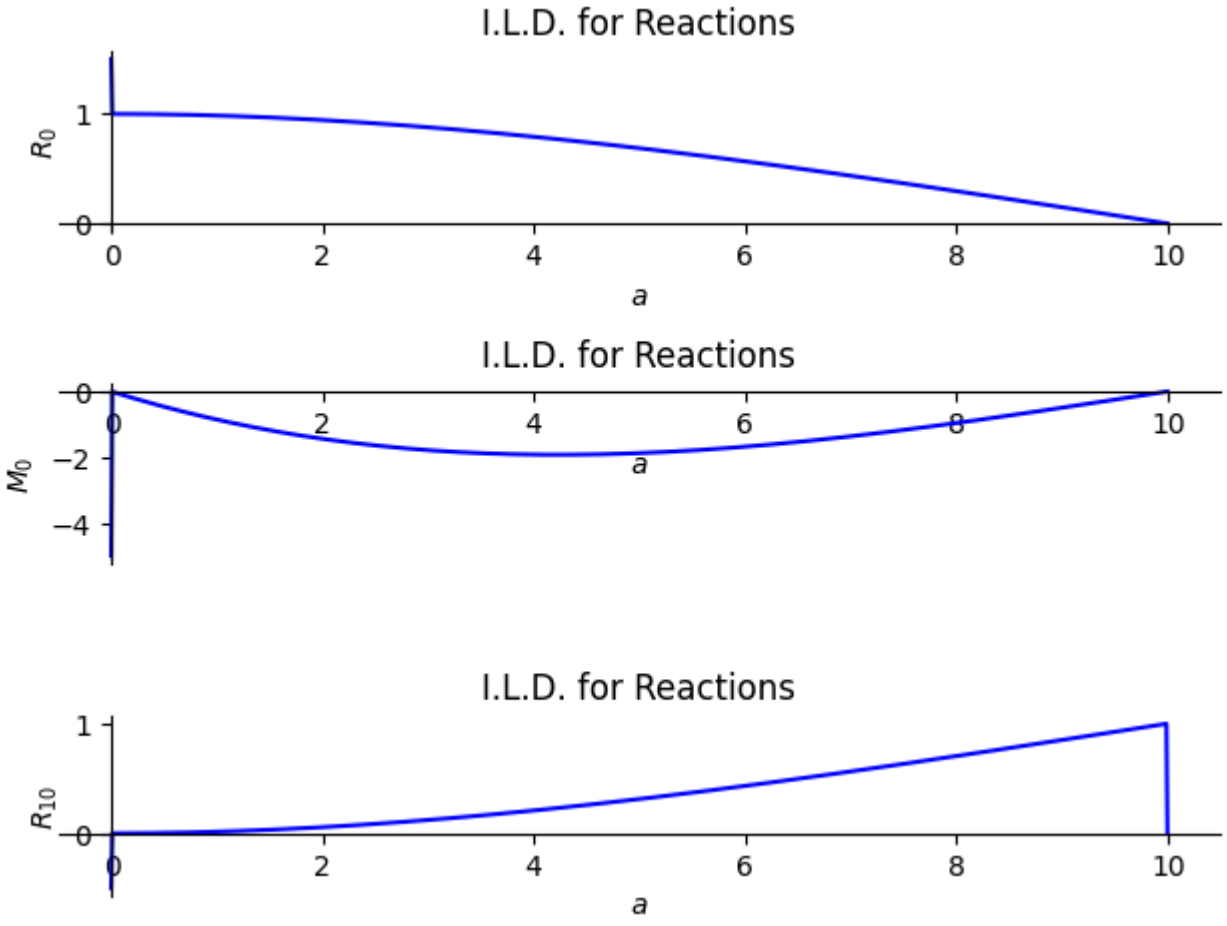


Example 10

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(10, E, I)
r0, m0 = b.apply_support(0, type="fixed")
r10 = b.apply_support(10, type="pin")
b.solve_for_ild_reactions(-1, r0, m0, r10)
```

```
b.plot_ild_reactions();
```

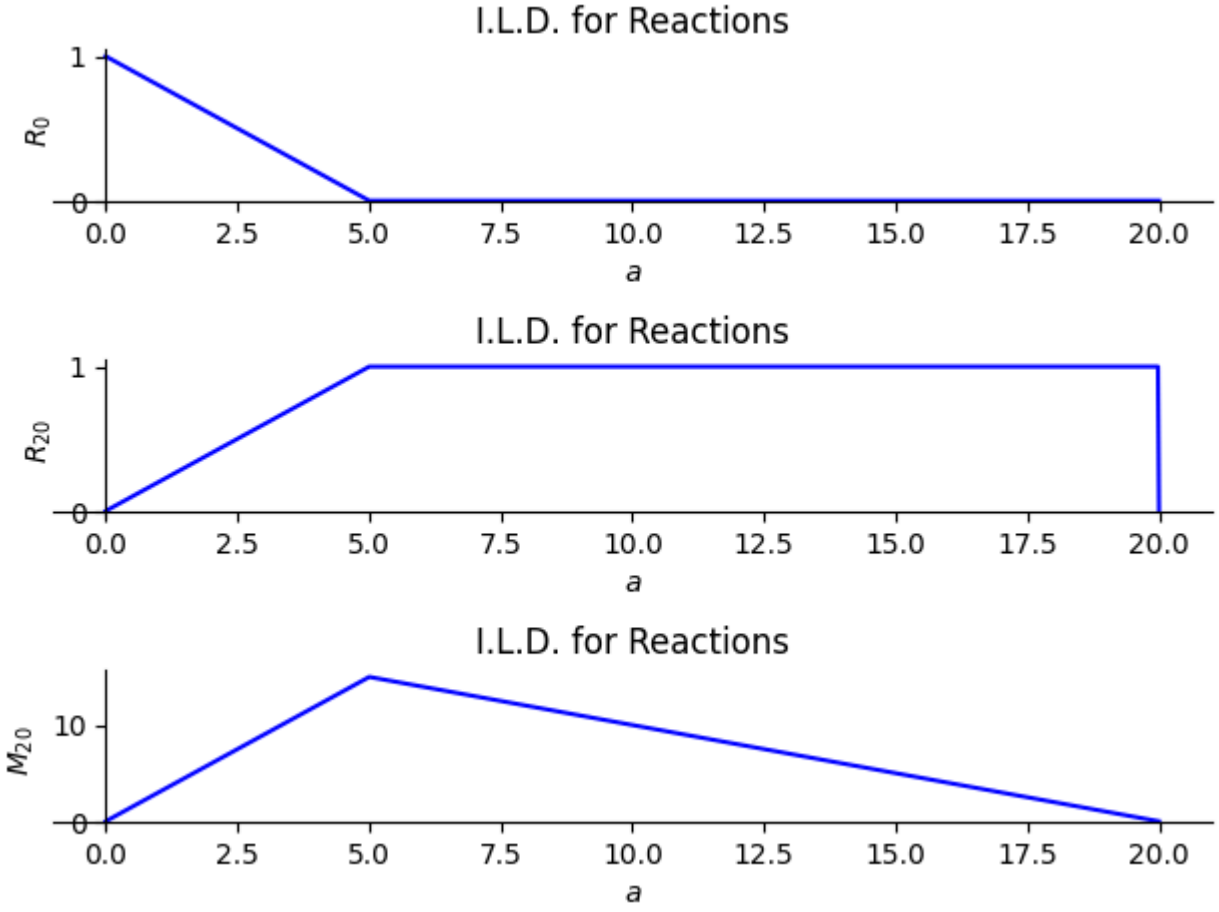


Example 11

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(20, E, I)
r0 = b.apply_support(0, type="pin")
r20, m20 = b.apply_support(20, type="fixed")
b.apply_rotation_hinge(5)
b.solve_for_ild_reactions(-1, r0, r20, m20)
```

```
b.plot_ild_reactions();
```

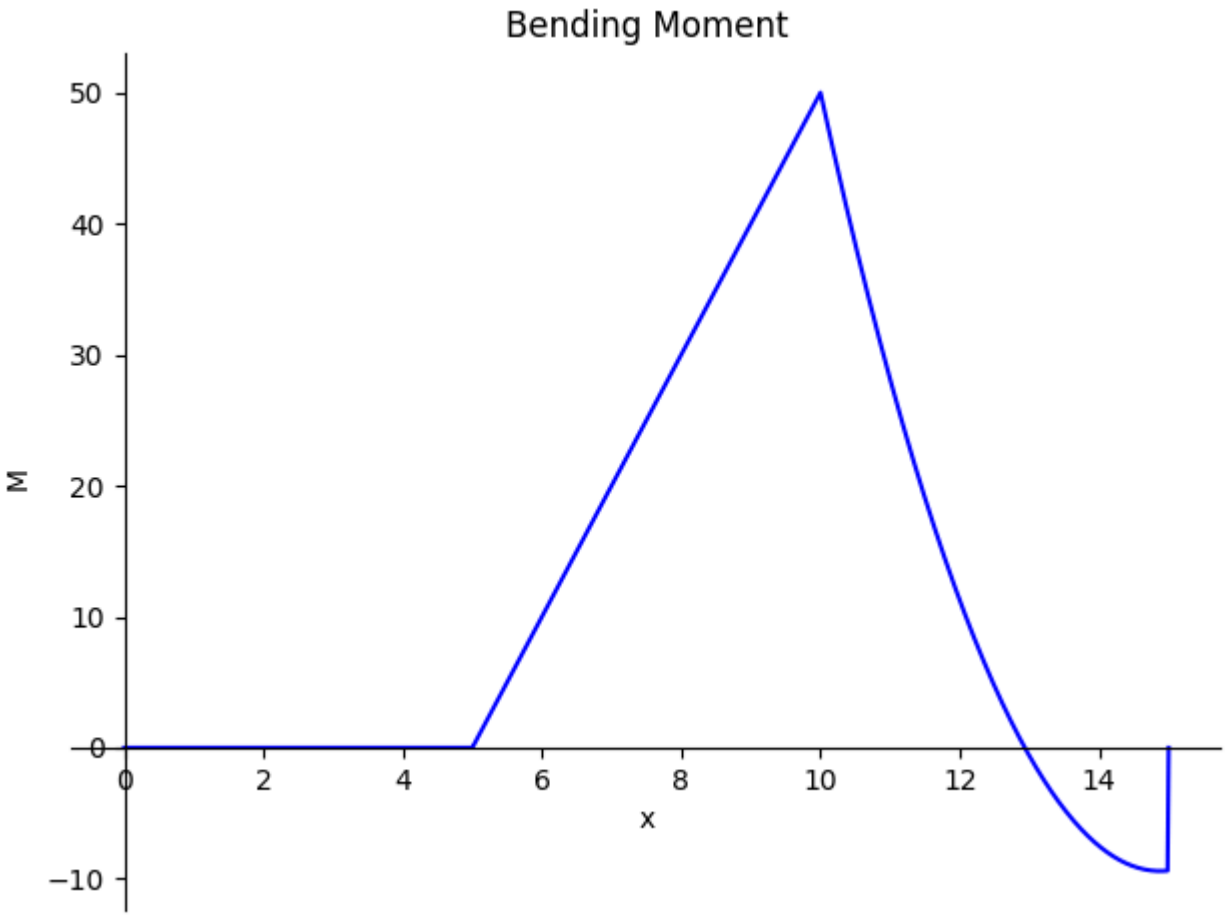


Example 12

```
%matplotlib inline
import sympy as sm
from sympy.physics.continuum_mechanics.beam import Beam
```

```
E = sm.Symbol('E')
I = sm.Symbol('I')
b = Beam(15, E, I)
r0 = b.apply_support(0, type='pin')
r10 = b.apply_support(10, type='pin')
r15, m15 = b.apply_support(15, type='fixed')
b.apply_rotation_hinge(5)
b.apply_load(-10, 5, -1)
b.apply_load(-5, 10, 0, 15)
b.solve_for_reaction_loads(r0, r10, r15, m15)
```

```
b.plot_bending_moment();
```



3-pin example

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
from sympy import Symbol
from sympy.physics.continuum_mechanics.beam import Beam
sym.init_printing()
%config InlineBackend.figure_formats = ['svg']
```

```
a = sym.symbols('a',real=True)
x = sym.Symbol('x')
E = sym.Symbol('E')
I = sym.Symbol('I')
A_v = sym.Symbol('A_v')
B_v = sym.Symbol('B_v')
C_v = sym.Symbol('C_v')
C_1, C_2, C_3, C_4 = sym.symbols('C_1 C_2 C_3 C_4')
q = A_v * sym.SingularityFunction(x, 0 , -1) +  B_v * sym.SingularityFunction(x, 5, -1) +
display(q)
```

$$A_v\langle x\rangle^{-1} + B_v\langle x-5\rangle^{-1} + C_v\langle x-10\rangle^{-1} + \langle -a+x\rangle^{-1}$$

```
V = -sym.integrate(q, x) + C_1
M = sym.integrate(V, x) + C_2
kappa = M / E / I
phi = sym.integrate(kappa, x) + C_3
w = -sym.integrate(phi, x) + C_4
```

```
Eq1 = sym.Eq(w.subs(x,0),0)
Eq2 = sym.Eq(w.subs(x,5),0)
Eq3 = sym.Eq(w.subs(x,10),0)
Eq4 = sym.Eq(M.subs(x,0-1),0)
Eq5 = sym.Eq(M.subs(x,10+1),0)
Eq6 = sym.Eq(V.subs(x,0-1),0)
Eq7 = sym.Eq(V.subs(x,10+1),0)
```

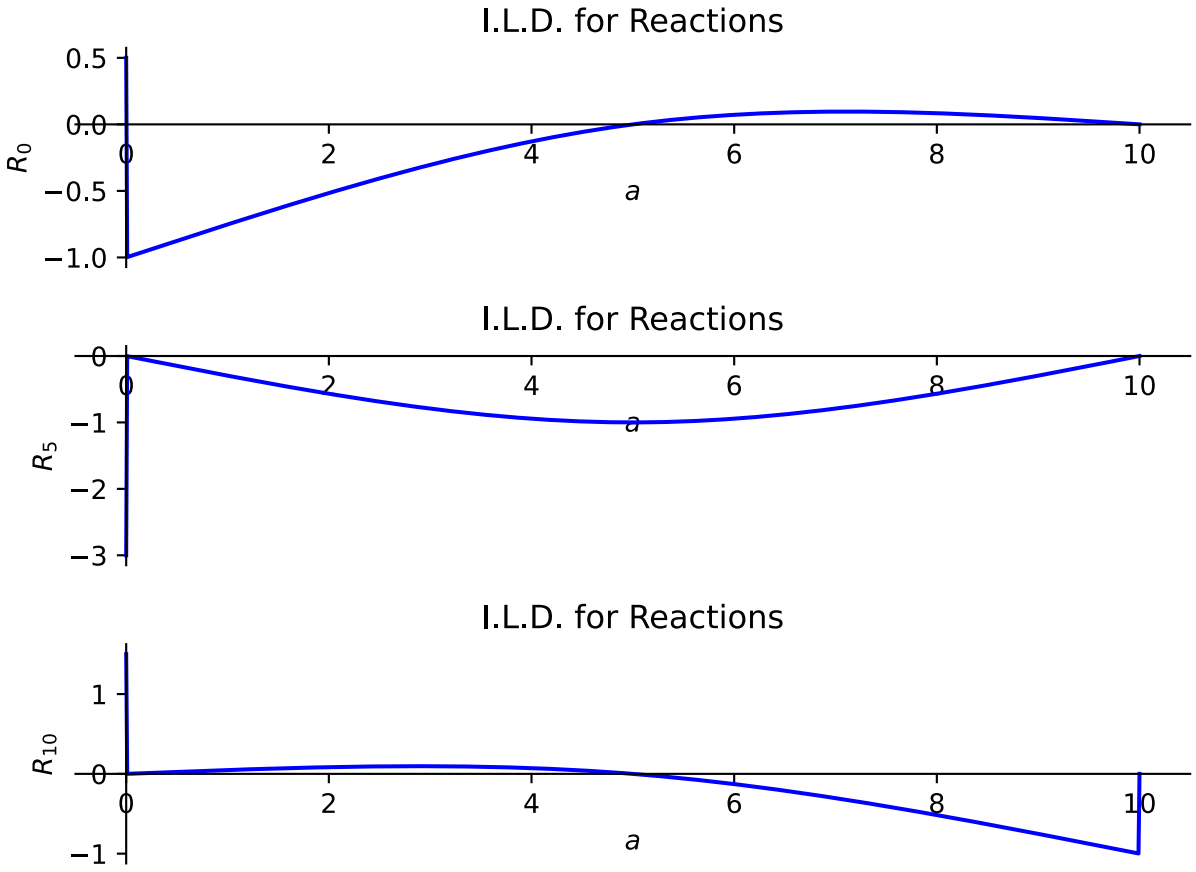
```
sol = sym.solve((Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7),(C_1,C_2,C_3,C_4,A_v,B_v,C_v))
display(sol)
```

$$\{A_v: 5\langle -a-1\rangle^0/4 + \langle -a-1\rangle^1/4 - \langle -a\rangle^1/500 + \langle 5-a\rangle^0/250 - \langle 10-a\rangle^1/500 - \langle 11-a\rangle^0/20 + \langle 11-a\rangle^1/20,\ B_v: -3\langle -a-1\rangle^0/10 - 3\langle -a-1\rangle^1/10 + \langle -a\rangle^1/250 - \langle 5-a\rangle^1/125 + \langle 10-a\rangle^0/250 + 3\langle 11-a\rangle^0/10 - 3\langle 11-a\rangle^1/10,\ C_v: \langle -a-1\rangle^0,\ C_1: \langle -a-1\rangle^1 + \langle -a-1\rangle^1,\ C_2: -\frac{39\langle -a-1\rangle^0}{40} - \frac{39\langle -a-1\rangle^1}{40} - \frac{\langle -a\rangle^1}{500} + \frac{9\langle 5-a\rangle^1}{500} - \frac{\langle 10-a\rangle^1}{1000} - \frac{51\langle 11-a\rangle^0}{40} + \frac{51\langle 11-a\rangle^1}{40},\ C_3: -\frac{\langle -a\rangle^1}{500},\ C_4: \langle -a-1\rangle^0/20 + \langle -a-1\rangle^1/20 - \langle -a\rangle^1/500 + \langle 5-a\rangle^0/250 - \langle 10-a\rangle^1/500 - 5\langle 11-a\rangle^0/4 + \langle 11-a\rangle^1/4\}$$

```
E = Symbol('E')
I = Symbol('I')
a = Symbol('a')
b = Beam(10, E, I)
r0 = b.apply_support(0, type="pin")
r5 = b.apply_support(5, type="pin")
r10 = b.apply_support(10, type="pin")
b.solve_for_ild_reactions(1, r0, r5, r10)
display(b.ild_reactions)
```

$\{R_0: -(-a)^3/500 + (5-a)^3/250 - (10-a)^3/500 + 3\langle -a\rangle^2\langle -a\rangle^2/2 + 3\langle -a\rangle^2\langle -a\rangle^1/10 + \langle a\rangle^2/2 - \langle a\rangle^1/20 + \langle a-10\rangle^1/20,\ R_{10}: -(-a)^3/500 + (5-a)^3/250 - (10-a)^3/500 + 3\langle -a\rangle^2\langle -a\rangle^2/2 + 3\langle -a\rangle^2\langle -a\rangle^1/10 + 3\langle a\rangle^2/2 - \langle a\rangle^1/4 + \langle a-10\rangle^2 + \langle a-10\rangle^1/4,\ R_5: \langle -a\rangle^3/250 - (5-a)^3/125 + (10-a)^3/250 - 3\langle -a\rangle^2\rangle^2 - 3\langle -a\rangle^2\langle -a\rangle^1/5 - 3\langle a\rangle^2 + 3\langle a\rangle^1/10 - 3\langle a-10\rangle^1/10\}$

```
b.plot_ild_reactions();
```



Borek: Building a Symbolic 2D Structural Analysis Tool Using SymPy

Contents

This report describes the development of a Python based tool for 2D structural analysis in civil engineering, using the SymPy library (Meurer A, 2017) and applying the Macaulay method (Macaulay, 1919) to enable symbolic computation for structural analysis. The tool serves as a proof of concept demonstrating the feasibility of symbolic computation in analyzing internal forces.

Central to the project is the extension of existing SymPy modules, specifically the Beam module, to support 2D analysis by introducing new modules: “Column” and “Structure2D.” The existing “Beam” module in SymPy, which computes vertical load effects, was used to handle shear forces and bending moments by creating a load equation for the vertical direction. To handle normal forces a “Column” module was proposed (although not implemented in this iteration), designed to compute horizontal load effects based on the material’s elasticity modulus, length, and cross-sectional area. The “Structure2D” module integrates both vertical and horizontal load computations by transforming a 2D structure into two corresponding 1D representations. This method enables separate computations for each direction, converting 2D problems into simpler 1D computations while preserving load and support behavior.

The tool’s workflow involves users defining structure components (members, loads, supports) on a 2D grid. Once defined, the structure is “unwrapped” to map each member along a continuous line. Each load is split into its horizontal and vertical components and applied to the corresponding axis. Through examples, the report showcases the tool’s ability to calculate shear forces, bending moments, and reaction loads.

> Although the project successfully demonstrates the potential of symbolic computation for 2D structural analysis, several limitations exist. The tool currently lacks a functional Column module, restricting it to single horizontal reaction loads and limiting its ability to solve for horizontal deflections and axial forces fully. Additionally, the structure must be non-branching and linear, excluding complex configurations like truss networks. Material properties must be uniform across members, a constraint that simplifies the model but limits its real-world applicability. Addressing these limitations would involve developing a more advanced unwrapping algorithm for intersecting members and introducing a complete Column module.
>
> In summary, this prototype validates the possibility of a symbolic 2D structural analysis tool, providing civil engineering students and educators with an analytical framework that combines Macaulay's method and modern symbolic computation. This tool lays the groundwork for future developments that could expand its functionality, to move away from proof of concept to a fully featured open-source product.

Saheli (2024)

Documenten

- TU Delft repository
- GitHub repository, example as in report shown in this book

Example 1

```
%pip install git+https://github.com/BorekSaheli/sympy.git@structure2d
```

Consider a two-dimensional structure, consisting of three connected members. The first member is 4 meters long and horizontal. The second member is connected to the end of the first member and is 5 meters long, forming an angle of arctangent $\left(\frac{3}{4}\right)$ relative to the x-axis. The third member is also 5 meters long and forms an angle of arctangent $\left(\frac{4}{3}\right)$ relative to the x-axis. There is a fixed support at the end of the third member. The sign convention used in this example is, forces acting downward and to the right are considered positive. Angles for loads are measured relative to the positive x-axis in the anticlockwise direction, and angles for members are measured relative to the positive x-axis in the clockwise direction.

The material properties are defined as follows:

- Elasticity Modulus (E): 30000 kN/m^2
- Second Moment of Area (I): 1 m^4
- Cross-sectional Area (A): 10000 m^2

The structure has two-point loads:

- F_h: A horizontal load of 15 kN applied at the start of the first member, acting to the right (positive x).
- F_v: A vertical load of 16 kN applied at the midpoint of the first member, acting downward. (negative y)

Additionally, there are two distributed loads, noted as q_v:

- The first distributed load is 6 kN/m, applied along the entire length of the first member, acting downward (negative y).
- The second distributed load is 6 kN/m, applied over the first half of the second member, also acting downward (negative y).

Let's solve the example using the structure2d module. First, we import the Structure2d module from the SymPy library with `sympy.physics.continuum_mechanics.structure2d` . We then define the material properties: the elasticity modulus (`E`), the second moment of area

(`I`), and the cross-sectional area (`A`). Next, we create an instance of the Structure2d class, assigning it to the variable `s`. This instance sets up the code to run method functions on the Structure2d object.

Note that the line `%config InlineBackend.figure_format = 'svg'` is used in Jupyter notebooks to configure the output format of plots for better image quality and `init_printing` is used for proper math output.

```
import matplotlib.pyplot as plt
from sympy.physics.continuum_mechanics.structure2d import Structure2d
from sympy import init_printing
init_printing()
%config InlineBackend.figure_format = 'svg'

E = 3e4
I = 1
A = 1e4
s = Structure2d()
```

Once the structure object is initialized, we need to input the example problem into the Structure2D object. This input consists of two main steps: adding members and adding loads. Both steps involve defining x and y coordinates on a two-dimensional grid, where the structure can be drawn. Additionally, the user can generate a plot of the structure at any step, allowing for easy visualization throughout the process. Values for loads can be either SymPy symbols or numerical values.

Let's build the structure from the example. Start by selecting an arbitrary reference point on the grid in this case, the reference point is chosen to be $(0, 0)$. The first member is horizontal and 4 meters long, starting at the reference point and extending from left to right. Therefore, the coordinates for the start and end of this member are $(0, 0)$ and $(4, 0)$, respectively.

For the two remaining members, the coordinates are less straightforward. Since Structure2D currently doesn't support adding members using angles and lengths directly, we must first convert these into sets of coordinates based on the geometry of the structure. This design choice was made to save time, there is a possibility to make a new function that can add members based on their angle and length. Suppose the second member is angled and extends from $(4, 0)$ to $(8, 3)$, and the third member extends from $(8, 3)$ to $(11, -1)$. These coordinates are determined by calculating the end points based on lengths and angles or from known coordinates in the problem. Once the coordinates are determined, we assign the material properties defined earlier to all members.

```

```

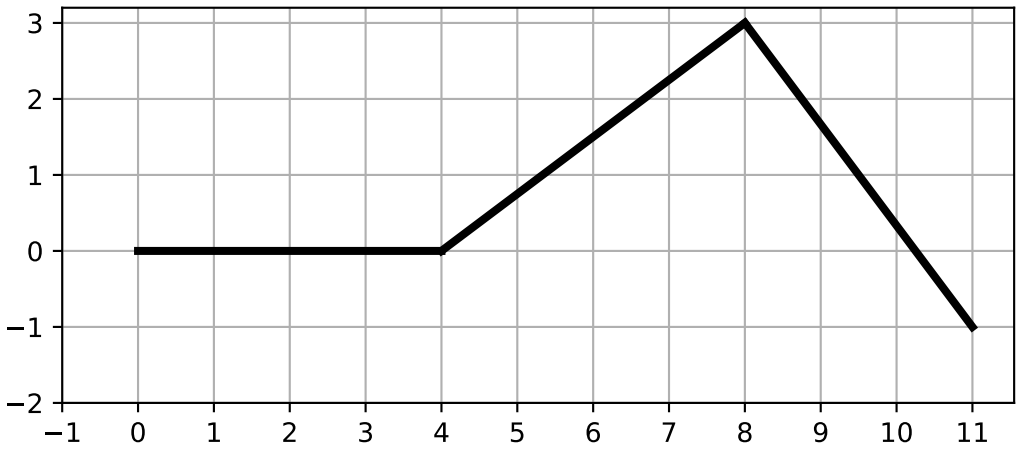
```
s.add_member(x1=0, y1=0, x2=4 , y2=0 , E=E, I=I, A=A)
s.add_member(x1=4, y1=0, x2=8 , y2=3 , E=E, I=I, A=A)
s.add_member(x1=8, y1=3, x2=11, y2=-1, E=E, I=I, A=A)
```

At any point during the process of solving, it is possible to use the draw method to obtain a plot that shows the current state of the structure. This not only helps to visualize the result but also assists during the setup process by verifying the correct placement of members. The draw method offers several parameters that allow you to customize the visualization:

- `forced_load_size` : A numerical value that forces all loads to be drawn with the specified magnitude, based on the grid units. This can help standardize the appearance of loads for better visual comparison, when load values are very high.
- `show_load_values`: A Boolean value (`True` or `False`) that toggles whether the numerical values of the loads are displayed on the plot. Setting it to True will annotate the loads with their magnitudes.
- `draw_support_icons`: A Boolean value (`True` or `False`) that determines whether icons representing the types of supports (fixed, pinned, roller) are drawn at the support locations.

Let's use the draw method with these parameters to visualize the current state of the structure.

```
s.draw()
```



Now that the structure is set up, we can add loads to it. The Structure2D object provides two methods to apply loads: `apply_load()` for external loads and `apply_support()` for support reactions. In the example, there are four external loads: two-point loads and two distributed loads. The `apply_load()` method takes several parameters:

- `start_x` and `start_y`: the coordinates where the load starts.
- `value`: the magnitude of the load.

- `global_angle`: the angle at which the load acts, measured in degrees from the positive x-axis.
- `order`: the order of the load, based on the mathematical theory of singularity functions -1 for point loads and 0 for constant distributed loads.
- `end_x` and `end_y` (optional): the coordinates where the load ends (required for distributed loads).

Let's input the first two-point loads, F_h and F_v. F_h is a point load of 15 kN acting at $(0, 0)$ along the positive x-axis (0 degrees). F_v is a point load of 16 kN acting at $(2, 0)$ downward (270 degrees)

```
s.apply_load(start_x=0, start_y=0, value=15, global_angle=0 , order=-1)
s.apply_load(start_x=2, start_y=0, value=16, global_angle=270, order=-1)
```

For the distributed loads, we need to specify the start and end coordinates. For example, a distributed load of 6 kN/m acting downward between $(0, 0)$ and $(4, 0)$ is applied and another distributed load from $(4, 0)$ to $(6, 1.5)$ is applied.

```
s.apply_load( start_x=0, start_y=0, value=6, global_angle=270, order=0, end_x=4, end_y=0,
s.apply_load( start_x=4, start_y=0, value=6, global_angle=270, order=0, end_x=6, end_y=1.
```

Next, we add the support reactions using the `apply_support()` method. This method takes the following parameters:

- `x` and `y`: the coordinates of the support.
- `type`: the type of support ('fixed', 'pin', 'roller').

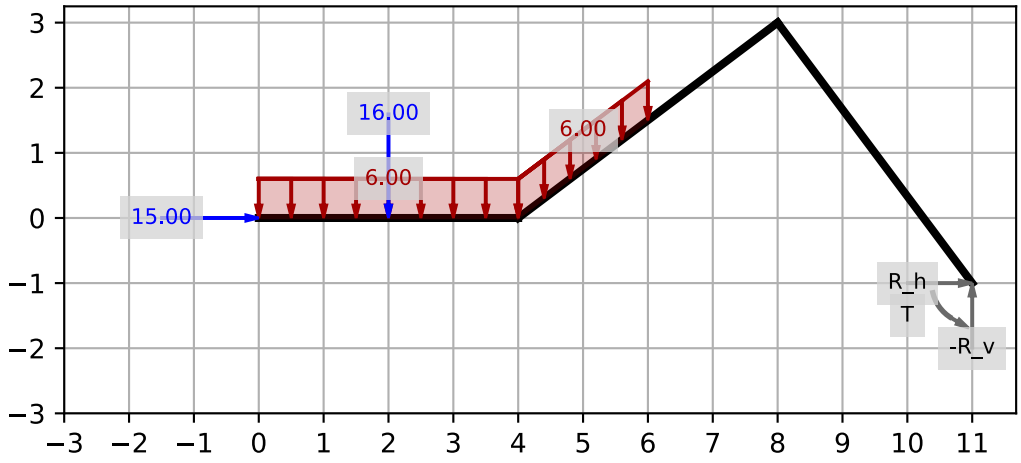
In the example, we have a fixed support at $(11, -1)$. The `apply_support()` method returns the symbolic reaction forces, these need to be assigned to variables (we will use these in the next step). As our example has a fixed support, 3 reaction loads are returned.

```
Rv, Rh, T = s.apply_support(x=11, y=-1, type="fixed")
```

In this code, `Rv` and `Rh` represent the vertical and horizontal reaction forces, and `T` represents the moment reaction at the fixed support. By using the $apply_support()$ method, we have added support reactions to our structure. Let's use the draw method to visualize what we just added, this time we can set the `show_load_values` input to `True` to view the load values.

```
```

```
s.draw(show_load_values=True)
```



We are done with the setup and have input all the information from the example into the structure2d class and are ready to solve.

To solve the structure, we use the `solve_for_reaction_loads()` method of the Structure2d object. This method calculates the reaction forces at the supports and determines the shear force and bending moment equations along the structure. The method takes the following parameter:

- `reaction_force_variables`: Variables representing all reaction loads acting on the structure.

In our example, we have reaction forces `Rv`, `Rh`, and `T` at the fixed support located at $(11, -1)$. We pass these variables to the `solve_for_reaction_loads()` method to solve the structure.

```
s.solve_for_reaction_loads(Rv, Rh, T)
```

$$\left\{R_h^{11,-1} : -15.0,\ R_v^{11,-1} : -55.0,\ T^{11,-1} : -435\right\}$$

After running this code, the structure is in a solved state and the method will return the reaction loads. This means that the reaction loads at the supports have been determined, and the shear force and bending moment equations for each member have been calculated.

With the structure solved, we can generate outputs to analyze the results. The Structure2D object provides several methods to display the results, including plotting the structure with loads and supports, printing a summary of the reaction forces and internal forces, and plotting the shear force and bending moment diagrams. First, we can print a summary of the solved reaction loads and internal forces using the summary method. This method takes an optional parameter `round_digits` to specify the number of decimal places for rounding, if set

to `None`, it will display exact analytical values. Currently the exact analytical values in some cases are not always computed for the reaction loads resulting in numerical values, this is a known bug. In this summary:

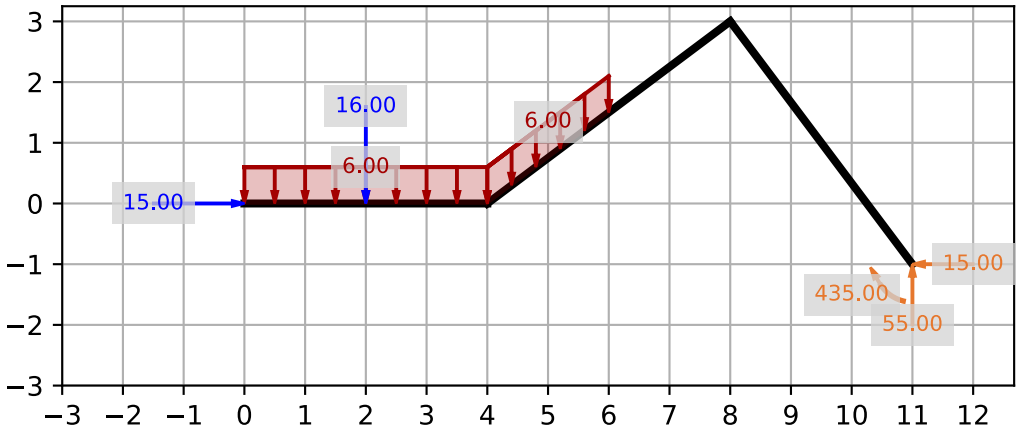
- Reaction Loads: Displays the calculated reaction forces at the supports. The coordinates of the support are given in square brackets, and the unwrapped coordinate along the structure is given in parentheses.
- Points of Interest - Bending Moment: Lists the bending moment values at specific points along the structure.
- Points of Interest - Shear Force: Lists the shear force values at specific points along the structure.
- The location computation for the coordinates on the structure is not yet implemented for bending moments and shear forces, only the unwrapped coordinates are shown. The square brackets currently have place holders.

```
s.summary(round_digits=None)
```

```
==================== Structure Summary ====================

Reaction Loads:
R_v   [11.00,-1.00]  (14.00)             = -55.0000000000000
T     [11.00,-1.00]  (14.00)             = -435
R_h   [11.00,-1.00]  (14.00)             = -15.0000000000000

Points of Interest - Bending Moment:
bending_moment at [x.xx,y.yy]  (0.00)    = 0
bending_moment at [x.xx,y.yy]  (4.00)    = -80
bending_moment at [x.xx,y.yy]  (9.00)    = -330
bending_moment at [x.xx,y.yy]  (14.00)-  = -434.999979000000

Points of Interest - Shear Force:
shear_force at [x.xx,y.yy]  (0.00+)      = 0
shear_force at [x.xx,y.yy]  (2.00-)      = -11.9999940000000
shear_force at [x.xx,y.yy]  (2.00+)      = -28
shear_force at [x.xx,y.yy]  (4.00-)      = -39.9999940000000
shear_force at [x.xx,y.yy]  (4.00+)      = -41
shear_force at [x.xx,y.yy]  (9.00-)      = -53.0000000000000
shear_force at [x.xx,y.yy]  (9.00+)      = -21
shear_force at [x.xx,y.yy]  (14.00-)     = -21.0000000000000
```

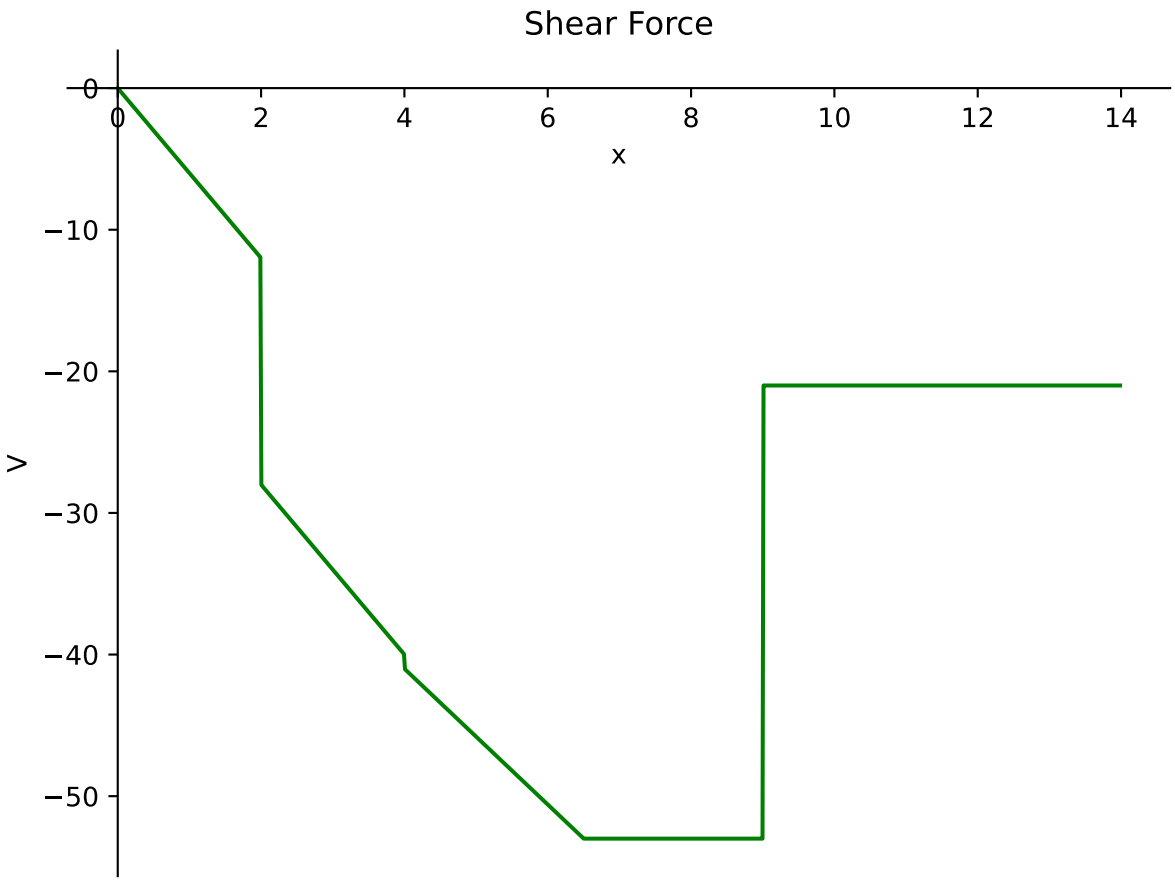
Next, we can redraw the structure, now numerical values of the reaction loads are displayed.

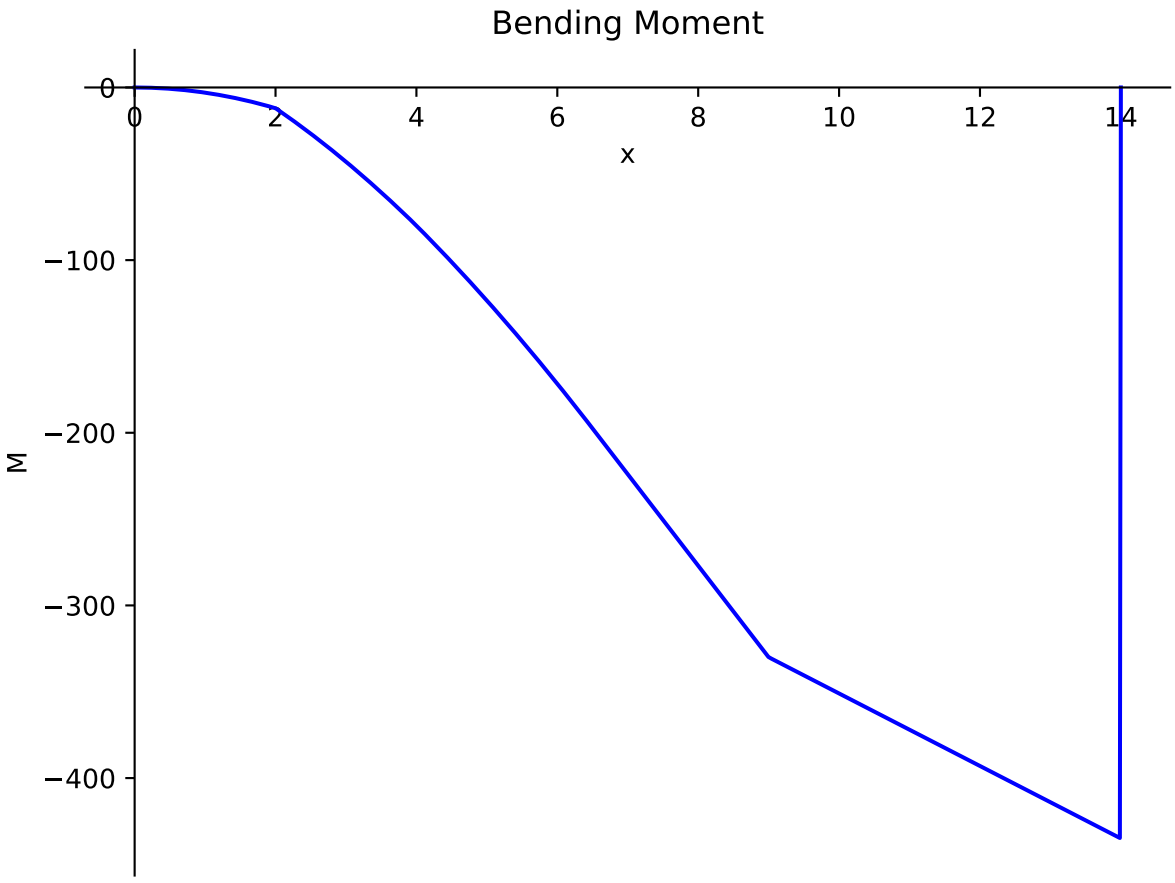
```
s.draw(show_load_values=True)
```



We can also plot the shear force and bending moment diagrams using the `plot_shear_force()` and `plot_bending_moment()` methods. These methods generate plots of the shear force and bending moment along the length of the unwrapped structure. Note that due to current limitations, these diagrams are plotted along the unwrapped coordinate of the structure and are not overlaid onto the original two-dimensional geometry, this could be done another possible future enhancement for the module.

```
s.plot_shear_force()
s.plot_bending_moment()
```





Additionally, we can evaluate the shear force and bending moment at any point along the structure using the `shear_force()` and `bending_moment()` methods. These methods accept either an x coordinate or both x and y coordinates.

```
print(s.shear_force(2,0))
print(s.bending_moment(2,0))
```

```
-28
-12
```

In this example, we evaluate the shear force and bending moment at the point $(2, 0)$ on the structure. Due to current limitations, when using both x and y coordinates, the methods do not support evaluating values just before or just after a point where discontinuities may occur (e.g., at points of applied loads). This is important because shear forces can exhibit sudden jumps at these locations. As a workaround, if only the x coordinate is provided, the evaluation is performed along the unwrapped coordinate of the structure. This allows us to evaluate the values just before or just after a discontinuity by using a very small value dx.

```
dx = 1e-6
print(s.shear_force(2 - dx))
print(s.shear_force(2 + dx))
```

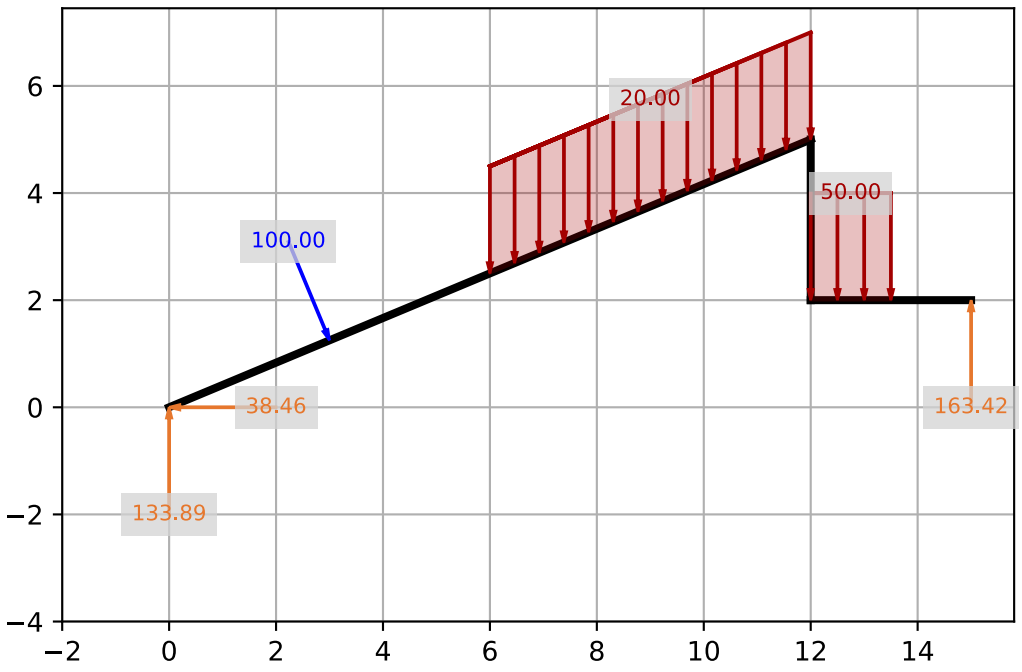
```
-11.9999940000000
-28.0000060000000
```

Example 2

```
%pip install git+https://github.com/BorekSaheli/sympy.git@structure2d
```

A constant distributed load is applied over the second half of the first member and a second distributed load is applied over first hald of the third member. The structure is supported by a pin support at one end and a roller support at the other end.

```
import matplotlib.pyplot as plt
from sympy.physics.continuum_mechanics.structure2d import Structure2d
%config InlineBackend.figure_format = 'svg'

E = 3e4
I = 1
A = 1e4
F = 15
s = Structure2d()
s.add_member(x1=0, y1=0, x2=12, y2=5, E=E, I=I, A=A)
s.add_member(x1=12, y1=5, x2=12, y2=2, E=E, I=I, A=A)
s.add_member(x1=12, y1=2, x2=15, y2=2, E=E, I=I, A=A)
s.apply_load(
    start_x=6,
    start_y=2.5,
    value=20,
    global_angle=270,
    order=0,
    end_x=12,
    end_y=5,
)
s.apply_load(
    start_x=12,
    start_y=2,
    value=50,
    global_angle=270,
    order=0,
    end_x=13.5,
    end_y=2,
)
s.apply_load(
    start_x=3,
    start_y=1.25,
    value=100,
    global_angle=s.members[0].angle_deg + 270,
    order=-1,
)
Rv1 = s.apply_support(x=15, y=2, type="roller")
Rv2, Rh2 = s.apply_support(x=0, y=0, type="pin")
s.solve_for_reaction_loads(Rv1, Rv2, Rh2)
s.draw(show_load_values=True, forced_load_size=2)
```

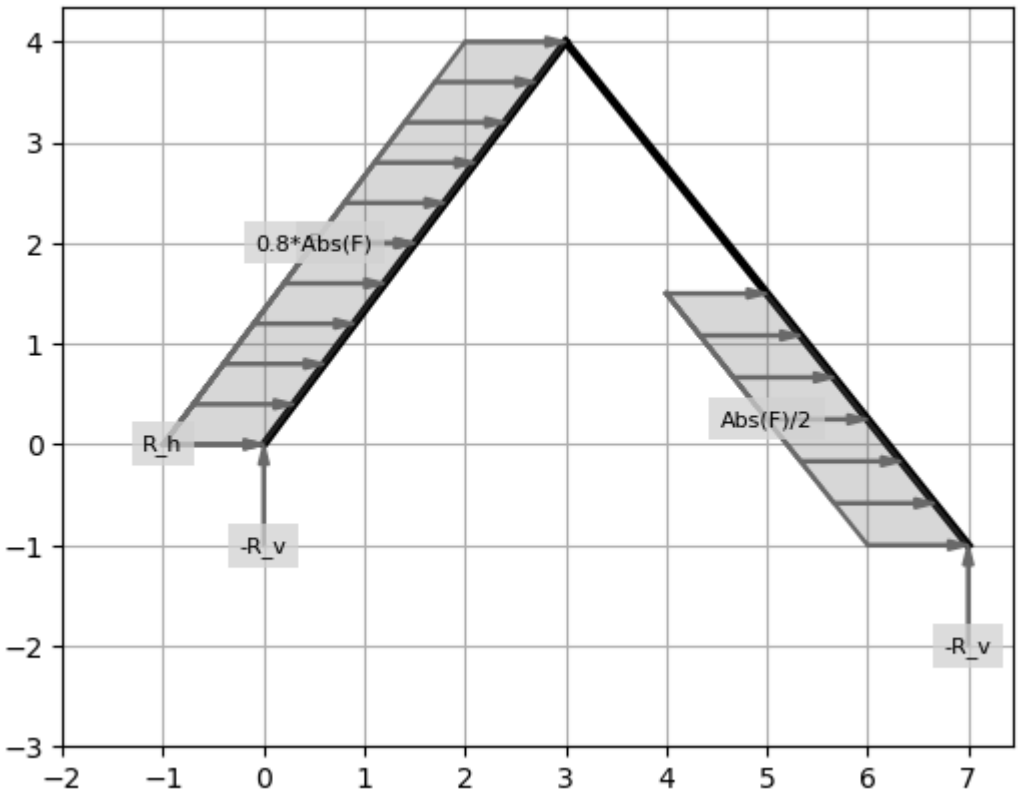

20.00
50.00
100.00
38.46
133.89
163.42
6
4
2
0
−2
−4
−2
0
2
4
6
8
10
12
14

Example 3

There is a structure containing 2 members. A constant distributed load is applied over the length of the first member. The structure is supported by a pin support at one end and a roller support at the other end. With multiple loads symbolic loads it can be visualized using the draw method.

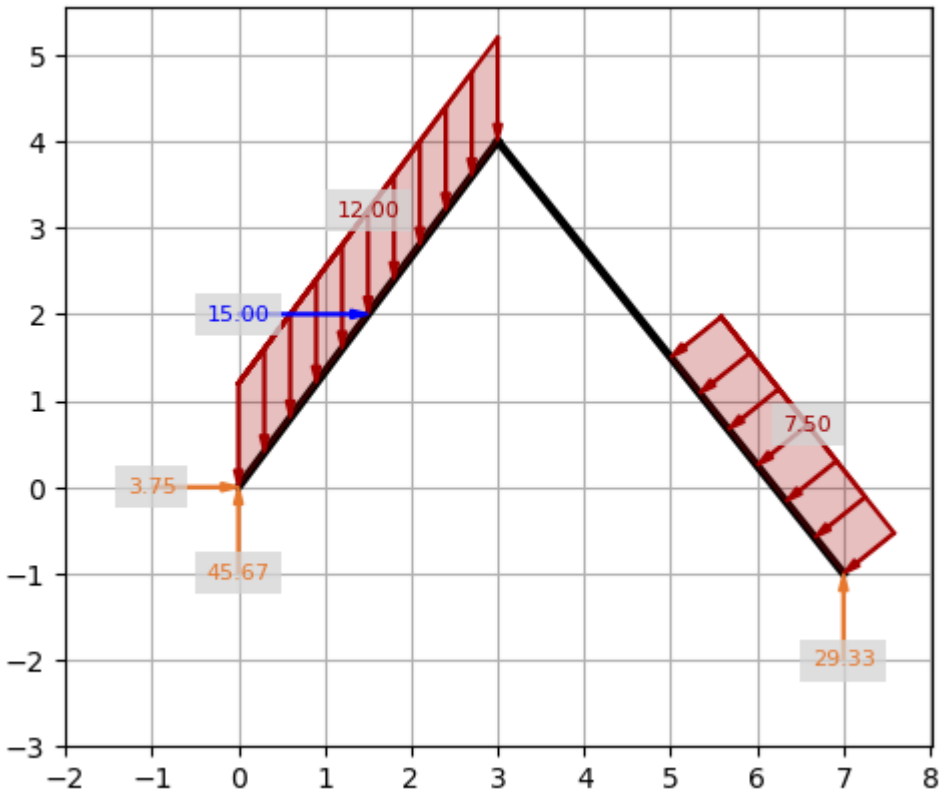
```
%pip install git+https://github.com/BorekSaheli/sympy.git@structure2d
```

```
import matplotlib.pyplot as plt
from sympy.physics.continuum_mechanics.structure2d import Structure2d
%config InlineBackend.figure_format = 'svg'
from sympy.core.symbol import symbols
E = 3e4
I = 1
A = 1e4
F = symbols("F")
s = Structure2d()
s.add_member(x1=0, y1=0, x2=3, y2=4, E=E, I=I, A=A)
s.add_member(x1=3, y1=4, x2=7, y2=-1, E=E, I=I, A=A)
s.apply_load(start_x=1.5, start_y=2, value=F, global_angle=0, order=-1)
s.apply_load(
    start_x=5,
    start_y=1.5,
    value=F / 2,
    global_angle=s.members[1].angle_deg + 270,
    order=0,
    end_x=7,
    end_y=-1,
)
s.apply_load(
    start_x=0,
    start_y=0,
    value=F * 0.8,
    global_angle=270,
    order=0,
    end_x=3,
    end_y=4,
)
Rv1 = s.apply_support(x=7, y=-1, type="roller")
Rv2, Rh2 = s.apply_support(x=0, y=0, type="pin")
s.solve_for_reaction_loads(Rv1, Rv2, Rh2)
s.draw(show_load_values=True)
```



The same plot can be generated without symbols using nummeric values.

```
E = 3e4
I = 1
A = 1e4
F = 15
s = Structure2d()
s.add_member(x1=0, y1=0, x2=3, y2=4, E=E, I=I, A=A)
s.add_member(x1=3, y1=4, x2=7, y2=-1, E=E, I=I, A=A)
s.apply_load(start_x=1.5, start_y=2, value=F, global_angle=0, order=-1)
s.apply_load(
    start_x=5,
    start_y=1.5,
    value=F / 2,
    global_angle=s.members[1].angle_deg + 270,
    order=0,
    end_x=7,
    end_y=-1,
)
s.apply_load(
    start_x=0,
    start_y=0,
    value=F * 0.8,
    global_angle=270,
    order=0,
    end_x=3,
    end_y=4,
)
Rv1 = s.apply_support(x=7, y=-1, type="roller")
Rv2, Rh2 = s.apply_support(x=0, y=0, type="pin")
s.solve_for_reaction_loads(Rv1, Rv2, Rh2)
s.draw(show_load_values=True)
```

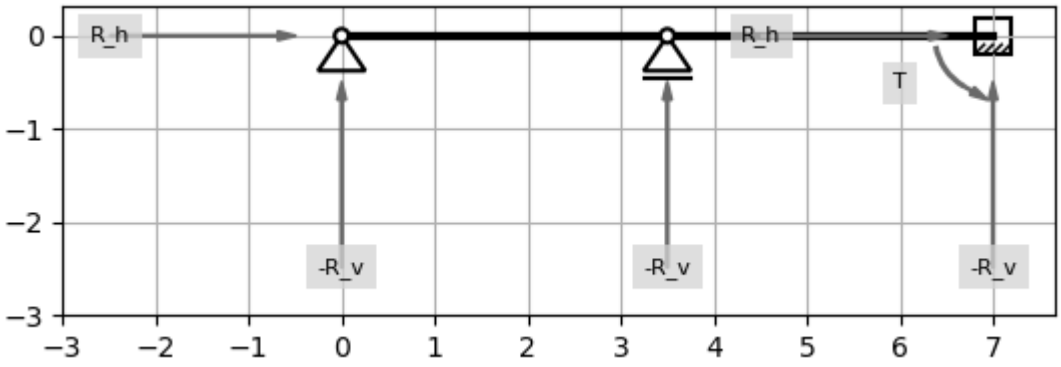

12.00
15.00
7.50
3.75
45.67
29.33

Example 4

```
%pip install git+https://github.com/BorekSaheli/sympy.git@structure2d
```

The optional parameter `draw_support_icons` will draw the following icons for the supports:

```
import matplotlib.pyplot as plt
from sympy.physics.continuum_mechanics.structure2d import Structure2d
%config InlineBackend.figure_format = 'svg'

E = 3e4
I = 1
A = 1e4
s = Structure2d()
s.add_member(x1=0, y1=0, x2=7, y2=0, E=E, I=I, A=A)
Rv1, Rh1 = s.apply_support(x=0, y=0, type="pin")
Rv2 = s.apply_support(x=7/2, y=0, type="roller")
Rv3, Rh3, T1 = s.apply_support(x=7, y=0, type="fixed")
s.draw(show_load_values=True, forced_load_size=2, draw_support_icons=True)
```



References

Baudoin, A. (2024 , Juni). Continue macaulay methode voor geknikte, vertakte en gesloten constructies. *TU Delft Education Repository*. URL: https://repository.tudelft.nl/record/uuid:f921c7d0-dde9-4d3c-9572-dd8462e9df4a

Jankie, J. (2023 , November). Macaulay's methode toegepast met invloedslijnen. *TU Delft Education Repository*. URL: http://resolver.tudelft.nl/uuid:d6869e59-57a0-4dca-9d4d-19050a0e1a89

Macaulay, W.H. (1919). A note on the deflection of beams. *Messenger of Mathematics*, *48*, 129-130.

Qadriyeh, E. (2024 , Januari). Analyse van schuine en kromme liggers met de macaulay methode. *TU Delft Education Repository*. URL: http://resolver.tudelft.nl/uuid:e82c8dd5-fbc1-46a3-b022-048d70425e2c

Saheli, B. (2024 , Oktober). Building a symbolic 2d structural analysis tool using sympy. *TU Delft Education Repository*. URL: https://resolver.tudelft.nl/uuid:e4961d2e-230f-419c-9a15-545ff0f049f8

van der Wulp, J. (2023 , June). De methode van macaulay voor tweedimensionale constructies. *TU Delft Education Repository*. URL: http://resolver.tudelft.nl/uuid:96ec934a-4520-465f-8079-01b9fad73360

van Gelder, M. (2024 , Juni). Expanding the implementation of macaulay’s method in python. *TU Delft Education Repository*. URL: https://repository.tudelft.nl/record/uuid:73518fcb-e2a6-4b63-9c6c-22607e11ebe9

Credits and License

Contents

You can refer to this book as:

> van Woudenberg, T., van der Wulp, J., Jankie, J., Qadriyeh, E., Baudoin, A., van Gelder, M., & Saheli, B. (2025) *Extension Macaulay's method*. TeachBooks/Macaulays_method

This book is also published on Zenodo DOI 10.5281/zenodo.15099577

You can refer to individual chapters or pages within this book as:

We anticipate that the content of this book will change significantly. Therefore, we recommend using the source code directly with the citation above that refers to the GitHub repository and lists the date and name of the file. Although content will be added over time, chapter titles and URL's in this book are expected to remain relatively static. However, we make no guarantee, so if it is important for you to reference a specific location within the book. For example:

> `<Title of Chapter or Page>`. In van Woudenberg, T., van der Wulp, J., Jankie, J., Qadriyeh, E., Baudoin, A., van Gelder, M., & Saheli, B. (2025) *Extension Macaulay's method*. TeachBooks/Macaulays_method (`./book/intro/` chapter, accessed `date`).

How the book is made

This book is created using open source tools: it is a JupyterBook that is written using Markdown, Jupyter notebooks and Python files to generate some figures. Additional tooling is used from the TeachBooks initiative to enhance the editing and reading experience. The files are stored on a public GitHub repository. The website can be viewed at https://teachbooks.io/Macaulays_method. Contact the authors for additional information.

About the Editors

Tom van Woudenberg

Tom van Woudenberg is a lecturer at Delft University of Technology. Equipped with a dedication to education in structural mechanics, I strive to cultivate a blended learning environment that engages students actively and rewards their efforts.

Tom graduated in 2020 at Delft University of Technology on structural optimisation. From August 2020 to August 2022, Tom worked at Amsterdam University of Applied Sciences as lecturer construction, specialised in structural mechanics. Since September 2022, Tom is working at Delft University of Technology.

Next to courses on structural mechanics, Tom (co)teaches courses on optimisation, numerical methods, data-analysis and statistics. Next to that, Tom supervises BSc- and MSc-students, partly on the research project of Macaulay's method. Furthermore, Tom is actively involved in cross-faculty collaboration on PRIMECH, TeachBooks and various digital tools for teaching like ANS and Git.

- T.R.vanWoudenberg@tudelft.nl
- tomvanw@hotmail.com
- +31152789739
- TU Delft – Civil Engineering & Geosciences - Department 3MD – Section Applied Mechanics - Room 6.45
- GitHub profile
- LinkedIn profile
- Delft University of Technology profile

Acknowledgements

Big thanks to the various colleagues (with Robert Lanzafame in particular) and teaching assistant part of the TeachBooks for developing tools and providing support to improve this book.