



## JUPYTER, TOEGANGSPOORT TOT...

De krachtige combinatie van symbolisch  
rekenen en programmeren

Levenslang Leren  
voor Leerkrachten

© 2023 KU Leuven – Faculteit Industrieel Ingenieurswetenschappen  
Uitgegeven in eigen beheer, **Dimitri Coppens**, Gebroeders De Smetstraat 1, Gent (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

# Inhoudsopgave

<b>1</b>	<b>Jupyter en SymPy: vlot van start</b>	<b>5</b>
1.1	Julia, Python en R . . . . .	5
1.2	SymPy: vertrekken van het vertrouwde domein van wiskundige symbolen . . . . .	6
1.3	Installatie op Windows . . . . .	6
1.4	Opstarten Jupyter . . . . .	7
<b>2</b>	<b>Overzichtelijk werken in Jupyter notebooks</b>	<b>8</b>
2.1	De Jupyter startpagina in een browservenster . . . . .	8
2.2	Aanmaken van een notebook . . . . .	8
2.3	Verkenning van een notebook . . . . .	9
2.4	De balk met basisbewerkingen voor notebooks . . . . .	10
2.5	Hoofdingen en ondervelingen . . . . .	10
2.6	Correct afsluiten van een notebook en Jupyter . . . . .	11
2.7	Voorstel van werkwijze voor deze bundel . . . . .	11
<b>3</b>	<b>Basisterminologie en -bouwstenen van programmeren in Python</b>	<b>13</b>
3.1	Variabelen . . . . .	13
3.1.1	Getallen: integers en floats . . . . .	13
3.1.2	Basisbewerkingen . . . . .	14
3.1.3	Tekst: Strings . . . . .	15
3.2	Gebruikersinput . . . . .	15
3.2.1	Tekst . . . . .	15
3.2.2	Getallen . . . . .	15
<b>4</b>	<b>Aan de slag met SymPy</b>	<b>17</b>
4.1	Inladen van de SymPy bibliotheek . . . . .	17
4.2	Uitdrukkingen . . . . .	17
4.3	Een voorbeeld . . . . .	18
4.4	Enkele aandachtspunten . . . . .	18
4.5	Uitdrukkingen manipuleren . . . . .	18
4.5.1	Ontbinden in factoren . . . . .	18
4.6	Concrete waarden toekennen aan variabelen . . . . .	19
4.7	Evalueren . . . . .	19
4.8	Vergelijkingen oplossen . . . . .	20
<b>5</b>	<b>Enkele functies en hun grafieken</b>	<b>21</b>
5.1	Basisstructuur van een Matplotlib plot . . . . .	21
5.1.1	Figure, Axes en Axes . . . . .	21
5.2	Interactieve plots . . . . .	25
5.3	Veeltermfuncties . . . . .	25
5.4	Terugkerende code . . . . .	26
5.5	Lambda-notatie . . . . .	27
5.6	Schaalverdelingen en grid . . . . .	28
5.7	Horizontale en verticale lijnen, tekenbereik . . . . .	28
5.8	Rationale functies . . . . .	30
5.9	Irrationale functies . . . . .	32
5.10	Exponentiële functies . . . . .	33
5.11	Logaritmische functies . . . . .	34
5.12	Goniometrische functies . . . . .	34

<b>6</b>	<b>Calculus</b>	<b>38</b>
6.1	Limieten . . . . .	38
6.1.1	Terugkerende code . . . . .	38
6.1.2	Rekenen met oneindig . . . . .	38
6.1.3	Limieten naar $\infty$ van functies . . . . .	39
6.1.4	Limieten naar $a \in \mathbb{R}$ . . . . .	40
6.2	Afgeleiden . . . . .	41
6.2.1	Terugkerende code . . . . .	41
6.2.2	Het Derivative() commando . . . . .	41
6.3	Integralen . . . . .	45
6.4	Terugkerende code . . . . .	45
6.5	Facultatieve toepassing: Riemannintegratie . . . . .	45
<b>7</b>	<b>Matrices en vectoren</b>	<b>48</b>
7.1	Dimensies . . . . .	48
7.2	Opvragen van rijen en kolommen . . . . .	49
7.3	Rijen en kolommen toevoegen en verwijderen . . . . .	49
7.4	Bewerkingen . . . . .	50
7.5	Determinanten . . . . .	50
7.6	Interveren van een matrix . . . . .	50
7.7	Enkele shortcuts voor specifieke matrices . . . . .	50
7.8	Rijgereduceerde vorm . . . . .	51
7.8.1	Indeces . . . . .	51
7.9	Vectoren . . . . .	53
7.9.1	Terugkerende code . . . . .	53
7.9.2	Aanmaken van twee-en driedimensionale vectoren . . . . .	53
7.9.3	Bewerkingen met vectoren . . . . .	53
7.9.4	Vermenigvuldiging van vector met scalar . . . . .	53
7.9.5	Inwendig Product . . . . .	53
7.9.6	Vectornorm . . . . .	54
7.9.7	Visualisatie van tweedimensionale vectoren . . . . .	54
7.10	Visualisatie van Driedimensionale Vectoren . . . . .	55
<b>8</b>	<b>Bijlages</b>	<b>57</b>

## Introductie

### Jupyter

Ben je op zoek naar een toegankelijke en gebruiksvriendelijke manier om enkele belangrijke wiskundige principes uit te testen met een instap-programmeertaal? Wil je vlot afleiden en integreren, bewerkingen met matrices uitvoeren, toepassingen uit de goniometrie en meetkunde narekenen? Dan is Jupyter Notebook een sterke aanrader. Je werkt gewoon in je browser en je organiseert wiskundige werkbladen heel intuïtief met behulp van 'cellen'. Met behulp van het Sympy pakket kan je symbolisch rekenen met de leerstof van het vijfde en zesde middelbaar. En voor wie de smaak echt te pakken heeft, krijgt met de Numpy bibliotheek een indrukwekkende wetenschappelijke toolbox ter beschikking. Helemaal gratis.

### Relatie met de leerplannen

Het onderwerp richt zich op de aankomende leerplandoelstellingen van de derde graad, met name het onderdeel "Computationeel Denken" binnen het vak Informaticawetenschappen. Het softwarepakket biedt mogelijkheden om bepaalde leerdoelstellingen binnen de wetenschappelijke vakken en wiskunde aan te halen.

# 1 Jupyter en SymPy: vlot van start

De doelstelling van deze bundel is om met een minimum aan programmeerkennis, zo snel mogelijk wiskundige berekeningen uit te voeren in een overzichtelijk werkblad. We starten met het overlopen van het basisgereedschap dat we hiervoor inschakelen.

## 1.1 Julia, Python en R

Jupyter is een open-source project dat een interactieve omgeving biedt voor het ontwikkelen van code, het uitvoeren van analyses en het maken van documentatie. Het is vernoemd naar de drie programmeertalen die het ondersteunt: **Julia**, **Python** en **R**.

- Python is een veelzijdige programmeertaal die populair is vanwege zijn eenvoudige en leesbare syntax. Verschillende toepassingen maken gebruik van, zoals webontwikkeling, datascience en kunstmatige intelligentie. Python heeft een uitgebreide standaardbibliotheek en een grote gemeenschap van ontwikkelaars. Dit alles maakt van Python de ideale kandidaat als programmeertaal voor deze syllabus.
- Julia is ontworpen voor numerieke en wetenschappelijke berekeningen en heeft een syntax die lijkt op die van Python. Julia is vooral geschikt voor het verwerken werken met grote datasets. Julia zelf komt niet aan bod in deze tekst.
- R is een programmeertaal ontworpen voor statistische analyses en datavisualisatie. R komt evenmin aan bod.

Met Jupyter kan men code schrijven in zogenaamde 'notebooks'. Deze 'werkbladen' verschijnen in je standaardbrowser (Chrome, Firefox, ...) en ze bestaan uit heldere cellen die zowel code als tekst kunnen bevatten. Dit maakt het gemakkelijk om code uit te voeren en tegelijkertijd gedachten en analyses te documenteren. Men kan resultaten, grafieken en visuele elementen binnen het notebook bekijken, wat nuttig is bij het verkennen en presenteren van gegevens. Jupyter wordt veel gebruikt in data-analyse, wetenschappelijk onderzoek, machine learning en educatieve doeleinden. Het stelt gebruikers in staat om interactief te werken met code, visualisaties te maken en complexe analyses uit te voeren, allemaal binnen één flexibele omgeving.

Tabel 1

Programmeertaal:	Java	C++	Julia	Python	R	...
Programmeeromgeving:	Eclipse	Dreamweaver	Jupyter			...

Tabel 1: Enkele voorbeelden van programmeertalen en een geschikte ontwikkelomgeving

Zodra de gebruiker vertrouwd is met basissyntax van Python, beschikt hij dadelijk een krachtige wetenschappelijke rekenmachine ter beschikking. Complexiteit toevoegen aan de standaardmogelijkheden van Python kan o.a. door uitbreidingen met *bibliotheeken*. Wil je verdergegaan met de ondersteuning van wetenschappelijk werk via Python, kan door zich verder te verdiepen in het programmeren zelf en door de basismodule uit te breiden met bijvoorbeeld NumPy. NumPy is een krachtige wetenschappelijke tool voor *numerieke* berekeningen. Dit laatste valt niet onder de scope van deze tekst. We spitsen ons toe op de SymPy bibliotheek.


Programmeertaal	Python			
Mogelijke uitbreidingen	SymPy	NumPy	matplotlib	...
geschikt voor	symbolisch rekenen	numerieke berekeningen	grafieken tekenen	...

Tabel 2: Enkele voorbeelden van uitbreidingsbibliotheeken van Python

Lezers die zich verder willen verdiepen in Python, kunnen we volgende literatuur aanraden:



- *De programmeursleerling*<sup>1</sup>

Dit Nederlandstalige cursusboek is bedoeld om Python 3 te doceren aan studenten en middelbare scholieren die nog niet over programmeerervaring beschikken. Het bevat veel voorbeelden en oefeningen. De pdf is gratis te downloaden. In deze tekst zullen we verwijzen naar enkele specifieke hoofdstukken van dit werk, we laten dan de programmeursleerling in de kantlijn verschijnen. Het symbooltje  in de kantlijn geeft aan dat de aangehaalde materie dieper ingaat op aspecten van het programmeren. Wens je de notebooks enkel te gebruiken als geavanceerde rekenmachine, dan kan je deze paragrafen overslaan.

- *Python from the very beginning* van John Whittington
- *Learn to code by solving problems* van Daniel Zingaro

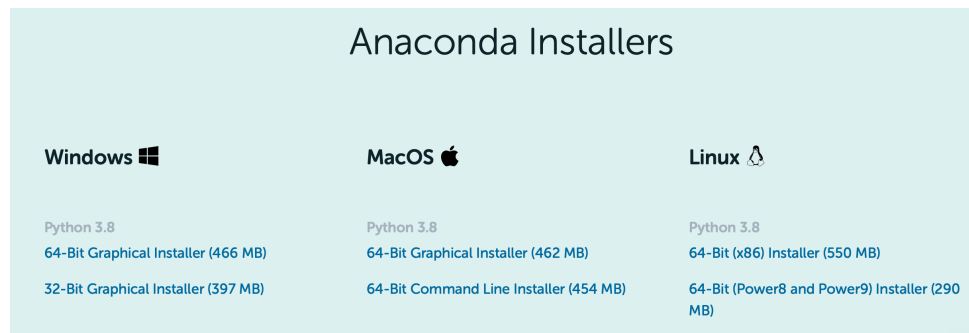
## 1.2 SymPy: vertrekken van het vertrouwde domein van wiskundige symbolen

Inladen van de SymPy-bibliotheek maakt het mogelijk om met de Python programmeertaal *symbolische* wiskundige berekeningen te maken (in tegenstelling tot de *numerieke*.) De doelstelling van SymPy is immers om te evolueren naar een volledig uitgerust *computersysteem voor symbolische algebra* (CAS). Een CAS behoudt wiskundige formules en vergelijkingen in hun symbolische vorm en is in staat om ze te manipuleren. Zo is het mogelijk om symbolisch vergelijkingen op te lossen, uitdrukkingen te vereenvoudigen, functies af te leiden,... Het biedt een breed scala aan wiskundige operaties, waaronder algebra, calculus, lineaire algebra, differentiaalvergelijkingen en nog veel meer. Voor een volledig overzicht verwijzen we naar bijlage 8

## 1.3 Installatie op Windows

Om gebruik te kunnen maken van Python in de Jupyter werkomgeving en dadelijk ook over SymPy en andere veelgebruikte bibliotheken te beschikken, is het interessant om het totaalpakket *Anaconda* te installeren. Bovendien vereenvoudigt het eventuele installatie van extra modules en het updaten van reeds geïnstalleerde software.

1. Maak gebruik van deze downloadlink<sup>2</sup>...

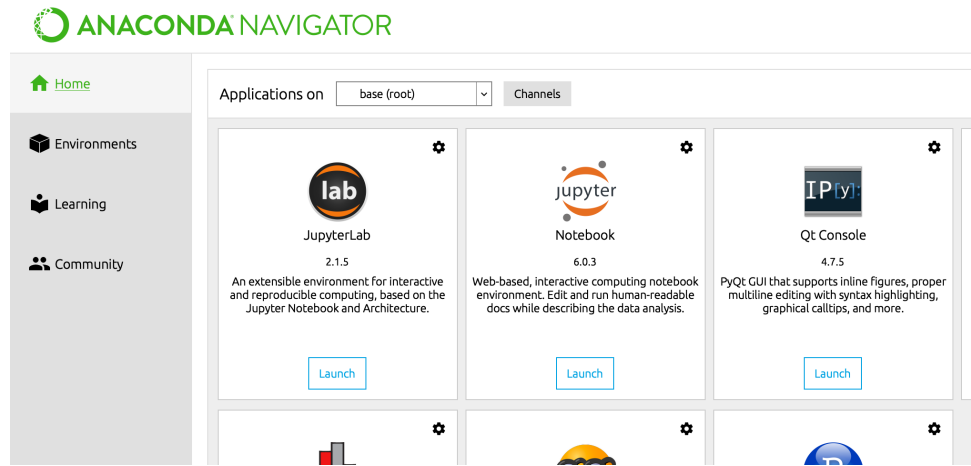


Figuur 1: Kies het geschikte installatiepakket in functie van het besturingssysteem.

2. ... kies het gewenste platform ...
3. ... en volg de richtlijnen van de installer.

<sup>1</sup><https://spronck.net/pythonbook/dutchindex.xhtml>

<sup>2</sup><https://www.anaconda.com/download>



Figuur 2: Bij opstarten van Anaconda krijgt men dit overzicht van alle onderdelen.

## 1.4 Opstarten Jupyter

Via Windows Start-menu:

1. Selecteer Anaconda Navigator
2. Een venster met het aanbod van de Anaconda onderdelen verschijnt. We kiezen voor Jupyter Notebook.
3. Gebruik windows verkenner om te navigeer naar de map met waarin je de notebooks wil opslaan.

## 2 Overzichtelijk werken in Jupyter notebooks

### 2.1 De Jupyter startpagina in een browservenster

Jupyter toont bij opstart een browservenster waarin je o.a. de bestanden zijn opgelijst in de actieve directory.

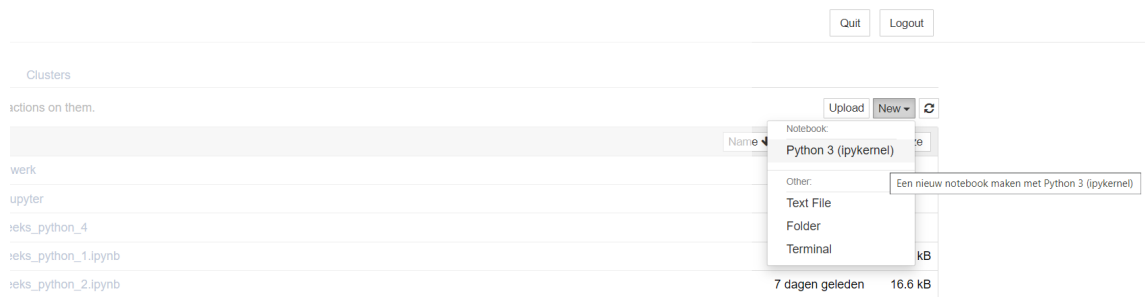
Dit is de *working directory*, nieuw aangemaakte notebook bestanden komen in deze map terecht. Je kan navigeren naar een dieperliggende map indien gewenst.



Figuur 3: Het startvenster van Jupyter in een browser.

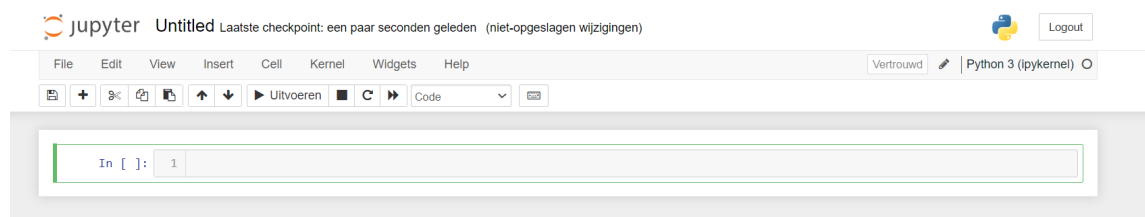
### 2.2 Aanmaken van een notebook

Een dropdownmenuutje met als titel *new* helemaal rechts biedt o.a. de opties aan om een nieuwe map of een nieuw werkblad aan te maken. Voor een nieuw werkblad kiezen we *Python 3 (ipykernel)*



Figuur 4: Een nieuw werkblad aanmaken.

Een nieuw werkblad, voorlopig zonder titel, is klaar voor gebruik. Dit werkblad is in staat om Python 3 te interpreteren. Hiervoor maakt het gebruik van de *ipython kernel*. We vermelden kort dat de kernel het eigenlijke denkwerk achter de schermen verricht. Wie wat meer achtergrond wenst, verwijzen we graag naar de documentatie van IPython <sup>3</sup>

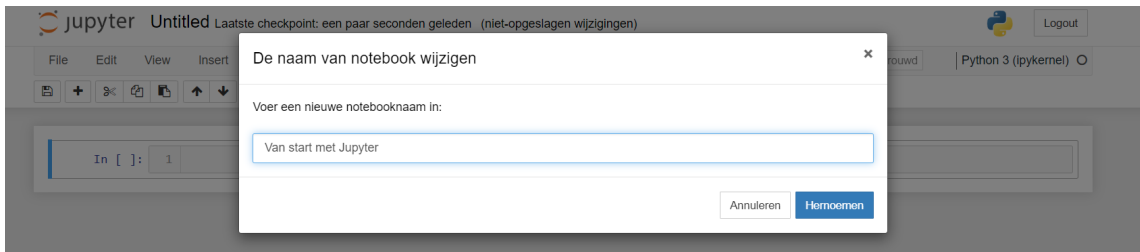


Figuur 5: Een nieuw werkblad, voorlopig nog zonder titel.

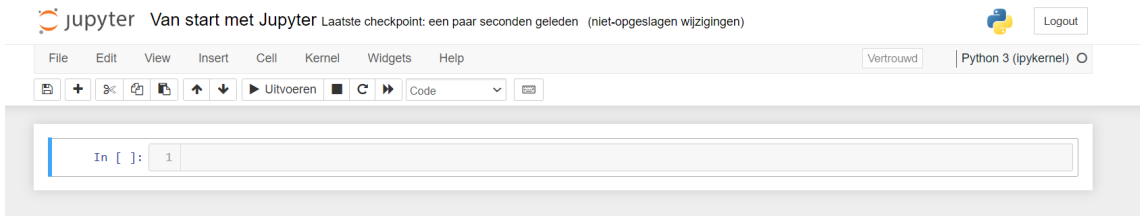
Aanpassen van de titel kan eenvoudig door *Untitled* aan te klikken en de tekst te wijzigen.

<sup>3</sup><https://ipython.org/ipython-doc/3/development/kernels.html>





Figuur 6: Aanklikken van `Untitled` opent een venstertje waarin je de naam van het werkblad kan aanpassen.

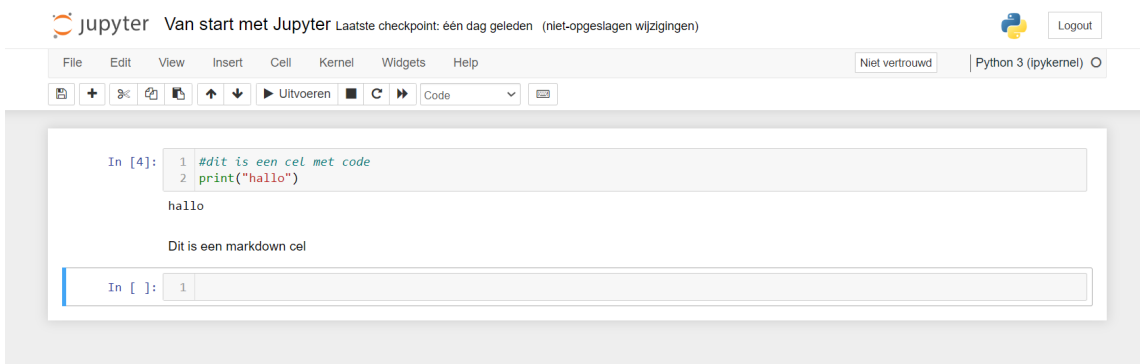


Figuur 7: De titel is aangepast.

## 2.3 Verkenning van een notebook

Een notebook of werkblad bestaat uit *cellen*. In deze bundel behandelen we *markdown* en *code* cellen:

- In een *markdown* cel kan je tekst schrijven, waarin je gerust ook hyperlinks kan opnemen. De opmaak van een markdown cel gebeurt immers aan de hand van HTML, maar ze verstaan ook tekstopmaak in  $\text{\LaTeX}$ . Daarvoor plaats je de wiskundige formules tussen dubbele dollartekens. Een markdown cel kan ook afbeeldingen bevatten. De meest eenvoudige manier om een afbeelding toe te voegen aan een markdown cel, is door de afbeelding vanuit de verkenner in de cel te slepen. Aangezien HTML een optie is, kan je ook met een url werken: ``
- *Code* cellen onderscheiden zich visueel door hun grijze achtergrond. Net links van de grijze balkjes staan steeds rechte haakjes waarin een volgnummer van uitvoering zal verschijnen.



Figuur 8: Een eerste cel met uitvoerbare code en het resultaat van het uitvoeren van de cel. Daaronder een *markdown* cel

Met het dropdown-menuutje rechtsboven, geef je aan welk type cel je wenst.



Figuur 9: Een cel met code. De uitvoer van de code verschijnt er net onder. Merk alvast op dat de eerste regel code begint met het hashtag symbooltje #. Python zal wat op deze regel staat 'negeren'. Dit is de manier om code van 'commentaar' te voorzien. Zo kan je je code van uitleg voorzien - voor anderen, maar ook voor je (toekomstige) zelf.

→ Door te dubbelklikken in een cel, kan je de inhoud ervan bewerken.

→ Een cel *uitvoeren* gebeurt door de toetsencombinatie **Ctrl** + **↵** of met behulp van de **▶** knop.

## 2.4 De balk met basisbewerkingen voor notebooks

Het icoontje met het plus teken, bovenaan links, laat toe om nieuwe cellen in te voegen. Je vindt in dit balkje ook de klassierers opslaan, knippen, plakken en kopiëren.



Figuur 10: De voornaamste bewerkingen voor cellen kregen een eigen knop in deze balk.

Met de pijltjestoesten verhuizen we een cel boven of onder andere cellen. Een cel aan het werk zetten - of, anders gezegd: de daarin opgenomen code uitvoeren - doen we met de play-knop. Het zwarte vierkantje biedt de mogelijkheid dat proces te stoppen. Dat zullen we doen als we merken dat een cel merkwaardig veel tijd nodig heeft om tot een resultaat te komen - wie weet is er een oneindige lus in ons denkwerk geslopen...

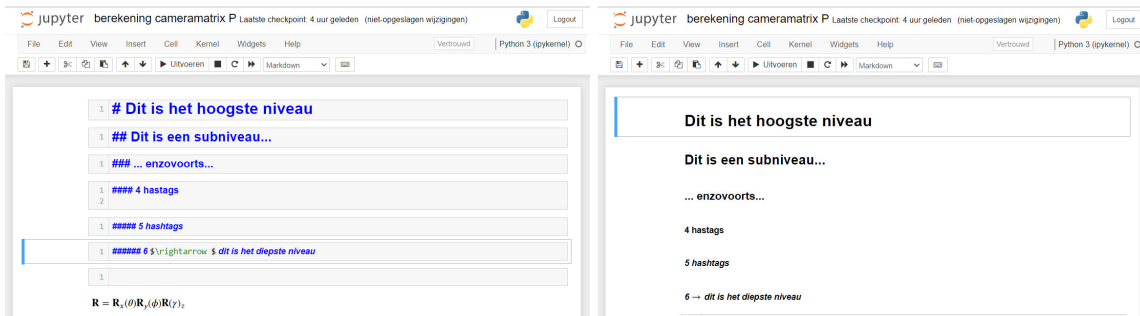
Achter de ronde pijl schuilt een belangrijk concept, waarvoor het nodig is om het concept van *variabelen* en de aan hen toegewezen opslagruimte, goed te begrijpen. Dit zal aan bod komen in het volgend hoofdstuk 3. Op dit moment kunnen we alvast verklappen dat de knop de een *tabula rasa* op vlak van geheugengebruik, zal uitvoeren.

De dubbele pijl bespaart ons veel klikwerk indien we, eens ons werkblad vele cellen bevat, alle cellen opnieuw willen laten uitvoeren. Dat gebeurt steeds van boven naar onder. Bij gebruik van deze knop verschijnt de waarschuwing dat de voorheen opgeslagen waarden, gewist zullen worden.

Zoals steeds, bestaan er voor de wat geoefende gebruiker, handige sneltoetsen en combinaties. Een overzicht is terug te vinden in de bijlage 43.

## 2.5 Hoofdingen en onderverdelingen

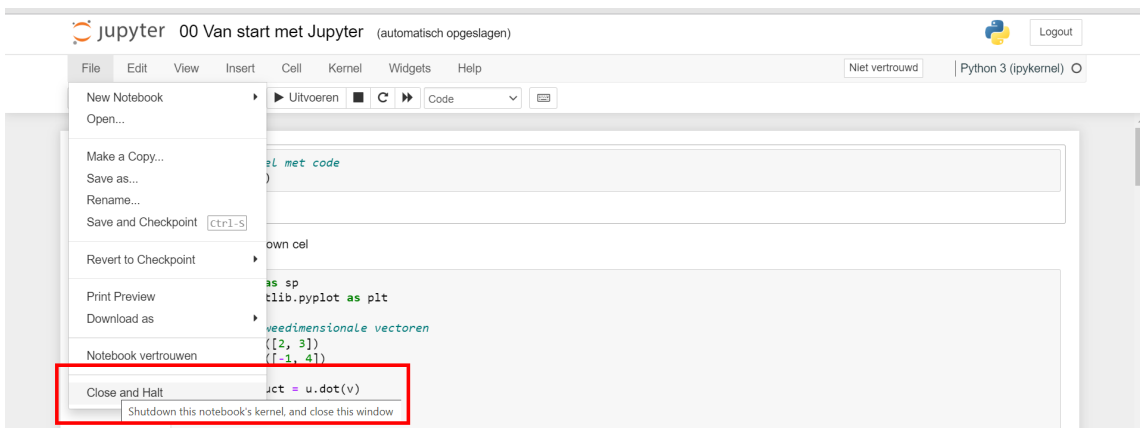
Een `markdown`-cel kan je eenvoudig omtoveren tot een hoofding met behulp van hashtag-tekens. Het aantal tekens bepaalt het niveau van de onderverdeling.



Figuur 11: Markdown cel met hoofdingen in verschillende niveaus.

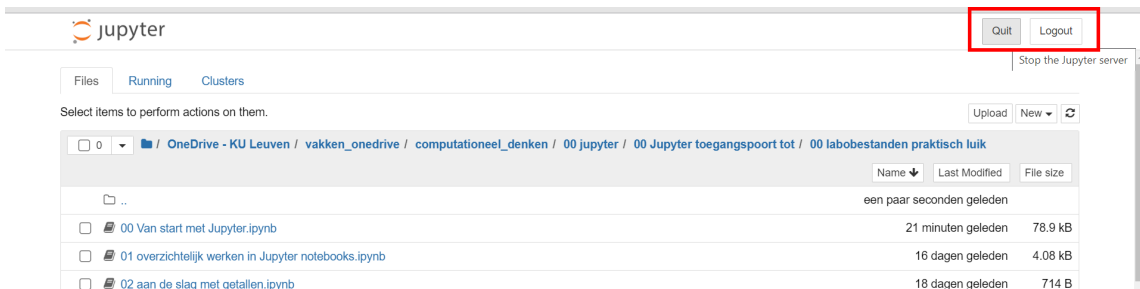
## 2.6 Correct afsluiten van een notebook en Jupyter

Zoals vermeld verricht een kernel het echte denkwerk op de achtergrond. Om de processen onder de motorkap correct te beëindigen, volstaat het niet om het browservenster of -tabblad te sluiten. Voor het afsluiten van een werkblad kies je **File** en vervolgens **Close and Halt**, zie ook figuur 12



Figuur 12: Start steeds met het afsluiten van het werkblad.

Voor het volledig afsluiten van Jupyter, gebruik je de knop rechtsboven op de overzichtspagina, zie ook figuur 13



Figuur 13: Het afsluiten van Jupyter zelf gebeurt met deze knop.

## 2.7 Voorstel van werkwijze voor deze bundel

Net zoals bij het leren fietsen, volstaat de uitleg over de techniek niet. Zolang je niet op het zadel kruipt, zal je het fietsen nooit onder de knie krijgen. We raden de lezer sterk aan om in het vervolg van deze bundel systematisch de voorbeeldcode over te nemen in een werkblad.

We raden aan om

- per hoofdstuk minstens één apart werkblad te voorzien, zo blijft het overzichtelijk.

- in de eerste cel van ieder werkblad de 'terugkerende code' waarmee ieder hoofdstuk begint, op te nemen. Figuur 14 toont dit principe aan de hand van het hoofdstuk 'Aan de slag met SymPy'.

Per hoofdstuk is er immers steeds een gelijkaardige voorbereiding nodig om over het juiste gereedschap te beschikken. Deze code staat gebundeld in een paragraaf over de voorbereidende code. Door dit op te nemen in de eerste cel van een werkblad en vervolgens in aparte cellen verder te werken, is het makkelijker om de aandacht te richten op de nieuwe lijnen code.

The screenshot shows a Jupyter Notebook interface. At the top is a toolbar with icons for adding, deleting, and moving cells, as well as buttons for 'Uitvoeren' (Run), a square icon, a circular arrow (refresh), and a double arrow (next). A dropdown menu is set to 'Code'. Below the toolbar is a red-bordered box containing the following text:

**Verscheidene hoofdstukken starten met het luik 'terugkerende code'.  
Plaats deze in de eerste code cel van het werkblad.**

Below this box is a code cell labeled 'In [4]:' containing the following code:

```
1 import sympy as sp
2 sp.init_printing()
```

Below the code cell is another code cell labeled 'In [22]:' containing the following code:

```
1 x =sp.Symbol('x')
2 y= sp.Symbol('y')
3 z =sp.Symbol('z')
4 x
```

Below the second code cell is an output cell labeled 'Out[22]:' showing the output 'x'.

Figuur 14: Start ieder werkblad met een cel waarin je de 'terugkerende code' opneemt.

## 3 Basisterminologie en -bouwstenen van programmeren in Python

In dit hoofdstuk halen we kort enkele basisbouwstenen en bijhorende terminologie aan die nodig zullen zijn in verdere uitwerkingen. Voor een diepere, en vaak vollediger, verklaring van deze termen verwijzen we graag naar de eerder vermelde Python cursussen of kan je een snelle Google zoekopdracht uitvoeren.

### 3.1 Variabelen

Een programmeervariabele kan je beschouwen als een container waarin je informatie kunt bewaren en waarnaar je, met een zelfgekozen naam, kan verwijzen in je code. De variabele kan verschillende soorten gegevens - datatypes - bevatten, zoals tekst (strings), getallen (integer of float), booleans (waar of onwaar) of complexe objecten.

Een parkeergarage met meerdere verdiepingen, voorzien van parkeerplaatsen voor motoren en personenauto's, is een mogelijke metafoor voor de opslagruimte van een computer. Veronderstel dat je

- naar iedere verdieping kan verwijzen met een *letter*letter
- en naar iedere parkeerplaats met een *getal*,

dan is het duidelijk om welke concrete parkeerplaats het gaat met een code als

$C_{15}$

De personenwagens die er hun plekjes zullen vinden, *wijzigen* naargelang auto's toekomen en vertrekken.

Het parkeren van een motor vraagt minder fysieke ruimte. Het 'toekennen' van een personenwagen aan een motorparkeerplaats zal een foutmelding geven in computerterminologie.<sup>4</sup> Afhankelijk van het soort datatype, reserveert een programma immers meer of minder geheugenruimte. Eens er is vastgelegd dat een parkeerplaats uitsluitend bestemd is voor personenwagens, is het niet meer mogelijk er een motor te parkeren.

#### 3.1.1 Getallen: integers en floats

Beschouwen we als eerste voorbeeld een variabele met als inhoud een geheel getal:

```
a = 3
```

We kennen hier aan de variabele *a* de waarde 3 toe. Er is nu geheugenruimte voorzien voor het onthouden van een gehele getalwaarde, een *integer*.

In Jupyter kunnen we actuele inhoud van de variabele opvragen:

```
a
```

geeft als output: 3

Voor het toekennen maakten we gebruik van het gelijkheidsteken, maar er is een belangrijk verschil met de wiskundige interpretatie. Het aanpassen van de waarde van een variabele brengt dit onderscheid aan het licht:

```
a = a + 1
```

Vragen we nu de actuele inhoud van de variabele op:

```
a
```

dan verkrijgen we: 4. We merken dus op dat de inhoud kan wijzigen. Dit is dan ook dé fundamentele eigenschap van een variabele. We zeggen dat de waarde wordt *overschreven*.

Tijd voor het aanmaken van een variabele met als inhoud een reëel getal, een *float*. We drukken onze wens om geheugenruimte te voorzien voor een float uit door een getal mee te geven na de 'komma'. In Python dienen we dit te doen aan de hand van een puntje dienen te noteren:

<sup>4</sup>De kritische lezer kan opmerken dat een motor wel kan parkeren op de fysieke plaats van een personenwagen, maar we gaan uit van een reglement van de parkeergarage, waarin dit verboden is. In computerpraktijk komt het inefficiënt gebruik van variabelen, en dus nodeloos verloren laten gaan van geheugen (of parkeerplaats) nog vaak voor

```
b = 5.4
```

Komt op het moment van declareren van onze variabele de waarde van het reëel getal overeen met een geheel getal, maar willen we er (later) reële waarden in opslaan, dan lossen we dit eenvoudig op als volgt:

```
c = 6.0
```

We keren nu even terug op de metafoor van de parkeerkgarage, om de in het hoofdstuk 2 beloofde toelichting te geven bij het icoontje met de ronde pijl.



Figuur 15: De icoontjes in de taakbalk bovenaan.

Eens we cellen hebben uitgevoerd en data opgeslagen zijn in variabelen, dan blijven deze bewaard 'op de achtergrond', zelfs na aanpassen van code of het deleten van cellen. Dit dient de gebruiker steeds in het achterhoofd te houden. Tijdens het proces van schrijven en aanpassen van code, kan het zijn dat in de huidige sessie alles netjes functioneert. Starten we een tijdje later Jupyter opnieuw op, dan kan het gebeuren dat er onaangename meldingen verschijnen omdat bepaalde variabelen onbekend blijken - hoewel het de vorige keer toch prima verliep?

Dit vormt één reden om voldoende frequent de ronde pijl te gebruiken. De werking ervan, kan je vergelijken met het volledig vrij maken van alle parkeerplaatsen van de garage. Een *tabula rasa* dus. Hoewel: niet alle wagens en motoren verlaten de parkeergarage, ook alle labels van de parkeergarage verdwijnen mee.<sup>5</sup>

Een cel die aan het rekenen is, maakt met een sterretje (tussen de rechte haakjes links van de cel) duidelijk dat hij aan de gebruiker even geduld vraagt. Een andere situatie die uitnodigt deze knop te gebruiken, is wanneer dit proces verdacht lang duurt. Het kan zijn dat er zich op de achtergrond conflicten voordoen met eerder opgeslagen waarden van variabelen. Zoiets durft zich wel eens manifesteren tijdens een wat langere programmeersessie. De ronde pijl kan dan tot redding dienen.

### 3.1.2 Basisbewerkingen

Een werkblad kan prima dienst doen als rekenmachine. De cel

```
1+2
```

geeft na uitvoeren, helemaal correct, als antwoord: 3.

Voer nu zelf volgende berekeningen uit:

```
1+3.5 -1+2.5 100-45 -1.1+5
```

Vermenigvuldigen doen we met \*, delen met / (voer zelf uit):

```
3*2 3.5*1.5 3/2 4/2
```

De resultaten van de laatste twee lijnen zijn respectievelijk 1.5 en 2.0. Merk op dat ook in het tweede geval, er niet een geheel getal als datatype verschijnt, hoewel dit cijfermatig wel correct zou zijn. Voor het resultaat van een deling voorziet Python standaard een float. Wens je enkel de gehele getalwaarde, dan gebruik je dubbele deeltekens: //

```
3//2
```

levert 1 op. Deze operatie staat bekend als *floor division*. Het deelt het eerste getal door het tweede en rondt vervolgens af naar beneden, ook in het geval van negatieve getallen:

<sup>5</sup>Opnieuw rekenen we op de inschikkelijkheid van de lezer om de wat manke metafoor te aanvaarden. Meer zelfs: mocht de lezer aan een geschiktere metafoor, dan bewijst hij de schrijver een grote dienst bij het updaten van deze bundel.

```
-3//2
```

zal -2 opleveren. Ben je net op zoek naar de rest van deling, dan kan je beroep doen op de modulo-operator %.

```
9%2
```

geeft inderdaad 1. Machtsverheffingen gebeuren met behulp van dubbele \* tekens:

```
2**10
```

is 1024 De exponent mag hierbij kleiner zijn dan één. Een derde wortel berekenen, kan als volgt:

```
8**(1/3)
```

De haakjes zijn nodig voor de correcte rekenvolgorde. De hiërarchie van bewerkingen staat in tabel 6

()	haakjes
x ** y	exponenten
*, /	vermenigvuldiging en deling
+, -	optellen en aftrekken

Tabel 3: Volgorde van bewerkingen.

Ga zelf het verschil na tussen het resultaat van volgende lijnen code:

```
5+5*5 (5+5)*5
```

### 3.1.3 Tekst: Strings

Tekst slaan we op in het datatype String.

```
d = "hallo"
```

Vragen we de inhoud van de variable *d* op:

```
d
```

dan merken we dat we dat de sympathieke notebook ons begroet:

```
hallo
```

## 3.2 Gebruikersinput

### 3.2.1 Tekst

A journey of a thousand miles begins with a single step. Neem je stiekem deel aan deze workshop om binnenkort de wereld van artificiële intelligentie in te duiken, dan dient ons programma in staat te zijn tot interactie met de gebruiker.

Een vriendelijke kennismaking is alvast een goede start.

```
naam = input("Halo. Hoe heet u?")
print("Aangename kennismaking, ", naam)
```

### 3.2.2 Getallen

Wensen we getalwaarden op te vragen, dan *typecasten* we de gebruikersinvoer naar integers. Standaard beschouwt Python de invoer immers als Strings. We wensen deze tekstinformatie om te zetten (te 'casten') naar een ander datatype. Onderstaande code toont hoe dit in zijn werk gaat. Ze maakt hiervoor gebruik van het `type()` commando, dat weergeeft welk datatype de in de variabele in kwestie zijn onderkomen vindt.

```
ingave_gebruikersinput = input("Hoe oud bent u?")
print(type(ingave_gebruikersinput))
leeftijd = int(ingave_gebruikersinput)
print(type(leeftijd))
```

Vaak zal men in één lijn zowel de gebruikersinput opvragen én dadelijk omzetten naar een getalwaarde:

```
leeftijd = int(input("Hoe oud bent u?"))
print(type(leeftijd))
```

1. Schrijf een programma dat aan de gebruiker zijn geboortejaar vraagt en berekent welke je leeftijd de gebruiker in 2030 zal hebben.

**Voorbeeld**

Geef jouw geboortejaar: 1999  
In 2030 ben je 31 jaar.

2. **Gemiddelde**

- Lees 3 gehele getallen in en bereken het gemiddelde.  
Het resultaat moet een reëel getal zijn.
- Laat het resultaat een tweede keer zien, maar nu op 2 cijfers na de komma

3. **Zet een tijd om naar seconden**

Lees een natuurlijk getal in (seconden) en zet om in  
- dagen, - uren, - minuten en - resterende seconden.

**Voorbeeld**

Geef een tijd in seconden:1000000  
Dagen: 11  
Uren: 13  
Minuten: 46  
Seconden: 40



## 4 Aan de slag met SymPy

### 4.1 Inladen van de SymPy bibliotheek

De code `import sympy as sp` zorgt ervoor dat we met de verkorte notatie `sp` gevolgd door een puntje, de commando's van de SymPy bibliotheek kunnen aanspreken.

De functie `sp.init_printing()` uit de SymPy bibliotheek activeert een heldere weergave van wiskundige uitdrukkingen. Ieder notebook start dus best met een cel die het volgende bevat:

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

#### Opmerking

In principe kan je dadelijk ook alle functies van de bibliotheek ter beschikking stellen m.b.v. een sterretje: `from sympy import *`. De prefix `sp.` is dan niet meer nodig. Nu zal het vaak voorkomen dat je functies uit verschillende bibliotheken zal inladen. Zo zullen we in deze bundel ook gebruik maken van de Numpy- en Matplotlib-bibliotheken. Het is al snel niet meer duidelijk uit welke bibliotheken de functies ontleend zijn. Een eenvoudig voorbeeld is dat van de wortel-functie: zowel Numpy als Sympy beschikken over de functie `sqrt()`. De werkwijze van de functie kan verschillen afhankelijk van de implementatie in de specifieke bibliotheek. De prefix is dan handig om steeds zicht te hebben over de afkomst van de functie.

Anderzijds zal je ook vaak op zoek gaan naar (extra) informatie over functies. Ook dan is het nodig om te weten in welke bibliotheek je dient te duiken.

### 4.2 Uitdrukkingen

Om symbolisch te kunnen rekenen, zullen we dit voorbeeld starten we met het aanmaken van drie veranderlijken `x`, `y` en `z`. Dit gebeurt als volgt:

```
x = sp.Symbol('x')
y = sp.Symbol('y')
z = sp.Symbol('z')
```

We kunnen dit ook bundelen in één regel code:

```
u, v, w = sp.symbols('u v w')
```

Een aandachtspunt is het gebruik van de hoofdletter voor `Symbol` versus de kleine letter in `symbols`. Vanaf nu kunnen we de basisbewerkingen uit tabel 4 toepassen op de veranderlijken.

voorbeeld		
optelling	+	$x + y$
afbrekking	-	$z - x$
deling	/	$x/z$
vermenigvuldiging	*	$y * z$
machtsverheffing	**	$x * 2$ (voor $x^2$ )

Tabel 4: De basisoperatoren

We mogen deze symbolen ook combineren met getallen, zoals te zien in dit voorbeeld:

```
uitdrukking = x + 2*y
```

SymPy gebruikt expressies om wiskundige uitdrukkingen in op te slaan. Expressies kunnen dus bestaan uit symbolen en getallen gecombineerd met behulp van wiskundige operatoren zoals optellen, aftrekken, vermenigvuldigen en delen. Vervolgens verder rekenen met de aangemaakte expressies is dan mogelijk: de resulterende expressie `expr` kan verder worden gemanipuleerd en symbolisch of numeriek worden geëvalueerd met behulp van verschillende SymPy-functies.

Mits het declareren van Griekse letters al symbolen, kan SymPy er prima mee overweg.

```
alpha, omega = sp.symbols('alpha omega')
alpha
```

toont dan heel mooi  $\alpha$

Een overzicht van Griekse letters en hun code vind je in bijlage 8.

### 4.3 Een voorbeeld

Veronderstel dat we formule 4.3 willen invoeren:

$$\frac{\sqrt{2}e^{-\frac{(-\mu+x)^2}{2\sigma^2}}}{2\sqrt{\pi}\sqrt{\sigma^2}} \quad (1)$$

dan hebben we het één en ander nodig.

→ Voor  $e^x$  gebruik je `sp.exp(x)`.

→ De vierkantswortel verkrijg je met `sp.sqrt`.

→ In  $\text{\LaTeX}$  een aantal programmeertalen is  $\wedge$  het symbool voor machtsverheffing. SymPy volgt de Python conventie van de dubbele sterretjes `**`:

```
x, y = sp.symbols('x, y')
x**y
```

geeft ons  $x^y$ .

Het  $\wedge$  symbool is immers gereserveerd voor de XOR operator<sup>6</sup>:  $\hat{x}y$  geeft  $x \vee y$

De uitdrukking 4.3 zal je verkrijgen met:

```
sigma, mu, x = sp.symbols('sigma mu x')
1/sp.sqrt((2*pi*sigma**2))*sp.exp(-(x-mu)**2/(2*sigma**2))
```

### 4.4 Enkele aandachtspunten

Python zal bij de deling van twee gehele getallen, een reëel getal als resultaat geven.

$1/2$  zal dus  $0.5$  opleveren. Om met rationale getallen te werken, bestaat het commando `sp.Rational(1,2)` of `sp.S(1)/2`. In combinatie met symbolen zou

```
x + sp.S(1/2)
```

als output  $x+0.5$  genereren. Dit in tegenstelling tot

```
x + sp.Rational(1,2)
```

wat  $x + \frac{1}{2}$  toont.

### 4.5 Uitdrukkingen manipuleren

#### 4.5.1 Ontbinden in factoren

Het commando `factor` zal een uitdrukking trachten te ontbinden in factoren:

```
uitdrukking = x**2 - y**2
sp.factor(uitdrukking)
```

levert  $(x - y) * (x + y)$  Met het commando `expand` kunnen we de omgekeerde weg afleggen:

<sup>6</sup>De XOR-operator (ook wel exclusieve OF-operator genoemd) is een logische operator die wordt gebruikt om te controleren of twee beweringen verschillend zijn. Het resultaat van XOR is waar als precies één van de beweringen waar is, en onwaar als beide beweringen hetzelfde zijn (beide waar of beide onwaar).

```
uitdrukking = x**2 - y**2
factoren = sp.factor(uitdrukking)
expand(factoren)
```

levert  $x^2 - y^2$

Probeer beide richtingen uit aan de hand van de formule

$$(x - y)^3 = x^3 - 3x^2y + 3xy^2 - y^3$$

## 4.6 Concrete waarden toekennen aan variabelen

Om een uitdrukking als

```
x = sp.symbol('x')
uitdrukking = x + 1
```

concreet aan het rekenen te zetten door  $x$  de waarde 2 te laten aannemen, gebruiken we `subs`:

```
uitdrukking.subs(x,2)
```

De notebook geeft dan het resultaat 3 weer.

Veronderstel nu dat we in onderstaande vergelijking met twee onbekenden  $x$  en  $y$ :  $x^2 + 2xy + y^2$  de concrete waarden 1 en 2 willen toekennen aan  $x$  en  $y$  en vervolgens het resultaat willen berekenen. We voeren dan volgende stappen uit: <sup>7</sup>

```
uitdrukking = x**2 + 2*x*y + y**2
resultaat = uitdrukking.subs({x:1, y:2})
```

Geven we `res` mee aan een cel en voeren we de cel uit:

```
resultaat
```

dan verkrijgen we 9

Met commando `simplify` kan Sympy het één en ander vereenvoudigen, zoals te zien in volgende voorbeelden:

```
x = sp.symbol('x')
sp.simplify(sp.sin(x)**2 + sp.cos(x)**2)
```

Dit geeft als resultaat 1

De uitkomst van de regel

```
sp.simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
```

zal  $x-1$  zijn.

## 4.7 Evalueren

SymPy blijft trouw aan de symbolische weergave, zoals te zien in de output van deze cel:

```
sp.sqrt(2)
```

$$\sqrt{2}$$

Wensen we de concrete getalwaarde op te vragen, dan roepen we de functie `evalf` op. We geven mee hoeveel getallen na de komma dienen te verschijnen.

```
sp.sqrt(2).evalf(7)
```

1.414214

Bereken de waarde van  $\pi$  tot 100 cijfers na de komma.

<sup>7</sup>De syntax van hetgeen tussen de haakjes wordt meegegeven aan de functie `subs`, is die van een Python *dictionary*

## 4.8 Vergelijkingen oplossen

Een vergelijking als  $x^2 = 4$  noteer je als

```
sp.Eq(x**2,4)
```

Om deze op te lossen gebruik je het `solve` commando. Een overzicht van de mogelijkheden van dit commando vind je in bijlage 10. Hiermee berekent SymPy een lijst met (symbolische) oplossingen. Zo zal

```
sp.solve(sp.Eq(x**2,4),x)
```

de lijst `[-2,2]` teruggeven.

Het `solve`-commando kan ook rechtstreeks op een vergelijking inwerken:

```
sp.solve(x**2+3*x-3,x)
```

Merk op dat SymPy zelf interpreteert dat het rechterlid van deze vierkantsvergelijking 0 is.

### Een lineair stelsel oplossen

Ingeven van een  $2 \times 2$  stelsel

$$2x + 3y = 6$$

$$3x + 2y = 12$$

gaat als volgt:

```
x = Symbol('x')
y = sp.Symbol('y')
vgl1 = 2 * x + 3 * y - 6
vgl2 = 3 * x + 2 * y - 12
```

We geven beide vergelijkingen mee, tussen extra haakjes, aan het commando `solve`. We geven dadelijk ook de optie `dict = True` mee.

```
sp.solve((vgl1, vgl2), dict=True)
```

Zo krijgen we als resultaat `[x:24/5,y: -6/5]`

De optie `dict=True` in de `solve`-opdracht wordt gebruikt om de uitvoer in de vorm van een Python *dictionary* te krijgen.

Zonder vermelding van deze optie, is de terugkeerwaarde van de `solve` opdracht een *tuple*s met oplossingen.



Inzicht krijgen in de datastructuren *tuple* en *dictionary* vraagt wat meer uitleg, daarvoor verwijzen we graag naar de hoofdstukken 11 (Tuples) en 13 (Dictionaries) van het boek *De Programmeursleerling* van Pieter Spronck<sup>8</sup>.

De optie `dict=True` biedt als voordeel dat we het antwoord in deze vorm `[y: -6/5, x:24/5]` verkrijgen, wat zowel intuïtief makkelijker interpreteerbaar is én het mogelijk wordt de waarden voor *y* en *x* makkelijk op te vragen tijdens verdere berekeningen.

<sup>8</sup><https://www.spronck.net/pythonbook/pythonboek.pdf>

## 5 Enkele functies en hun grafieken

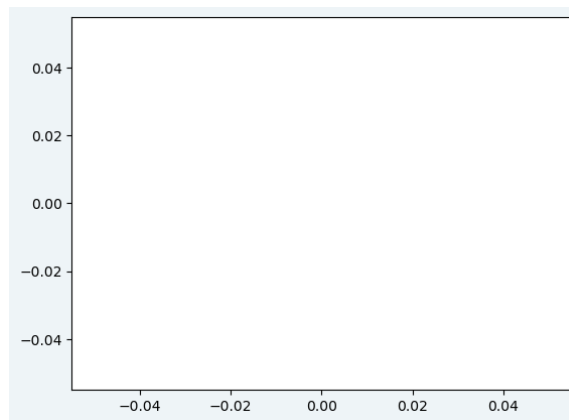
SymPy beschikt over een plotbibliotheek, die verderbouwt op de onderliggende, oudere en veel uitgebreidere Matplotlib van Python. Beide bibliotheken vertonen voor- en nadelen. Aangezien we in deze bundel ook aandacht zullen besteden aan het visualiseren van vectoren en matrices, en de jongere SymPy bibliotheek op het ogenblik van opmaak van deze tekst nog onvoldoende hiervoor is uitgerust, opteren we voor Matplotlib. Matplotlib gebruiken gaat hand in hand met de bijzonder sterke en uitgebreide NumPy-bibliotheek. Zoals eerder vermeld, is die gericht op numerieke berekeningen.

We starten met het inladen van de matplotlib-bibliotheek en kennen die dadelijk de alias `plt` toe. Dat spaart wat typewerk en houdt de code overzichtelijk.

```
import matplotlib as plt
```

### 5.1 Basisstructuur van een Matplotlib plot

#### 5.1.1 Figure, Axes en Axes



Figuur 16: Een lichtblauw canvas voorzien van een assenstelsel.

Figuur 16 toont de twee hoofdrolspelers in dit verhaal. Op het hoogste niveau bevindt zich de `figure`. Dit is het hele venster of canvas waarop je één of meerdere grafieken kunt tekenen. We gaven dit canvas een lichtblauwe kleur om het onderscheid met het assenstelsel te verduidelijken.

De code om deze figuur aan te maken, luidt:

```
import matplotlib.pyplot as plt
lichtblauw = "#EEF4F7"
fig = plt.figure(facecolor=lichtblauw)
ax = fig.add_subplot(111)
ax.plot()
plt.show()
```

Het aanmaken van de omvattende figuur voorzien van één assenstelsel kan ook met één lijn code:

```
fig, ax = plt.subplots(facecolor=lichtblauw)
```



Het puntje vervult hier een belangrijke taak. Het is de manier om **plt** 'aan te spreken'. Na het puntje volgt een 'commando', wat in programmeertermen een 'methode' of 'functie' genoemd. Een methode is te herkennen aan de ronde haakjes op het einde. Zo'n functie vertoont wel overeenkomsten met het wiskundig begrip, maar tevens belangrijke verschillen. We gaan hier niet dieper op in, maar in paragraaf 5.5 lichten we wel toe hoe we *zo dicht mogelijk* bij het wiskundige concept kunnen aanleunen. We vermelden wel dat het mogelijk is om al dan niet bepaalde gegevens aan het commando 'mee te geven', omdat een commando mogelijks 'materiaal nodig heeft' om zijn werk te kunnen doen. Dit doen we dan door het materiaal 'mee te geven' tussen de ronde haakjes. Heeft een commando geen externe informatie nodig, dan blijven de lege ronde haakjes wel noodzakelijk: het zijn de haakjes zelf die aangeven dat het hier over een uit te voeren instructie gaat.

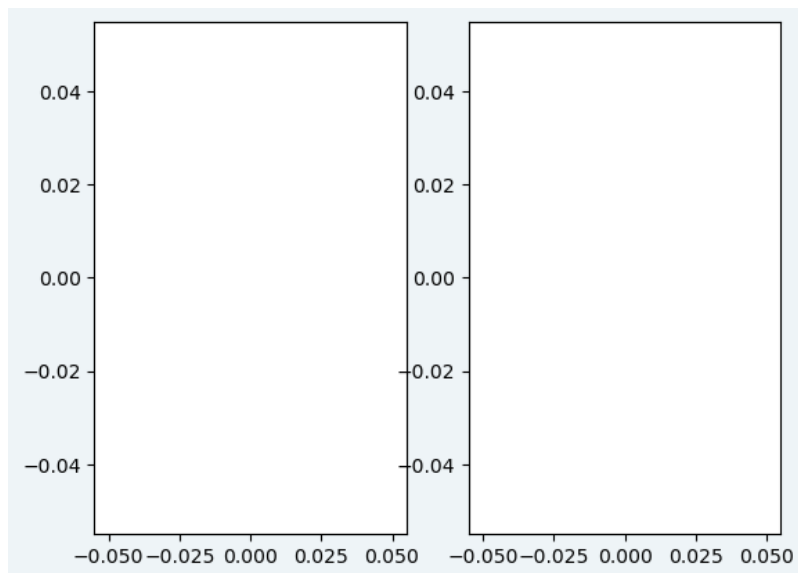


Om meerdere assenstelsels aan te maken, is een basiskennis van het `list`-datatype een meerwaarde. We verwijzen hiervoor naar hoofdstuk 12 (Lists) van het boek *De Programmeursleerling*<sup>9</sup> van Pieter Spronck.

Om vlot vooruit te kunnen gaan, voeren we in onderstaand voorbeeld de twee overbodige variabelen `eerste_assenstelsel` en `tweede_assenstelsel` in, om een intuïtief begrip van lists aan te wakkeren.

```
import matplotlib.pyplot as plt
lichtblauw = "#EEF4F7"
aantal_rijen = 1
aantal_kolommen = 2
fig, axes = plt.subplots(aantal_rijen, aantal_kolommen, facecolor=lichtblauw)
eerste_assenstelsel = axes[0]
tweede_assenstelsel = axes[1]
eerste_assenstelsel.plot()
tweede_assenstelsel.plot()
plt.show()
```

Deze code levert figuur 17



Figuur 17: Twee lege assenstelsels

Net dezelfde figuur kan gegenereerd worden zonder de omweg van de extra variabelen:

```
import matplotlib.pyplot as plt
lichtblauw = "#EEF4F7"
aantal_rijen = 1
aantal_kolommen = 2
fig, axes = plt.subplots(aantal_rijen, aantal_kolommen,
                        facecolor=lichtblauw)
axes[0].plot()
axes[1].plot()
```



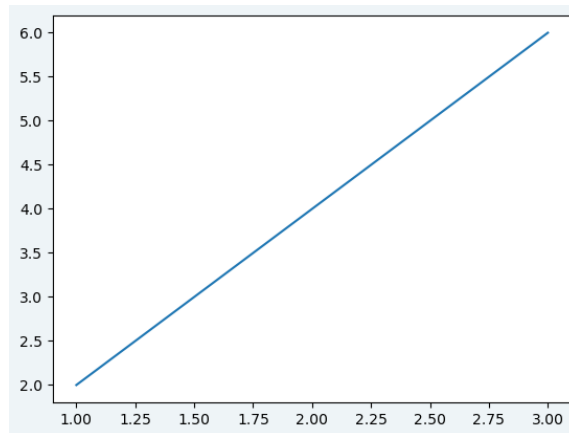
De `axes` vormen een 'lijst' met meerdere objecten. Lijsten zijn te herkennen aan rechte haken `[]`. We kunnen naar objecten in de lijst aanspreken met behulp van 'indexen', te starten van nul voor het eerste element: `axes[0]` geeft toegang tot het eerste object uit de `axes`-lijst, `axes[1]` tot het tweede. Vooral nog viel er in het assenstelsel weinig te beleven. We brengen hier verandering in door een rechte tot leven te roepen. Het tekenen van een grafiek is ook volledig gebouwd op het `list`-datatype.

<sup>9</sup><https://www.spronck.net/pythonbook/pythonboek.pdf>

We hopen met onderstaande stapsgewijze opbouw, het intuïtief interpreteren van lijsten, verder te stimuleren.

Bekijk in volgende voorbeeld hoe er met behulp van lijsten, waarden op de x-as en y-as worden meegegeven:

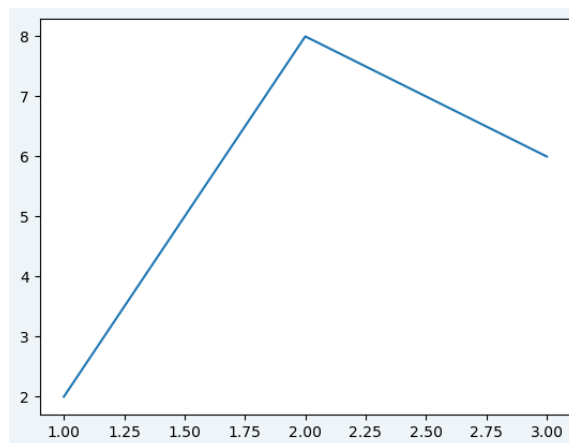
```
import matplotlib.pyplot as plt
lichtblauw = "#EEF4F7"
fig, ax = plt.subplots(facecolor=lichtblauw)
x_waarden = [1,2,3]
y_waarden = [2,4,6]
ax.plot(x_waarden,y_waarden)
plt.show()
```



Figuur 18: Een rechte  $y = 2x$

De y-waarden zijn systematisch het dubbele van de x-waarden, wat resulteert in de rechte. Bestudeer vervolgens de kleine aanpassing in dit voorbeeld:

```
import matplotlib.pyplot as plt
x_waarden = [1,2,3]
y_waarden = [2,8,6]
plt.plot(x_waarden,y_waarden)
plt.show()
```



Figuur 19: Een knik

De wijziging doet zich voor bij de tweede y-waarde in de lijst: die bedraagt nu 8 i.p.v. 4.

+	kruisje
de letter 'o'	cirkeltje
*	sterretje
s	vierkantje
1,2,3,4 ...	'y'-vormig symbootje onder verschillende rotatiehoeken

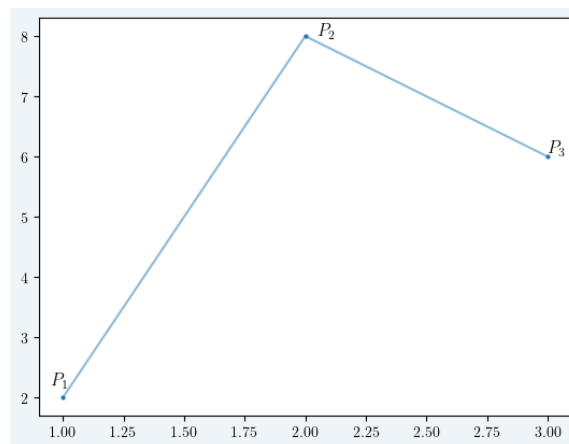
Tabel 5: Mogelijke markers

Dit verduidelijkt de werkwijze van de plot-functie: die zal uit de lijst met x- en y-waarden koppels opbouwen waartussen hij lijnstukken zal tekenen. In dit geval dus:

- Een eerste koppel (1,2), laten we dit  $P_1$  noemen - we beschouwen de koppels immers als coördinaten van dit tweedimensionale punt.
- Een tweede punt met  $P_2$  met coördinaten (2, 8)
- Tot slot  $P_3(3,6)$

Matplotlib tekent vervolgens de lijnstukken  $P_1 P_2$  en  $P_2 P_3$ .

Dit kan duidelijker. Bovenstaande uitleg vraagt om een figuur waarop de punten  $P_1$ ,  $P_2$  en  $P_3$  op staan aangeduid. Tijd om te bestuderen hoe we aantekeningen kunnen maken, dus.








Figuur 20: Aantekeningen op een figuur.

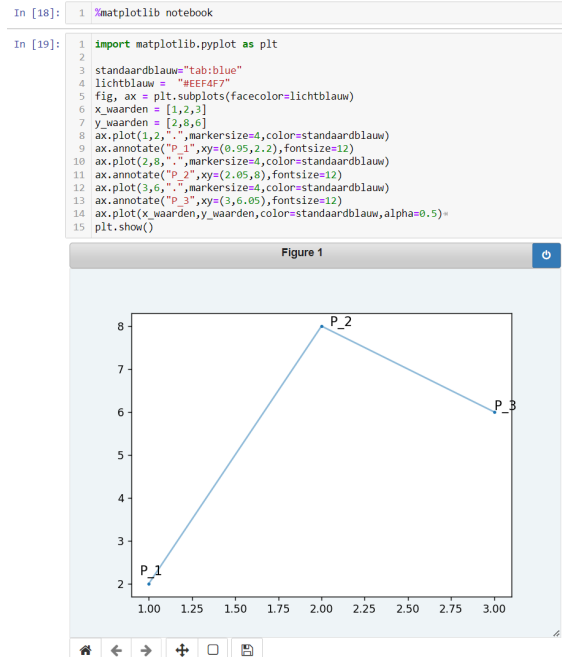
Met de regel `ax.plot(1,2,".",markersize = 4, color= standaardblauw)` verzoeken we om een blauwe puntje met grootte '4' te tekenen op de positie (1,2). In plaats van een puntje kan je de regel `ax.annotate("P_1",markersize = 4, color= standaardblauw)` verzoeken we om een blauwe puntje met grootte '4' te tekenen op de positie (1,2).

```
import matplotlib.pyplot as plt
%plt.rcParams['text.usetex'] = True\\
standaardblauw="tab:blue"
lichtblauw = "#EEF4F7"
fig, ax = plt.subplots(facecolor=lichtblauw)
x_waarden = [1,2,3]
y_waarden = [2,8,6]
ax.plot(1,2,".",markersize=4,color=standaardblauw)
ax.annotate("P_1",xy=(0.95,2.2), fontsize=12)
ax.plot(2,8,".",markersize=4,color=standaardblauw)
ax.annotate("P_2",xy=(2.05,8), fontsize=12)
ax.plot(3,6,".",markersize=4,color=standaardblauw)
ax.annotate("P_3",xy=(3,6.05), fontsize=12)
ax.plot(x_waarden,y_waarden,color=standaardblauw,alpha=0.5)
plt.show()
```



## 5.2 Interactieve plots

Wanneer je `%matplotlib notebook` uitvoert in een Jupyter Notebook-cel, genereert Matplotlib interactieve plots. Je kan dan o.a. in- en uitzoomen en de grafiek verslepen m.b.v. het  icoontje. Je krijgt eveneens de  $x$ - en  $y$ -coördinaten te zien van de cursor, wanneer deze zich boven de grafiek bevindt. Met  icoontje kan je een kadertje tekenen waarnaar je wenst in te zoomen. De pijlen  en  kan je interpreteren zoals de pijlen van een browser: je krijgt de vorige en volgende weergaven te zien. Met de  knop kan je steeds terugkeren naar de initiële weergave.



Figuur 21: Een interactieve grafiek na uitvoeren van het `%matplotlib notebook` in een cel.

## 5.3 Veeltermfuncties

We voelen er uiteraard weinig voor om op deze manier functievoorschriften in te voeren. Een goede strategie is om aan Python te vragen om een partitie te maken van interval op de  $x$ -as. Daarvoor doen we beroep op de functie `linspace` uit de Numpy-bibliotheek. We laden deze in en maken duidelijk dat we met `np` naar deze bibliotheek zullen verwijzen.

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

Het commando `linspace` verwacht drie parameters: de ondergrens en-bovengrens van het gesloten interval en het aantal gewenste onderverdelingen. Met

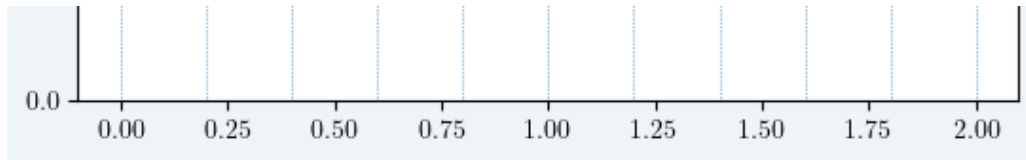
```
x = np.linspace(0,20,11)
x
```

verkrijgen we een datatype dat sterk lijkt <sup>10</sup> op een standaard Python lijst:

```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.])
```

Figuur 22 geeft deze opdeling grafisch weer.

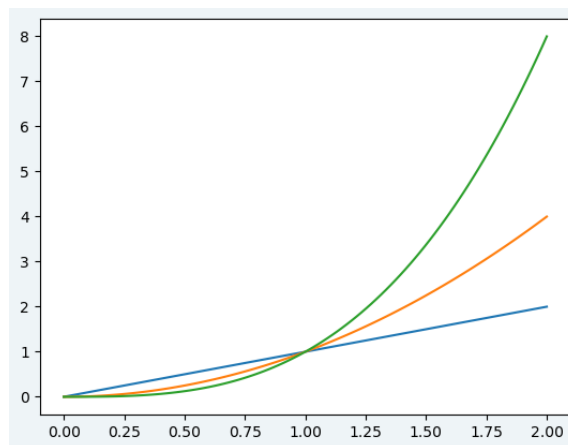
<sup>10</sup>Ondanks de schijnbare overeenkomst, dienen we te vermelden dat we hier te maken hebben met het array datatype uit de numpy bibliotheek. Die vertoont twee belangrijke verschillen met een standaard Python lijst. Enerzijds kunnen lijsten verschillende soorten datatypes bevatten, numpy arrays kunnen dat niet. We zeggen dat lijsten heterogeen zijn, arrays homogeen. Anderzijds kan het aantal elementen in een lijst wijzigen, in een array ligt dat aantal vast.



Figuur 22: Verticale lijnen ter hoogte van ieder element uit de array  
[0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0 ]

In onderstaande is het functievoorschrift rechtstreeks in het commando plot opgenomen. Per element uit de array met x-waarden, zal de corresponderende y-waarde berekend worden. Merk ook op dat je gerust aan een axis mag vragen om meerdere grafieken samen te plotten in één assenstelsel.

```
import matplotlib.pyplot as plt
import numpy as np
lichtblauw = "#EEF4F7"
fig, ax = plt.subplots(facecolor=lichtblauw)
x = np.linspace(0,2,100)
ax.plot(x,x)
ax.plot(x,x**2)
ax.plot(x,x**3)
plt.show()
```



Figuur 23: Merk op dat Matplotlib spontaan andere kleuren kiest bij het tekenen van meerdere grafieken binnen één figuur.

## 5.4 Terugkerende code

De code voor de voorbeeld grafieken in het vervolg van dit hoofdstuk, begint steeds met dezelfde lijnen. Om de aandacht van de lezer sneller te richten op de de relevante nieuwe code, zullen we de grijze kaders met code laten starten met [code §5.4], als verwijzing naar van volgende lijnen:

```
import matplotlib.pyplot as plt
import numpy as np
standaardblauw="tab:blue"
lichtblauw = "#EEF4F7"
fig, ax = plt.subplots(facecolor=lichtblauw)
ondergrens_interval = -10
bovengrens_interval = 10
onderverdeling = 1000
x= np.linspace(ondergrens_interval ,
               bovengrens_interval , onderverdeling )
```

Gaat het over een figuur met twee assenstelsels, dan betreft het deze code:

```
import matplotlib.pyplot as plt
import numpy as np
lichtblauw = "#EEF4F7"
aantal_rijen= 1
aantal_kolommen = 2
fig , axes= plt.subplots(aantal_rijen ,aantal_kolommen ,
                          facecolor=lichtblauw)
eerste_assenstelsel = axes[0]
tweede_assenstelsel = axes[1]
ondergrens_interval = -10
bovengrens_interval = 10
onderverdeling = 1000
x= np.linspace(ondergrens_interval ,
               bovengrens_interval , onderverdeling )
```

Mocht er toch een andere onverdeeling van het interval op de  $x$  -  $as$  van toepassing zijn, dan wordt deze expliciet vermeld ter vervanging van de laatste vier lijnen:

```
ondergrens_interval = -10
bovengrens_interval = 10
onderverdeling = 1000
x= np.linspace(ondergrens_interval , bovengrens_interval ,
               onderverdeling )
```

## 5.5 Lambda-notatie



We raden de lezer aan om stil te staan bij het onderscheid tussen het begrip 'functie' in wiskundige context en in de context van het programmeren, aan de hand van hoofdstuk 8 (Functies) van het boek *De Programmeursleerling*<sup>11</sup> van Pieter Spronck.



De laatste paragraaf behandelt de zogenaamde '*anonieme functies*'. Hier komt de lambda-notatie aan bod, die het mogelijk maakt om het verschil tussen de wiskundige en de programmeer notatie van een functie, zo klein mogelijk te houden. Hernemen we de derdegraadsfunctie uit vorig voorbeeld. Om terug te kunnen vallen op de meest intuïtieve vorm  $y = f(x)$ , is de stap `f = lambda x: x**3` nodig.

[code §5.4]

```
x = np.linspace(0,2,100)
f = lambda x: x**3
y = f(x)
ax.plot(x,y)
```

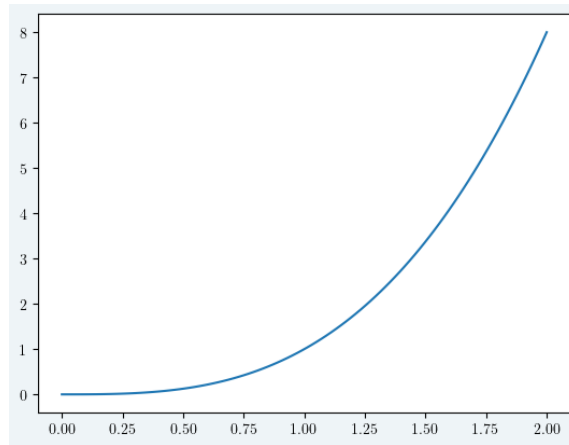
De stap  $y = f(x)$  kan van pas komen, maar is niet noodzakelijk. Dezelfde figuur kan je verkrijgen met de code `ax.plot(x,f(x))`.

We herhalen nog even dat er nog steeds, achter de schermen, met numpy arrays gewerkt wordt. Met onderstaand voorbeeld illustreren we hoe de lambda functie in staat is om de lijst  $[0, \frac{\pi}{6}, \frac{\pi}{2}]$  één voor één om te rekenen naar elementen in een numpy array.

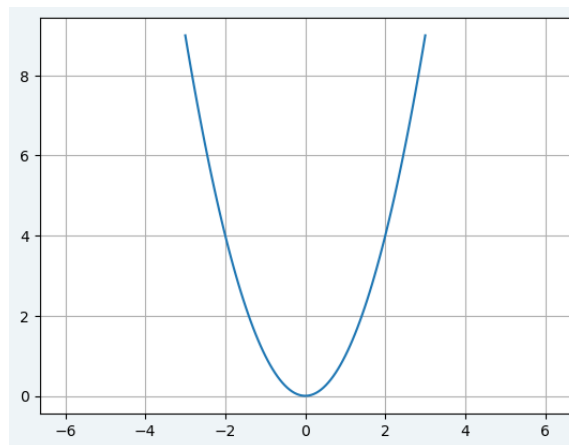
```
import numpy as np
x_waarden = [0,np.pi/6,np.pi/2]
f = lambda x: np.sin(x)
f(x_waarden)
```

Deze laatste regel geeft als output: `array([0. , 0.5, 1. ])`

<sup>11</sup><https://www.spronck.net/pythonboek/pythonboek.pdf>



Figuur 24: Grafiek van een derdegraadsfunctie gegenereerd met de lambda notatie



Figuur 25: Weergave van een grid en gelijke schaalverdeling op de assen

## 5.6 Schaalverdelingen en grid

De wens voor een gelijke schaalverdeling op beide assen, geven we aan met de code `plt.axis('equal')`. Een grid verkrijgen we met het commando `plt.grid()`

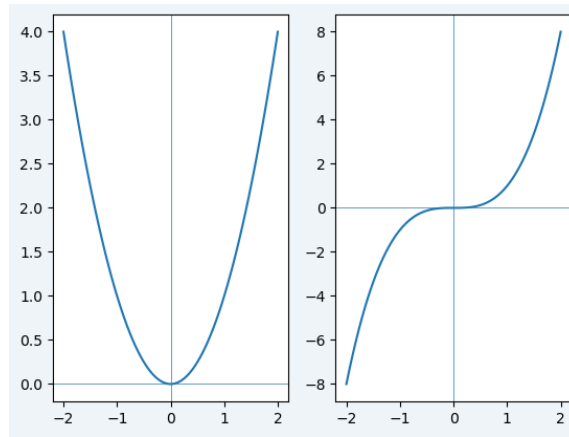
De code [code §5.4]

```
x = np.linspace(-3,3,100)
f = lambda x: x**2
ax.axis('equal')
ax.grid()
ax.plot(x, f(x))
```

levert ons figuur 25

## 5.7 Horizontale en verticale lijnen, tekenbereik

Veronderstel dat we het willen hebben over even en oneven functies, gebruik makende van  $f(x) = x^2$  en  $g(x) = x^3$ . Figuur 26 zou dan baat hebben bij weergave van twee assen door de oorsprong.



Figuur 26: Merk op dat de lijndikte van de assen door de oorsprong kleiner is dan die van de grafiek.

Stiekem trachten we met figuur 26 het verlangen te wekken om in beide figuren de x-as op dezelfde hoogte te tekenen. Geen nood, wat verderop in deze tekst komt dit aan bod.

[code §5.4]

```
x = np.linspace(-2,2,100)
f_1 = lambda x: x**2
f_2 = lambda x: x**3
eerste_assenstelsel.plot(x,f_1(x))
tweede_assenstelsel.plot(x,f_2(x))
eerste_assenstelsel.axhline(y=0,linewidth = 0.5)
eerste_assenstelsel.axvline(x=0,linewidth = 0.5)
tweede_assenstelsel.axhline(y=0,linewidth = 0.5)
tweede_assenstelsel.axvline(x=0,linewidth = 0.5)
eerste_assenstelsel.plot()
tweede_assenstelsel.plot()
plt.show()
```



Dit kan korter. We zien onze kans schoon om de geïnteresseerde lezer het bestaan van *lussen* te verklappen. In combinatie met zogenaamde *selecties* vormen ze de basis van algotimes.



In hoofdstuk 6 (Conditities) en hoofdstuk 7 (Iteraties) van het boek *De Programmeursleerling*<sup>12</sup> van Pieter Spronck krijg je hierover alle nodige uitleg.

Zonder verdere uitleg, tonen we hoe een lus bovenstaande code efficiënter formuleert: [code §5.4]

```
eerste_assenstelsel = axes[0]
tweede_assenstelsel = axes[1]
x = np.linspace(-2,2,100)
f_1 = lambda x: x**2
f_2 = lambda x: x**3
eerste_assenstelsel.plot(x,f_1(x))
tweede_assenstelsel.plot(x,f_2(x))
for ax in axes:
    ax.axhline(y=0,linewidth = 0.5)
    ax.axvline(x=0,linewidth = 0.5)
    ax.plot()
plt.show()
```

Zoals beloofd, brengen we de x-assen nu op gelijke hoogte, met behulp van de instructie `set_ylim(ondergrens, bovengrens)`. We maken ook per functie een apart domein aan.

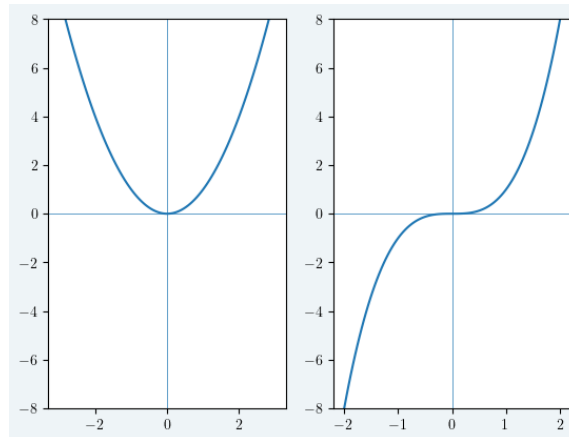
[code §5.4]

<sup>12</sup><https://www.spronck.net/pythonbook/pythonboek.pdf>

```

x_1 = np.linspace(-3,3,100)
x_2 = np.linspace(-2,2,100)
f_1 = lambda x: x**2
f_2 = lambda x: x**3
eerste_assenstelsel.plot(x_1,f_1(x_1))
tweede_assenstelsel.plot(x_2,f_2(x_2))
for ax in axes:
    ax.axhline(y=0,linewidth = 0.5)
    ax.axvline(x=0,linewidth = 0.5)
    ax.set_ylim(-8,8)
    ax.plot()
plt.show()

```



Figuur 27: Beide x-assen staan nu netjes uitgelijnd.

## Caveat

In deze paragraaf kwamen horizontale en verticale lijnen aan bod die doorliepen over de gehele figuur, met behulp van de code:

```

ax.axhline(y=0.5, color='r', linestyle='-')
ax.axvline(x=0.5, color='r', linestyle='-')

```

Verderop in de bundel zullen we ook horizontale en verticale lijnstukken tekenen, waarvan we expliciet de grenzen willen instellen. Daarvoor ziet het commando er anders uit:

```

ax.hlines(y=0.2, xmin=4, xmax=20, linewidth=2, color='r')
ax.vlines(x=0.2, ymin=4, ymax=20, linewidth=2, color='r')

```

## 5.8 Rationale functies

Onder rationale functies verstaan we uitdrukkingen van de vorm:  $f(x) = \frac{t(x)}{n(x)}$  met  $t(x)$  en  $n(x)$  veeltermfuncties en waarbij de graad van  $n(x)$  minstens 1 is.

We bestuderen dit voorbeeld:

$$f(x) = \frac{x^2}{1+x}$$

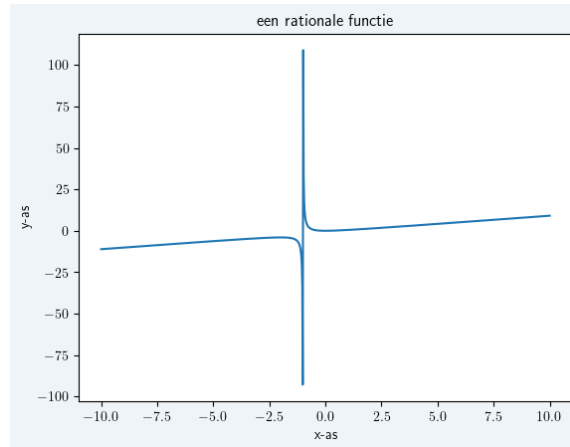
[code §5.4]

```

f = lambda x: x**2/(1+x)
y= f(x)
ax.plot(x,y,linewidth=0.5)

```

```
plt.xlabel('x-as')
plt.ylabel('y-as')
plt.title("een rationale functie")
plt.legend()
plt.show()
```



Figuur 28: Een rationale functie met een vermeende asymptoot

We merken ook op dat er zich ter hoogte van het nulpunt van de noemer - zoals te verwachten - een verticale asymptoot lijkt te manifesteren. Nu is het *niet* zo dat matplotlib spontaan deze verticale asymptoot zal tekenen. De lijnvoering en de kleur van de vermeende asymptoot zijn trouwens identiek aan de grafiek van de functie. Hoe komt dit? Matplotlib tekent - trouw aan de manier waarop het algoritme te werk gaat - lijnstukken tussen twee punten. De (schijnbaar) verticale lijn heeft als startpunt de waarde uit de array met x-waarden, die het dichtst bij  $-1$  langs de linkerzijde (denk aan de linkerlimiet) ligt. Met de gekozen onder- en bovengrens en onderverdeling voor deze figuur

```
ondergrens_interval = -10
bovengrens_interval = 10
onderverdeling = 1000
x = np.linspace(ondergrens_interval, bovengrens_interval, onderverdeling)
```

is dit de waarde

$$x_1 = -1.01101101$$

Voor deze  $x$ -waarde berekent matplotlib de bijhorende  $y$ -waarde:

$$y_1 = -92.82920116694255$$

Aan de rechterzijde doet zich een gelijkaardig verhaal voor met de waarde

$$x_2 = -0.99099099$$

Die heeft als functiewaarde:

$$y_2 = 109.00899680000154$$

De schijnbaar verticale lijn is dus niet de geachte asymptoot, maar het lijnstuk tussen de punten  $P_1(x_1, y_1)$  en  $P_2(x_2, y_2)$ . De  $y$ -waarden verklaren ook waarom matplotlib de  $y$ -as 'afknijpt' ter hoogte van  $-100$  onderaan en ter hoogte van  $110$  bovenaan.

Mits wat extra code kunnen we het linker- en rechterdeel t.o.v. de pool in  $x = -1$  apart plotten. Dit doen we door enerzijds het interval op te delen in een zone links en rechts van de pool en anderzijds de boven- en ondergrens lichtjes te verschuiven m.b.v. `bovengrens_links = -1-epsilon` en `ondergrens_rechts = -1+epsilon`.

[code §5.4]

```

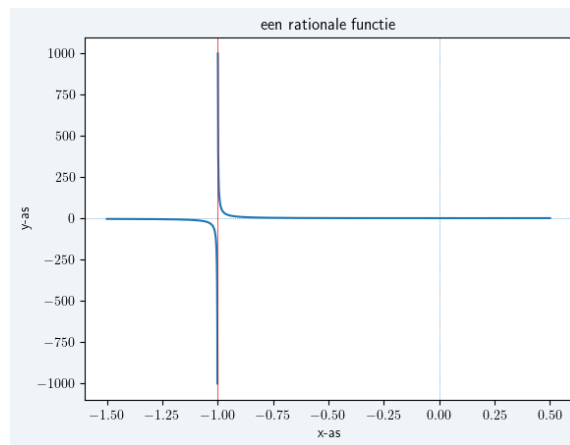
epsilon = 1/onderverdeling
ondergrens_links = -1.5
bovengrens_links = -1-epsilon
ondergrens_rechts = -1+epsilon
bovengrens_rechts = 0.5
x_l= np.linspace(ondergrens_links ,bovengrens_links , onderverdeling )
x_r= np.linspace(ondergrens_rechts ,bovengrens_rechts , onderverdeling )
f = lambda x:x**2/(1+x)
ax.plot(x_l,f(x_l),color=standaardblauw)
ax.plot(x_r,f(x_r),color=standaardblauw)
plt.axvline(x=0,linewidth=0.5,linestyle="dotted")
plt.axhline(y=0,linewidth=0.5,linestyle="dotted")
plt.axvline(x=-1,linewidth=0.5,color="red")
plt.xlabel('x-as')
plt.ylabel('y-as')
plt.ylim=(-2000,2000)
plt.xlim=(-1.5,0.5)
plt.show()

```

We staan even stil bij de het getal  $\epsilon$ . Ondertussen weten we dat de getekende curves geen vloeiende lijnen zijn, maar een opeenvolging van kleine lijnstukjes. Hoe hoger de opdeling van het interval, hoe vloeiender de lijn lijkt. We controleren met de variabele 'onderverdeling' dus de *resolutie* van curve. Door voor epsilon de waarde

$$\frac{1}{\text{onderverdeling}}$$

te kiezen en hiermee de bovengrens aan de linkerzijde van het nulpunt te verlagen vermijden we dat dit laatste element het startpunt zal vormen van een ongewenste lijnstuk.



Figuur 29: De correcte weergave van de rationale functie met inbegrip van een verticale asymptoot in het rood, in  $x = -1$

Plot volgende rationale functies met inbegrip van de asymptoten, in het rood:

- $f(x) = \frac{1}{x^2 - 1}$
- $f(x) = \frac{3x^2 - 9}{x - 3}$

## 5.9 Irrationale functies

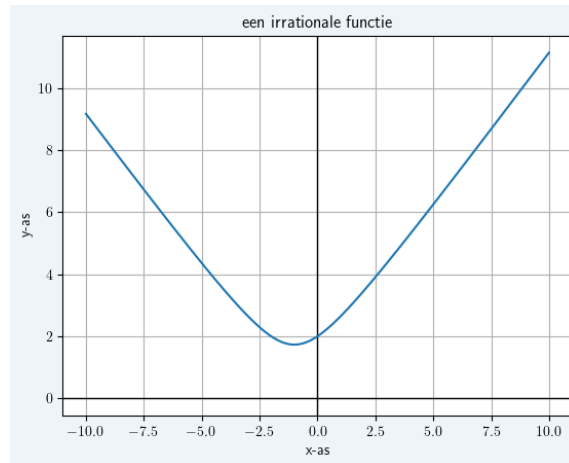
Als voorbeeld kiezen we dit maal de functie:

$$f(x) = \sqrt{x^2 + 2x + 4}$$



We voeren code in om zowel x- als y-as te labelen, om het functievoorschrift en een titel te laten verschijnen. [code §5.4]

```
plt.grid()
plt.xlabel('x-as')
plt.ylabel('y-as')
plt.axhline(y = 0, color = 'black', linestyle = '-', linewidth=1)
plt.axvline(x = 0, color = 'black',linestyle = '-',linewidth=1)
f = lambda x: np.sqrt(x**2+2*x+4)
ax.plot(x, f(x))
plt.title("een irrationale functie")
plt.show()
```



Figuur 30: Een grafiek aangevuld met extra informatie

Plot volgende irrationale functies:

- $f(x) = 2\sqrt[3]{x^5 - 3x^2 + 7} - x - 1$
- $f(x) = \frac{x - 2}{\sqrt{10 - x^2}}$

Zorg ervoor dat de intervallen zodanig zijn opgesteld dat de melding `invalid value encountered` zich niet voordoet

## 5.10 Exponentiële functies

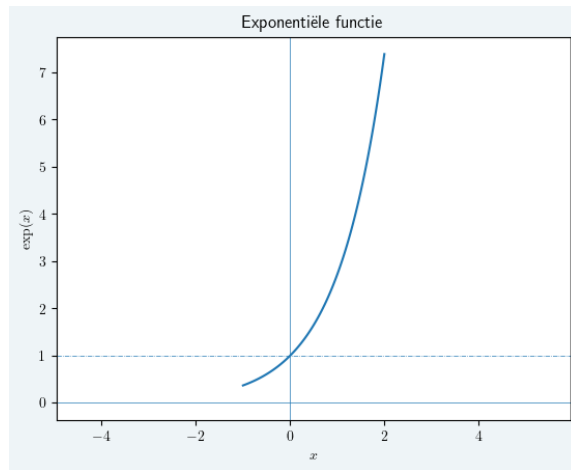
Dan is nu

$$y = e^x$$

aan de beurt. Merk op dat we deze opvragen vanuit de numpy bibliotheek met de code `np.exp(x)`. De aandachtige lezer vindt als extraatje de manier om een stippellijn te tekenen.

[code §5.4]

```
x = np.linspace(-1, 2, 100)
plt.axvline(x=0,linewidth=0.5)
plt.axhline(y=0,linewidth=0.5)
plt.axhline(y=1,linewidth=0.5,linestyle='-.')
f = lambda x : np.exp(x)
ax.axis('equal')
ax.plot(x, f(x))
plt.show()
```



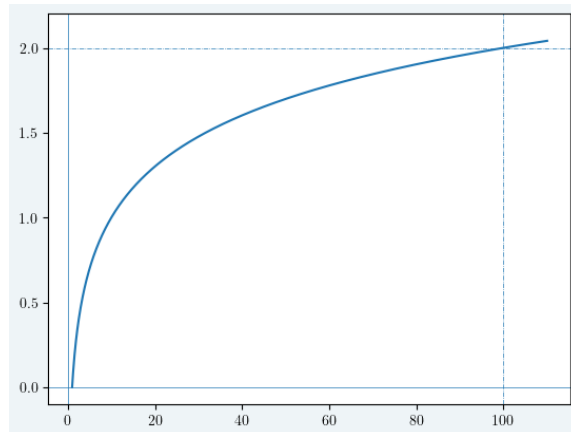
Figuur 31: De functie  $y = \exp x$

### 5.11 Logaritmische functies

Voor het tiendelig logaritme gebruiken we `np.log10()`, voor het natuurlijke `np.log()`. Dit keer specificeren we het gewenste bereik op de  $y$ -as, met `plt.ylim(-0.1, 2.2)`. (Analoog is dit mogelijk voor de  $x$ -as met `plt.xlim(-1, 1)`.)

[code §5.4]

```
f = lambda x: np.log10(x)
y = f(x)
ax.plot(x, y)
ax.axhline(0, linewidth = 0.5)
ax.axvline(0, linewidth = 0.5)
ax.axhline(2, linewidth = 0.5, linestyle="-.")
ax.axvline(100, linewidth = 0.5, linestyle="-.")
plt.ylim(-0.1, 2.2)
plt.show()
```



Figuur 32: De grafiek van de tiendelige logaritme.

### 5.12 Goniometrische functies

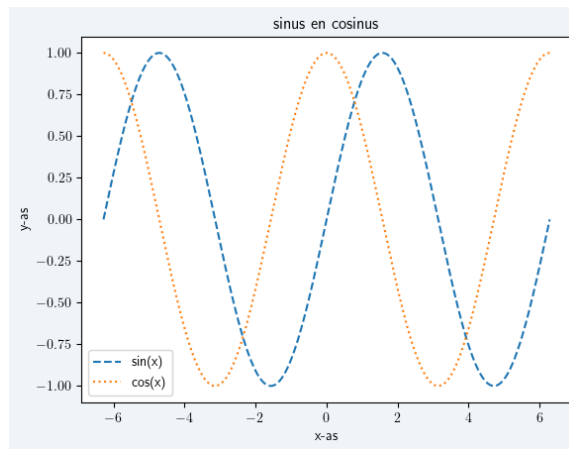
Dit voorbeeld toont een andere manier om lijntypes in te stellen. De optie `linestyle="dashdot"` behoort eveneens tot de mogelijkheden. De code

[code §5.4]

```

ondergrens = -2*np.pi
bovengrens = 2*np.pi
onderverdeling = 1000
x = np.linspace(ondergrens, bovengrens, onderverdeling)
f_1 = lambda x: np.sin(x)
f_2 = lambda x: np.cos(x)
ax.plot(x, f_1(x), label='sin(x)', linestyle="dashed")
ax.plot(x, f_2(x), label='cos(x)', linestyle="dotted")
plt.ylim(-1.1, 1.1)
plt.xlabel('x-as')
plt.ylabel('y-as')
plt.title("sinus en cosinus")
plt.legend()
plt.show()

```



Figuur 33: Een figuur voorzien van een legende.

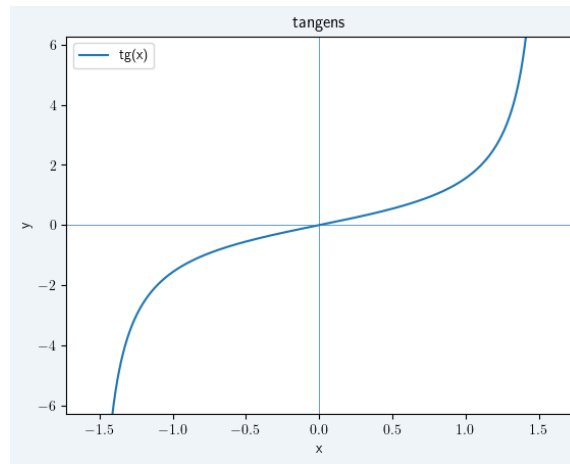
Ook bij het plotten van de tangens dienen we, omwille van de werking van het plotalgoritme, rekening te houden met verticale asymptoten. Zo duikt de grafiek van de tangens in de nabijheid van  $x = -\pi/2$  de diepte in, in de nabijheid van  $x = \pi/2$  schiet ze omhoog. Zoals we reeds eerder zagen bij de studie van de grafiek van rationale functies, hebben we baat bij het gebruik van een heel klein getal  $\epsilon$  om het interval waarover we willen plotten, subtiel aan te passen. Met deze elegante oplossing simuleren we een open interval. Voor figuur 34 werd voor  $\epsilon$  de waarde  $10^{-7}$  gekozen.

[code §5.4]

```

epsilon = 1e-7
x= np.linspace(-np.pi/2+epsilon, np.pi/2-epsilon, 1000)
f = lambda x: np.tan(x)
ax.plot(x, f(x), label='tg(x)')
ax.axhline(y = 0, linewidth=0.5)
ax.axvline(x = 0, linewidth=0.5)
plt.xlabel('x')
plt.ylabel('y')
plt.title("tangens")
plt.legend()
plt.ylim(-np.pi*2, np.pi*2)
plt.show()

```



Figuur 34: Ook de grafiek van de tangens functie vraagt wat extra aandacht inzake asymptoten.

De cotangens maakt het ons nog iets moeilijker. Het is namelijk zo dat Numpy-bibliotheek niet uitgerust is met een standaard cotangensfunctie.



In de paragraaf over de lambda-notatie 5.5 trachtten we de lezer al warm te maken om wat meer achtergrond te verwerven op vlak van *programmeerfuncties*. De cotangens vormt opnieuw een aanleiding om hoofdstuk 8 (Functies) van het boek *De Programmeursleerling*<sup>13</sup> van Pieter Spronck aan te bevelen.



We kunnen het ontbreken van de cotangensfunctie immers mooi oplossen door zelf een functie te **definiëren**, die als *input* de variabele  $x$  ontvangt en als resultaat de verhouding  $\frac{\cos(x)}{\sin(x)}$  *teruggeeft*. Vandaar, op het einde van de *body* van de methode, het woordje Engels: *return*. Achter de twee lijnen

```
def cotan(x):
    return np.cos(x)/np.sin(x)
```

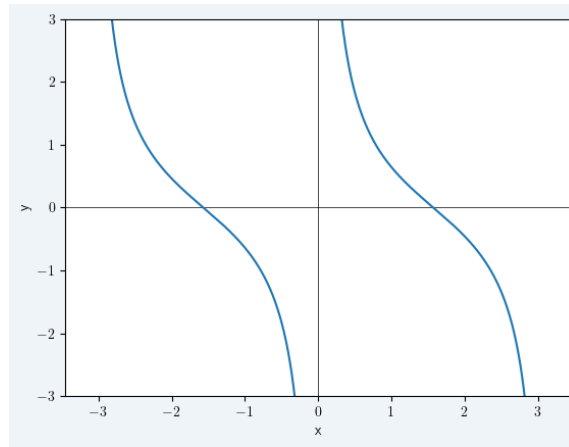
zit dus een fundamenteel programmeerconcept verscholen.

Het limietgedrag van de cotangens rond de oorsprong vraagt opnieuw voor een opdeling in deelintervallen links en rechts van de  $y$ -as.

[code §5.4]

```
def cotan(x):
    return 1/np.tan(x)
epsilon = 1e-3
x_l= np.linspace(-np.pi+epsilon,-epsilon,1000)
x_r = np.linspace(epsilon, np.pi-epsilon,1000)
ax.axvline(x = 0, color = 'black', linestyle = '-', linewidth=0.5)
ax.axhline(y = 0, color = 'black', linestyle = '-', linewidth=0.5)
ax.plot(x_r,cotan(x_r),color=standaardblauw)
ax.plot(x_l,cotan(x_l),color=standaardblauw)
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-3, 3)
plt.show()
```

<sup>13</sup><https://www.spronck.net/pythonbook/pythonboek.pdf>



Figuur 35: Grafiek van de cotangens

Los het volgende stelsel op met behulp van Sympy en visualiseer zowel de cirkel als de rechte in een assenstelsel.

$$eq1 = x^2 + y^2 - 9$$

$$eq2 = 2x - 1 - y$$

Om te kunnen plotten met de lambda-functie, dien je uit het impliciete voorschrift van de cirkel, de bovenste- en onderste helft aparte te definiëren als explicite functies. Gebruik de optie `axis = 'equal'` om te voorkomen dat er een ellips verschijnt. Teken kruisjes ter hoogte van de snijpunten van de rechte en de cirkel.

## 6 Calculus

Een volledig overzicht is terug te vinden op de officiële site<sup>14</sup> We overlopen enkele basisbegrippen over limieten, afgeleiden en integralen.

### 6.1 Limieten

De uitgebreide verzameling  $\overline{\mathbb{R}}$  omvat de reële getallen met inbegrip van de concepten min- en plus-oneindig  $\pm\infty$ . Een aantal klassieke rekenregels om te rekenen met deze begrippen zijn:

$$\begin{array}{lll} a + (+\infty) = +\infty; & a + (-\infty) = -\infty; & (+\infty) + (+\infty) = +\infty; \\ a \cdot (+\infty) = \pm\infty & \text{volgens teken van } a; & (-\infty) + (-\infty) = -\infty; \\ a \cdot (-\infty) = \mp\infty & \text{volgens teken van } a; & (+\infty) - (-\infty) = +\infty; \\ \frac{a}{\pm\infty} = 0 & a \text{ is eindig}; & (-\infty) - (+\infty) = -\infty; \end{array}$$

Tabel 6: Uitbreiding van rekenregels

Belangrijk is te onthouden dat volgende combinaties leiden tot *onbepaaldheden*:

$$(+\infty) + (-\infty) \quad (2)$$

$$0 \cdot (\pm\infty) \quad (3)$$

$$\frac{\pm\infty}{\pm\infty} \quad (4)$$

In programmeercontext zal de uitdrukking NaN ten tonele verschijnen. Dit staat voor Not a Number. Het geeft aan dat

- er een grootheid nog niet is *geïnitieerd* (er werd nog geen concrete waarde toegekend aan een variabele)
- er doet zich een onbepaaldheid voor

Sympy beschikt over de module 'Limit' die ons ter hulp schiet bij het berekenen van limieten. Om met het begrip oneindig aan het rekenen te gaan, doen we beroep op de module 'S'.

#### 6.1.1 Terugkerende code

Deze code dient de voorbeelden uit deze paragraaf vooraf te gaan:

```
import sympy as sp
x = Symbol('x')
```

#### 6.1.2 Rekenen met oneindig

We roepen  $+\infty$  en  $-\infty$  op met

[code(6.1.1)]

```
sp.S.Infinity
sp.S.NegativeInfinity
```

Willen we  $+\infty - (+3)$  bereken, dan kan dat als volgt:

[code(6.1.1)]

```
sp.S.Infinity -(+3)
```

<sup>14</sup><https://docs.sympy.org/latest/tutorials/intro-tutorial/calculus.html>

Dit geeft als resultaat

$\infty$

Zonder weergave van teken, heeft SymPy het dus over *plus* oneindig.

Bepaal, indien mogelijk:

1.  $-\infty - \infty$
2.  $3 - (-\infty)$
3.  $-\infty + \infty$
4.  $(-\infty) \cdot \left(\frac{-1}{3}\right)$
5.  $0 \cdot (+\infty)$
6.  $\frac{-\infty}{-3}$
7.  $\frac{-3}{-\infty}$
8.  $\frac{0}{-\infty}$
9.  $3(-\infty)^3 + 5(-\infty)$
10.  $3(+\infty)^2 - (+\infty) + 4$
11.  $\sqrt[4]{-\infty}$
12.  $\sqrt{4(-\infty)^2 - (-\infty)}$

### 6.1.3 Limieten naar $\infty$ van functies

Voor het berekenen van een limiet op oneindig van een functie met een variabele  $x$

$$\lim_{x \rightarrow \infty} \frac{1}{x}$$

beschikken we over volgende code:

[code(6.1.1)]

```
sp.Limit(1/x,x,S.Infinity)
```

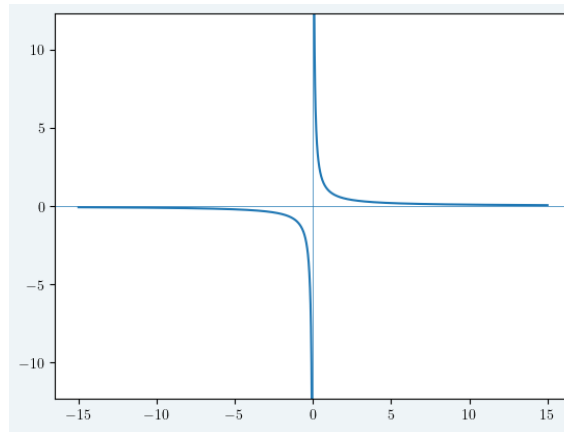
Uitvoeren van bovenstaande cel blijft beperkt tot het mooi symbolisch uitschrijven van deze uitdrukking. Willen we effectief een waarde bekomen, dan moeten we wat aandringen met `doit()`:

```
sp.Limit(1/x,x,S.Infinity).doit()
```

SymPy laat ons nu gelukkig wel weten dat de waarde op plus oneindig 0 bedraagt. Willen we het gedrag op min oneindig kennen, dan vullen we de *direction* aan:

```
sp.Limit(1/x,x,S.Infinity,dir='-').doit()
```

Dit geeft 0 als resultaat.



Figuur 36: De grafiek van de functie  $f(x) = \frac{1}{x}$

Bepaal de limieten voor  $x \rightarrow \pm\infty$  van volgende functies:

1.  $\lim_{x \rightarrow \pm\infty} \frac{(2x-1)^5}{(3x+4)^8}$
2.  $\lim_{x \rightarrow \pm\infty} \frac{(3x-7)^4}{9x^4+2}$
3.  $\lim_{x \rightarrow \pm\infty} \left[ \left( 4 + \frac{3}{x^2+1} \right) - \left( 9 - \frac{2}{x+1} \right) \right]$
4.  $\lim_{x \rightarrow \pm\infty} \frac{2 - \frac{9}{x^2}}{x + \frac{5}{x+3}}$

#### 6.1.4 Limieten naar $a \in \mathbb{R}$

Coderen van een uitdrukking als

$$\lim_{x \rightarrow 0^-} \frac{1}{x}$$

gebeurt als volgt:

```
sp.Limit(1/x,x,0,dir='-')
```

Merk op dat het hier over de linkerlimiet gaat.

SymPy is in staat om om te gaan met onbepaaldheden als  $\frac{0}{0}$  en  $\frac{\infty}{\infty}$ , zoals te zien in de berekening van

$$\lim_{x \rightarrow 0} \frac{\sin(x)}{x}$$

[code(6.1.1)]

```
sp.Limit(sin(x)/x,x,0).doit()
```

Zo komen we te weten dat het resultaat 1 bedraagt.

We sluiten deze paragraaf af met de code nodig voor het genereren van de figuur 36:

[code §5.4]

```
epsilon = 10**(-2)
fig = plt.figure(facecolor=lichtblauw)
ax = fig.add_subplot(111)
ax.axis('equal')
xl= np.linspace(-15,-epsilon,1000)
xr= np.linspace(epsilon,15,1000)
ax.plot(xl,1/xl,color=standaardblauw)
```



```
ax.plot(xr,1/xr,color=standaardblauw)
ax.set_ylim(-10,10)
ax.set_xlim(-10,10)
ax.axhline(y=0,linewidth=0.5,color=standaardblauw)
ax.axvline(x=0,linewidth=0.5,color=standaardblauw)
plt.show()
```

## 6.2 Afgeleiden

### 6.2.1 Terugkerende code

Laden we de klasse `Derivative` in, dan behoort afleiden tot de mogelijkheden. De voorbeelden van deze paragraaf hebben nood aan deze voorafgaande lijnen:

```
import sympy as sp
init_printing()
x = Symbol('x')
```

### 6.2.2 Het `Derivative()` commando

We starten met de kwadratische functie  $f(x) = 3x^2 - 2x + 6$  en gaan op zoek naar de afgeleide functie. [code(6.2.1)]

```
f = 3*x**2-2*x+6
sp.Derivative(f,x)
```

SymPy toont ons

$$\frac{d}{dx}(3x^2 - 2x + 6)$$

We bedanken SymPy voor het mooi presenteren van de uitdrukking, maar we dringen aan om de berekening effectief uit te voeren:

```
sp.Derivative(f,x).doit()
```

Tevreden stellen we vast dat de gezochte afgeleide functie nu wel verschijnt:

$$6x - 2 \quad (5)$$

Berekenen van een concrete waarde doen we met behulp van het eerder gebruikte `subs()` commando. Om het wat overzichtelijk te houden, slaan we eerst de afgeleide functie op in een nieuwe variabele.

```
d = sp.Derivative(f,x)
d.doit().subs({x:1})
```

Dit resulteert in de waarde 4. Mits declareren van een extra symbolische variabele, kunnen we ook een symbolische substitutie uitvoeren:

[code(6.2.1)]

```
x, x1 = symbols('x x1')
f = 3*x**2-2*x+6
d = sp.Derivative(f,x)
d.doit().subs({x:x1})
```

$$6x_1 - 2$$

[code(6.2.1)]

```
x, x1 = symbols('x x1')
f = 3*x**2-2*x+6
d = sp.Derivative(f,x)
d.doit().subs({x:x1})
```

Bereken de afgeleide functie van

$$(x^3 + x^2 + x)(x^2 + x)$$

Net zoals we symbolen kunnen definiëren, kunnen we onbepaalde functies in het leven roepen met het commando `Function()`:

```
f = sp.Function('f')
```

Dit laat ons toe een berekening te maken als

```
sp.diff(f(x), x, 1)
```

om  $\frac{d}{dx}f(x)$  als antwoord te krijgen.

Het vinden van extrema zal mogelijk zijn door de nulpunten van de afgeleide functie te onderzoeken.

[code(6.2.1)]

```
f = x**5-30*x**3+50*x
afgeleide = sp.Derivative(f,x).doit()
kritische_punten = sp.solve(afgeleide)
kritische_punten
```

We verkrijgen dan de lijst met kritische punten:

$$\left[ -\sqrt{9 - \sqrt{71}}, \sqrt{9 - \sqrt{71}}, -\sqrt{\sqrt{71} + 9}, \sqrt{\sqrt{71} + 9} \right]$$

Opvragen van een element uit de lijst, doen we met behulp van rechte haken, waarin we de *index* van het element vermelden. Belangrijk is dat het eerste element als index 0 heeft. Opvragen van het eerste element doen we dus als

```
kritische_punten[0]
```

De tweede afgeleide functie berekenen kan uiteraard door afgeleide functie opnieuw af te leiden, maar we kunnen ook als extra parameter de tweede orde meegeven:

```
tweede_afgeleide = sp.Derivative(f,x,2).doit()
```

Om de concrete cijferwaarde horende bij de tweede afgeleide in het eerste kritische punt te verkrijgen, combineren we volgende commando's:

```
tweede_afgeleide.subs({x:kritische_punten[0]}).evalf()
```

Zo leren we dat de waarde 127.661060789073 bedraagt. Door systematisch alle kritische punten te evalueren, krijgen we meer inzicht in de aard van deze extrema. Een grafiek mag bij dit soort oefening uiteraard niet ontbreken. Opgepast: code zorgt zowel voor een doorlopende horizontale lijn, met `axhline` als voor horizontale lijnstukken, met `hlines`

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

standaardblauw="tab:blue"
lichtblauw = "#EEF4F7"

fig = plt.figure(facecolor=lichtblauw)
ax = fig.add_subplot()

x= np.linspace(-5,5,1000)
f= lambda x: x**5-30*x**3+50*x
y = f(x)
```

```

ax.plot(x, f(x)) #, label='tg(x)')
ax.axhline(y = 0, linewidth=0.5)
ax.axvline(x = 0, linewidth=0.5)
plt.xlabel('x')
plt.ylabel('y')

sp.init_printing()
x = sp.Symbol('x')
functie = x**5-30*x**3+50*x

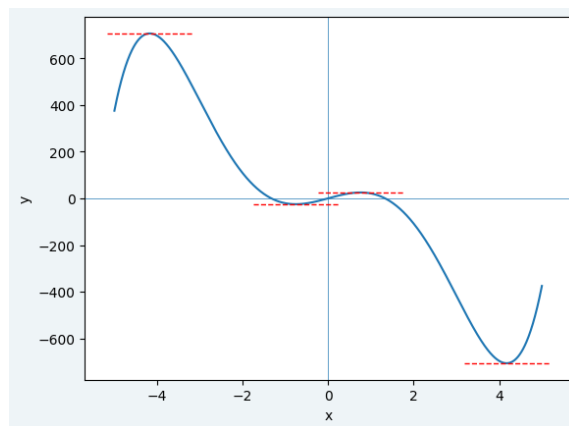
afgeleide = sp.Derivative(functie, x).doit()
kritische_punten = sp.solve(afgeleide)
kritische_punten

#tweede_afgeleide = Derivative(f, x, 2).doit()
#tweede_afgeleide.subs({x: kritische_punten[0]}).evalf()

for point in kritische_punten:
    ax.hlines(f(point), point-1, point+1, color='red',
              linestyle='dashed', linewidth=1)

plt.show()

```



Figuur 37: De functie met weergave van horizontale raaklijnen ter hoogte van kritische punten.

Bepaal de afgeleide functie van:

1.  $f(x) = (2x^2 - 5)(x + 1)$

2.  $f(x) = 3x^{2-r} + 5x^{2r-m}$

3.  $f(x) = \frac{1}{2x^4}$

4.  $f(x) = \frac{1}{(x-3)(x+2)}$

5.  $f(x) = \sqrt{5x^3 + x^2}$

6.  $f(x) = \sqrt{5 + \frac{3}{x}}$

7.  $f(x) = \frac{x}{\sqrt[8]{(x^2 - 1)^3}}$

8.  $f(x) = \ln(x^2 + 1)$

9.  $f(x) = \frac{\exp(-x)}{\exp(-x) + 1}$

10.  $f(x) = x^6 \exp(x)$

11.  $f(x) = \cot x$

12.  $f(x) = \operatorname{Arccos} x$

13.  $f(x) = \frac{\sin^2 x + 4}{\sin^2 x}$

14.  $f(x) = \operatorname{Arctan}\left(\frac{a}{b} \tan x\right)$

15.  $f(x) = \sec x$

## 6.3 Integralen

### 6.4 Terugkerende code

De aanpak voor het berekenen van integralen vertoont sterke gelijkenissen met die voor afgeleiden. De onbepaalde integraal van  $\frac{x^2 + 2x + 2}{x + 2}$  verkrijgen we door:

```
import sympy as sp
sp.init_printing()
x = sp.Symbol('x')
f = (x**2+2*x+2)/(x+2)
integraal_1 = sp.Integral(f)
integraal_1.doit()
```

Dit geeft ons

$$\frac{x^2}{2} + 2 \log(x + 2)$$

Voor de bepaalde integraal vullen we het commando `Integral` aan met de veranderlijke in kwestie en de boven- en ondergrens:

```
waarde = sp.Integral(f, (x, 0, 2))
waarde.doit()
```

$$2 \log(2) + 2 + 2 \log(4)$$

### 6.5 Facultatieve toepassing: Riemannintegratie

We eindigen dit hoofdstuk met een -voor deze bundel gevorderde - toepassing. Het programmeerwerk vereist kennis van

- programmeerfuncties
- lijsten (arrays)
- Lussen

Vermeldingen van deze aparte, facultatieve onderwerpen kwamen voor in de vorige hoofdstukken. Deze paragraaf is dus gericht op lezers die nieuwsgierig zijn naar toepassing waarin dit alles gecombineerd wordt.

Beschouw de functie

$$f(x) = x^2$$

De bepaalde integraal voor een ondergrens  $x = -2$  en bovengrens  $x = 2$  is

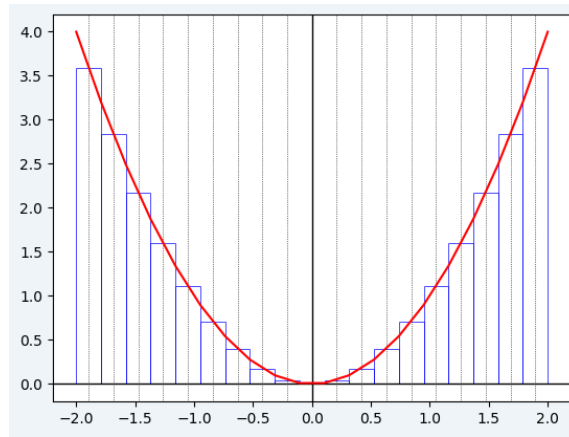
$$\int_{-2}^2 x^2 dx = \left[ \frac{x^3}{3} \right]_{-2}^2 \quad (6)$$

$$= \frac{1}{3} (2^3 - (-2)^3) \quad (7)$$

$$= \frac{1}{3} (8 + 8) \quad (8)$$

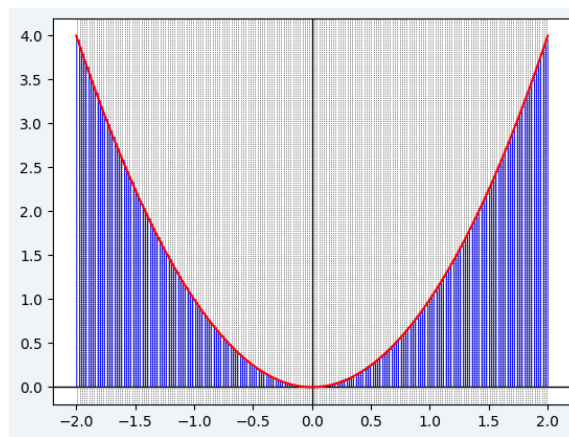
$$= \frac{16}{3} \approx 5.3333 \dots \quad (9)$$

Het rekenwerk horende bij figuur 38 geeft oppervlakte bij benadering: 5.318559556786704



Figuur 38: Een partitie van het interval  $[-2, 2]$  in  $20 - 1 = 19$  rechthoeken

Het rekenwerk horende bij figuur 38 geeft oppervlakte bij benadering: 5.3331986565995795



Figuur 39: Een partitie van het interval  $[-2, 2]$  in  $200 - 1 = 199$  rechthoeken

```
import matplotlib.pyplot as plt
import numpy as np
lichtblauw = "#EEF4F7"
fig, ax = plt.subplots(facecolor=lichtblauw)
g = lambda x: x**2

ondergrens = -2
bovengrens = 2
onderverdeling = 20
x = np.linspace(ondergrens, bovengrens, onderverdeling)
oppervlakte = 0
for i in range(0, len(x)-1):
    x_in_midden = (x[i]+x[i+1])/2
    ax.vlines(x=x[i], ymin=0, ymax=g(x_in_midden),
              linewidth=0.5, color="blue", ls='-')
    ax.hlines(y=g(x_in_midden), xmin=x[i], xmax=x[i+1],
              linewidth=0.5, color="blue", ls='-')
    ax.vlines(x=x[i+1], ymin=0, ymax=g(x_in_midden),
              linewidth=0.5, color="blue", ls='-')
    ax.axvline(x=x_in_midden,
               linewidth=0.5, color="black", ls=':')
    breedte = x[i+1]-x[i]
```

```

opp_i= g(x_in_midden)*breedte
#print("oppervlakte van segment ", i, "=", oppervlakte )
oppervlakte = oppervlakte +opp_i

```

```

ax.axhline(y=0,linewidth=1,color="black")
ax.axvline(x=0,linewidth=1,color="black")
ax.plot(x,g(x),color= "red")
print("oppervlakte bij benadering: ", oppervlakte)
plt.show()

```

**Los de volgende onbepaalde integralen op:**

1.  $\int 10^x - \frac{1}{\sin^2 x} dx$
2.  $\int 2 \cosh(x) dx$
3.  $\int \frac{10}{\cosh^2(x)} - 3a^x - b \sin x dx$
4.  $\int 5 \cdot 3^x - \frac{1}{2\sqrt{x}} dx$
5.  $\int \sqrt{x} \sqrt{x} dx$

**Bereken de volgende bepaalde integralen**

1.  $\int_1^e \frac{dt}{t}$
2.  $\int_1^4 \frac{1-z^2}{z} dz$
3.  $\int_{\pi}^2 \cos \phi d\phi$
4.  $\int_{0,5}^5 \frac{4}{t} dt$
5.  $\int_0^{\pi/4} \frac{1 - \cos^2 x}{2 \cos^2 x} dx$
6.  $\int_1^9 \sqrt{x}(2-x) dx$

## 7 Matrices en vectoren

Een volledig overzicht is terug te vinden op de officiële site<sup>15</sup>

We starten opnieuw met het inladen van de sympy bibliotheek in de eerste cel.

```
import Sympy as sp
```

Het aanmaken van een  $2 \times 2$  matrices gebeurt als volgt:

```
sp.Matrix([[1, 2], [3, 4]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Merk de 'geneste' structuur van de arrays (aangegeven met rechte haken) op, nodig voor het aanmaken van een matrix.

Enkel voor het aanmaken van matrices met één enkele kolom, is dit niet nodig:

```
sp.Matrix([1, 2, 3])
```

geeft

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

We zullen wat verderop zien dat dit de manier is om vectoren voor te stellen.

Een voorbeeld met symbolen, dadelijk ook met een alternatieve manier om matrices te definiëren:

```
x, y, z = sp.symbols('x y z')
rij1 = [2*x, 3*y, -z]
rij2 = [-x, 1/y, 0]
rij3 = [x, y, z]
matrix = sp.Matrix([rij1, rij2, rij3])
```

$$\begin{bmatrix} 2x & 3y & -z \\ -x & \frac{1}{y} & 0 \\ x & y & z \end{bmatrix}$$

In dit voorbeeld werden de rijen eerst apart de rijen als arrays opgegeven, om in een volgende stap de matrix te creëren.

### 7.1 Dimensies

Opvragen van de dimensies gebeurt met het commando `shape()`. Beschouw de matrix  $M$

```
M = sp.Matrix([[1, 2, 3], [-2, 0, 4]])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ -2 & 0 & 4 \end{bmatrix}$$

```
sp.shape(M)
```

Aangezien  $M \in \mathbb{R}^{2 \times 3}$  verkrijgen we (2, 3) na uitvoeren van bovenstaande cel.

---

<sup>15</sup><https://docs.sympy.org/latest/tutorials/intro-tutorial/matrices.html>



## 7.2 Opvragen van rijen en kolommen

Om specifieke rijen of kolommen te bereiken, gebruik je `row( index van rij)` en `row( index van kolom)`

De laatste kolom kan je eveneens opvragen door als index `-1` mee te geven.

Gebruiken we opnieuw de matrix  $M$  uit de vorige paragraaf, dan geeft

```
M.row(0)
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

terug, en

```
M.col(-1)
```

levert

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

## 7.3 Rijen en kolommen toevoegen en verwijderen

De commando's `row_del(index van rij)` en `col_del(index van kolom)` laten toe rijen en kolommen te verwijderen.

(We werken nog steeds met dezelfde matrix  $M$ .) Na uitvoeren van de code

```
M.col_del(0)
```

$M$

ziet  $M$  er als volgt uit:

$$\begin{bmatrix} 2 & 3 \\ 0 & 4 \end{bmatrix}$$

Nog wat verder snoeien door

```
M.row_del(1)
```

zal  $M$  herleiden tot

$$\begin{bmatrix} 2 & 3 \end{bmatrix}$$

Krijgen we teveel medelijden, dan kunnen we  $M$  opnieuw laten groeien met de commando's `row_insert(positie waar de rij ingevoerd dient te worden, de rij zelf)` en `col_insert(positie waar de kolom ingevoerd dient te worden, de kolom zelf)` De code

```
M = M.row_insert(1, Matrix([[0, 4]]))
```

laat  $M$  de vorm

$$\begin{bmatrix} 2 & 3 \\ 0 & 4 \end{bmatrix}$$

aannemen.

Voeren we vervolgens

```
M = M.col_insert(0, Matrix([1, -2]))
```

uit, dan zal  $M$  zich tonen als

$$\begin{bmatrix} 1 & 1 & 3 \\ -2 & 0 & 4 \end{bmatrix}$$

## 7.4 Bewerkingen

Voor de optelling beschikken we over de standaardoperatie `+`.

Berekenen van het product van matrices kan eenvoudig met het `*` symbool.

```
product = M*B
```

Machtversheffing kan eveneens met dezelfde notatie als die voor reële getallen: `**`.

Voor het voorstellen van de getransponeerde beschikken we over `Transpose`:

```
sp.Transpose(M)
```

Willen we het transponeren effectief uitvoeren, dan dringen we aan met het commando `doit()`

```
sp.Transpose(M).doit()
```

## 7.5 Determinanten

Het berkenen van een determinant kan met het commando `det()`:

```
M = sp.Matrix([[1, 0, 1], [2, -1, 3], [4, 3, 2]])
M.det()
```

Dit geeft  $-1$  als resultaat.

## 7.6 Interveren van een matrix

De inverse bekomen we door  $-1$  als macht op te geven.

```
M = Matrix([[1, 3], [-2, 3]])
M**-1
```

toont

$$\begin{bmatrix} \frac{1}{3} & -\frac{1}{9} \\ \frac{2}{3} & \frac{1}{9} \end{bmatrix}$$

## 7.7 Enkele shortcuts voor specifieke matrices

Met het commando `eye(orde van de matrix)` kan je snel eenheidsmatrices laten verschijnen:

Zo zal

```
eye(4)
```

de matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

tot leven roepen.

Analoge commando's bestaan voor het aanmaken van matrices met nullen en eentjes: gebruik `zeros(n,m)` en `ones(n,m)` voor een nulmatrix of matrix met eentjes met  $n$  rijen en  $m$  kolommen,

Diagonaalmatrices kan je verkrijgen met het commando `diag(diagonaalelementen)`.

```
sp.diag(1, 2, 3)
```

verwekt

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

## 7.8 Rijgereduceerde vorm

Een matrix

$$\begin{bmatrix} 1 & 0 & 1 & 3 \\ 2 & 3 & 4 & 7 \\ -1 & -3 & -3 & -4 \end{bmatrix}$$

terugbrengen tot

$$\begin{bmatrix} 1 & 0 & 1 & 3 \\ 0 & 1 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

kan met `M.rref()`.

Definieer de matrix  $A$  als:

$$\begin{bmatrix} 1 & 0 & 1 \\ -1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

Maak nu een kolomvector  $B$  met als elementen  $x, y, z$ . Bereken  $A \times B$  om tot het resultaat te komen:

$$\begin{bmatrix} x + z \\ -x + 2y + 3z \\ x + 2y + 3z \end{bmatrix}$$

Laat vervolgens de getransponeerde van het resultaat als output van de notebookcel verschijnen.

### Symbolisch rekenen met matrices

Door beroep te doen op de functie `MatrixSymbol`, aangevuld met de gewenste symbolische weergaves van het aantal rijen en kolommen, creëren we een symbolische matrix van orde  $n \times m$ :

```
n, m = sp.symbols('n m', integer = True)
M = sp.MatrixSymbol("M", n, m)
b = sp.MatrixSymbol("b", m, 1)
```

Nu kunnen we bijvoorbeeld  $M * b$  berekenen:

```
M*b
```

geeft als resultaat `Mb`. Dankzij het commando `shape` kunnen we de dimensies van het resultaat van de berekening, opvragen:

```
(M*b).shape
```

vertelt ons `(n,1)`

#### 7.8.1 Indeces

Uitdrukkingen als  $A_{ij}$  bereiden we voor door te werken met `IndexBase`:

```
A = sp.IndexBase("A")
i = sp.Idx('i')
j = sp.Idx('j')
```

Opvragen van `A[i,j]` levert inderdaad

$$A_{ij}$$

op.

1. Gegeven zijn de volgende matrices

$$A = \begin{bmatrix} 1 & 7 \\ -3 & 0 \end{bmatrix}, B = \begin{bmatrix} 2 & -4 \\ 6 & 5 \end{bmatrix} \text{ en } C = \begin{bmatrix} 0 & 0 \\ 1 & 3 \end{bmatrix}$$

- (a) Bereken  $A - B + 2C$
- (b) Bereken  $4A + 7B - C$
- (c) Bepaal de matrix  $X$  als geldt  $2X + 5A = B$
- (d) Bepaal de matrix  $X$  als geldt  $X + B + C = A + B$

2. Zelfde opgaven als bij 1, voor de volgende matrices:

$$A := \begin{bmatrix} 2 & 1 & 3 \\ 4 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 8 & 4 & 0 \\ -4 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \text{ en } C = \begin{bmatrix} 0 & 0 & 2 \\ 2 & 3 & 1 \\ 0 & 5 & 2 \end{bmatrix}$$

3. Schrijf de getransponeerde matrix op van A en C uit opgave 1.

4. Bereken de volgend producten:

$$\begin{aligned} & \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 \\ 1 & 2 \end{bmatrix} \\ & \begin{bmatrix} 8 & 2 & 1 \\ 5 & 7 & 3 \\ 2 & 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ & \begin{bmatrix} 4 & 0 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \\ & \begin{bmatrix} 4 & 2 & -9 \\ 1 & 0 & -5 \\ -8 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & 0 \end{bmatrix} \\ & \begin{bmatrix} 1 & 4 & -5 \\ 0 & 1 & 3 \\ -5 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix} \\ & \begin{bmatrix} 1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 5 \\ -9 & 5 & -7 \\ 0 & 2 & 0 \end{bmatrix} \end{aligned}$$

5. Gegeven zijn de volgende matrices

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 \\ 2 & -1 \end{bmatrix} \text{ en } C = \begin{bmatrix} 1 & 0 \\ 5 & -2 \end{bmatrix}$$

Bereken

- (a)  $A(BC)$  en  $(AB)C$
- (b)  $A(B + C)$  en  $AB + AC$

6. Bereken de volgende determinanten

$$\begin{vmatrix} 1 & a \\ 1 & b \end{vmatrix}$$

$$\begin{vmatrix} \sin \alpha & \cos \alpha \\ -\cos \alpha & \sin \alpha \end{vmatrix}$$

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

52

$$\begin{vmatrix} a & 1 & 1 \\ ab & a & b \end{vmatrix}$$

## 7.9 Vectoren

### 7.9.1 Terugkerende code

Een notebook vertrekkende van deze paragraaf, begint best met deze eerste cel:

```
import SymPy as sp
import matplotlib.pyplot as plt
```

Met het oog op visualisatie, hebben we alvast extra functionaliteiten om te plotten, ingeladen.

### 7.9.2 Aanmaken van twee-en driedimensionale vectoren

Vanuit de algebra van vectorruimten, weten we dat we de kolommen en rijen van matrices kunnen beschouwen als kolom- en rijvectoren. SymPy steunt hierop: een vector definieer je met behulp van de `sympy.Matrix`-klasse. In het geval van tweedimensionale vectoren  $\mathbf{u2d} = (2, 3)$  en  $\mathbf{v2d} = (-1, 4)$  verkrijgen we: [code §7.9.1]

```
u2d = sp.Matrix([2, 3])
v2d = sp.Matrix([-1, 4])
```

We vullen aan met de driedimensionale vectoren  $\mathbf{u3d} = (2, -1, 3)$  en  $\mathbf{v3d} = (-1, 4, -2)$  [code §7.9.1]

```
u3d = sp.Matrix([2, -1, 3])
v3d = sp.Matrix([1, 4, -2])
```

### 7.9.3 Bewerkingen met vectoren

Vectoroptelling, -aftrekking, scalaire vermenigvuldiging en scalaire deling kan eenvoudig met behulp van de standaardrekenkundige operatoren (+, -, \*, /).

### 7.9.4 Vermenigvuldiging van vector met scalar

Zowel symbolisch als met getalwaarden kunnen we vectoren vermenigvuldigen met scalars: [code §7.9.1]

```
a, b = sp.symbols('a b')
v = Matrix([a, b])
2*v
```

geeft

$$\begin{bmatrix} 2a \\ 2b \end{bmatrix}$$

en

```
u = sp.Matrix([1, 2])
a*u
```

geeft

$$\begin{bmatrix} a \\ 2a \end{bmatrix}$$

### 7.9.5 Inwendig Product

Om het inwendig product tussen twee vectoren te berekenen, voorziet SymPy de `dot`-methode. Het resultaat van:

[code §7.9.1]

```
u2d = sp.Matrix([2, 3])
v2d = sp.Matrix([-1, 4])
inwendigproduct = u2d.sp.dot(v2d)
print(inwendigproduct)
```

levert, zoals verwacht: 5

Berekenen van het inwendig product van driedimensionale vectoren gebeurt volledig analoog:

```
inwendig_product3d = u3d.dot(v3D)
print(inwendig_product3d)
```

De uitvoer zou dan -8 moeten zijn.

### 7.9.6 Vectornorm

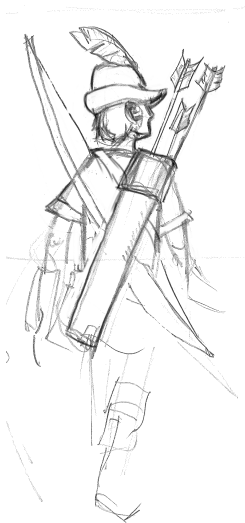
Aangezien het inwendig product van een vector met zichzelf gelijk is aan het kwadraat van zijn norm, kunnen we als volgt te werk gaan: [code §7.9.1]

```
vector = sp.Matrix([1, 2, 3])
norm = sp.sqrt(vector.dot(vector))
norm
```

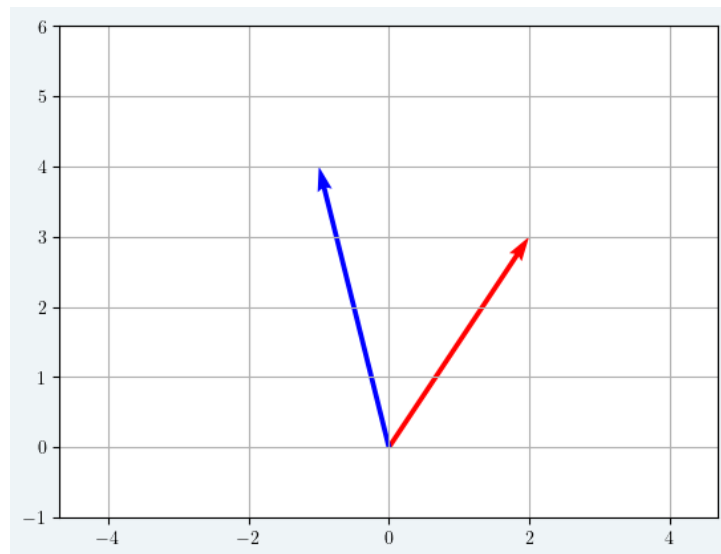
Dit berekent dan  $\sqrt{1^2 + 2^2 + 3^2}$  en komt zo tot  $\sqrt{14}$

### 7.9.7 Visualisatie van tweedimensionale vectoren

Het beeld van de vector als pijl is algemeen bekend. In het verlengde hiervan beschikt de Matplotlib bibliotheek over de functie `quiver`, de pijlenkoker. Daarmee kunnen we meerdere vectoren in een assenstelsel voorstellen. Een volledig overzicht van alle functies kan je hier<sup>16</sup> terugvinden.



(a) Robin Hood met pijlenkoker



(b) Twee vectoren

Figuur 40: Etymologie van *quiver* (links) en hedendaagse toepassing (rechts)

[code §7.9.1]

```
u2d = Matrix([2, 3])
v2d = Matrix([-1, 4])
plt.quiver(0, 0, int(u[0]), int(u[1]), angles='xy',
           scale_units='xy', scale=1, color='r', label='u')
plt.quiver(0, 0, int(v[0]), int(v[1]), angles='xy',
           scale_units='xy', scale=1, color='b', label='v')
plt.xlim(-5, 5)
plt.ylim(-1, 5)
plt.xlabel('X')
plt.ylabel('Y')
```

<sup>16</sup>[https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.quiver.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.quiver.html)

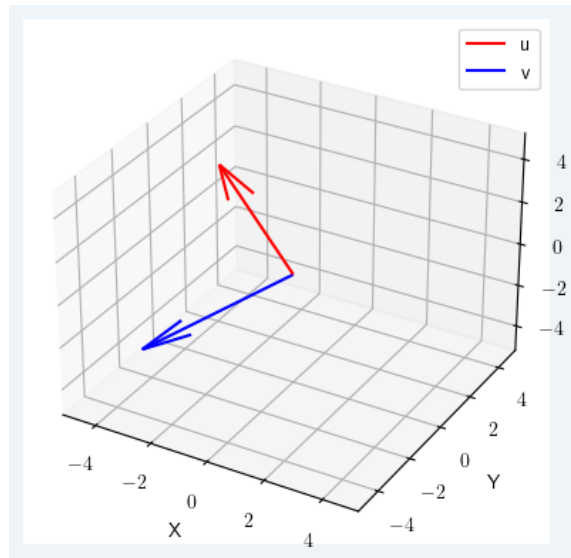
```
plt.legend()
plt.show()
```

- Bereken de som van beide vectoren en stel ze grafisch de resulterende vector voor.
- Bereken het product van de  $\mathbf{u}$  met 3 en stel dit grafisch voor.
- Ontbind de vector  $\mathbf{v}$  in zijn  $x$ - en  $y$ -componenten en stel deze grafisch voor.

## 7.10 Visualisatie van Driedimensionale Vectoren

Om de driedimensionale vectoren te visualiseren, laden we de 3D-plotmogelijkheden van Matplotlib in. Met de driedimensionale vectoren  $u$  en  $v$  in onderstaande code krijgen we de figuren op het einde van deze paragraaf.

```
import sympy as sp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = sp.Matrix([2, -1, 3])
v = sp.Matrix([1, 4, -2])
ax.quiver(0, 0, 0, u[0], u[1], u[2], color='r', label='u')
ax.quiver(0, 0, 0, v[0], v[1], v[2], color='b', label='v')
ax.set_xlim([-5, 5])
ax.set_ylim([-5, 5])
ax.set_zlim([-5, 5])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.show()
```



Figuur 41: Twee vectoren in een driedimensionaal assenstelsel

- Bereken de som van beide vectoren en stel ze grafisch de resulterende vector voor.
- Bereken het product van de  $\mathbf{u}$  met 3 en stel dit grafisch voor.
- Ontbind de vector  $\mathbf{v}$  in zijn  $x$ -,  $y$ - en  $z$ -componenten en stel deze grafisch voor.



## 8 Bijlages

### Overzicht van basisopties van Matplotlib

grootte van de totaalfiguur instellen	<code>from matplotlib.pyplot import figure</code> <code>figure(figsize=(16, 9), dpi=80)</code>
grid tonen	<code>plt.grid()</code>
horizontale lijn tekenen	<code>plt.axhline(y = 0, color = 'black', linewidth=1)</code>
verticale lijn tekenen	<code>plt.axvline(x = 0, color = 'black', linewidth=1)</code>
bereik instellen op x-as (e.g. van -5 tot 5)	<code>plt.xlim(-5,5)</code>
bereik instellen op y-as (e.g. van -2 tot 2)	<code>plt.ylim(-2,2)</code>
asrichting wijzigen	<code>plt.gca().invert_yaxis()</code>
asrichting wijzigen	<code>plt.gca().invert_xaxis()</code>
schaalverdeling op x- en y - as identiek	<code>plt.axis('equal')</code>

Figuur 42: Overzicht van handige sneltoetsen

fontsize	grootte van het lettertype, in punten
familie	font type
backgroundcolor	achtergrondkleur van het tekstlabel
color	tekstkleur
alpha	transparantie van de tekstkleur
rotation	hoek waaronder de tekst gedraaid wordt

Tabel 7: instellingen voor aantekeningen

color	lijnkleur	reëel getal tussen 0.0 (volledig transparant) en 0.1	reëel getal	
alpha	transparantie van de lijn			
linewidth, lw	breedte van een lijn			
linestyle, ls	lijnstijl			
marker			'-'	volle lijn
			'--'	stippellijn
			'.'	puntjeslijn
			'-.'	gemengd
markersize			+,o,*	
			s	
			.	
			1,2,3,4,...	
markerfacecolor				
markeredgewidth				
markeredgecolor				

Tabel 8: instellingen voor eigenschappen van lijnen

Actie	Sneltoets
laat een cel verschijnen met alle shortcuts	<b>h</b>
een nieuwe cel aanmaken <i>onder</i> de geselecteerde cel	<b>b</b>
een nieuwe cel aanmaken <i>boven</i> de geselecteerde cel	<b>a</b>
verander een cel in een markdown cel	<b>m</b>
verander een cel in een code cel	<b>y</b>
selecteer de vorige cel	<b>↑</b>
selecteer de volgende cel	<b>↓</b>
editteer de geselecteerde cel	<b>↩</b>
verlaat de editeermodus	<b>⌫</b>
een cel uitvoeren	<b>ctrl</b> + <b>↩</b>
een cel knippen	<b>x</b>
cellen kopiëren	<b>c</b>
(een) cel(len) boven de geselecteerde cel plakken	<b>v</b>
cellen verwijderen	<b>d</b> <b>d</b>
cellen verwijderen ongedaan maken	<b>Ctrl</b> + <b>Z</b>
een cel opsplitsen	<b>Ctrl</b> + <b>Shift</b> + <b>-</b>
herstart de kernel	<b>0</b> + <b>0</b>
onderbreek een cel in uitvoering	<b>i</b> + <b>i</b>
sla het werkblad op	<b>s</b>

Figuur 43: Overzicht van handige sneltoetsen

## Nog enkele elementaire begrippen

Tabel 9: Nog enkele elementaire begrippen

	commando	resultaat
het begrip oneindig $\infty$	<code>oo</code>	
absolute waarde	<code>Abs(-1)</code>	1
absolute waarde, symbolisch	<code>x = Symbol('x', real=True) Abs(-x)</code> <code>Abs(x**2)</code>	<code>Abs(x)</code> <code>x**2</code>
boogcosinus	<code>acos(1)</code> <code>acos(0)</code> <code>acos(oo)</code>	0 <code>pi/2</code> <code>oo*I</code>
boogcotangens	<code>acot</code> <code>acoth</code>	
Areaalsinus hyperbolicus	<code>asin</code> <code>asinh</code> <code>atan</code> <code>atan2</code> <code>asinh</code> <code>atanh</code> <code>cos</code> <code>cosh</code> <code>cot</code> <code>coth</code> <code>exp</code> <code>base</code> <code>ceiling</code> <code>conjugate</code> <code>arg</code> <code>acot</code>	

Tabel 10: Overzicht van de mogelijkheden van het solve-commando

Los een vergelijking algebraïsch op	$x^2 = y$	$x \in \{-\sqrt{y}, \sqrt{y}\}$
Los een stelsel algebraïsch op	$x^2 + y = 2z, y = -4z$	$\{(x = -\sqrt{6z}, y = -4z), (x = \sqrt{6z}, y = -4z)\}$
Los een of meerdere vergelijkingen numeriek op	$\cos(x) = x$	$x \approx 0.739085133215161$
Los een gewone differentiaalvergelijking algebraïsch op	$y''(x) + 9y(x) = 0$	$y(x) = C_1 \sin(3x) + C_2 \cos(3x)$
Vind de wortels van een polynoom algebraïsch of numeriek	$ax^2 + bx + c = 0$	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
los een matrixvergelijking op	$\begin{bmatrix} c & d \\ 1 & -e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{2e}{ce+d} \\ \frac{2}{ce+d} \end{bmatrix}$
reduceer een ongelijkheid of een stelsel van ongelijkheden voor een enkele variabele algebraïsch	$x^2 < \pi, x > 0$	$0 < x < \sqrt{\pi}$
Los een Diophantische vergelijking algebraïsch op	$a^2 + b^2 = c^2$	$(a = 2pq, b = p^2 - q^2, c = p^2 + q^2)$

<sup>17</sup><https://docs.sympy.org/latest/guides/solving/index.html>

## Griekse symbolen

Symbol	Naam	LaTeX
$\alpha$	alfabet	<code>\alpha</code>
$\beta$	bèta	<code>\beta</code>
$\gamma$	gamma	<code>\gamma</code>
$\delta$	delta	<code>\delta</code>
$\epsilon$	epsilon	<code>\epsilon</code>
$\zeta$	zèta	<code>\zeta</code>
$\eta$	èta	<code>\eta</code>
$\theta$	theta	<code>\theta</code>
$\iota$	jota	<code>\iota</code>
$\kappa$	kappa	<code>\kappa</code>
$\lambda$	lambda	<code>\lambda</code>
$\mu$	mu	<code>\mu</code>
$\nu$	nu	<code>\nu</code>
$\xi$	xi	<code>\xi</code>
$\omicron$	omikron	<code>\omicron</code>
$\pi$	pi	<code>\pi</code>
$\rho$	rho	<code>\rho</code>
$\sigma$	sigma	<code>\sigma</code>
$\tau$	tau	<code>\tau</code>
$\upsilon$	upsilon	<code>\upsilon</code>
$\phi$	phi	<code>\phi</code>
$\chi$	chi	<code>\chi</code>
$\psi$	psi	<code>\psi</code>
$\omega$	omega	<code>\omega</code>
$\Gamma$	Gamma	<code>\Gamma</code>
$\Delta$	Delta	<code>\Delta</code>
$\Theta$	Theta	<code>\Theta</code>
$\Lambda$	Lambda	<code>\Lambda</code>
$\Xi$	Xi	<code>\Xi</code>
$\Pi$	Pi	<code>\Pi</code>
$\Sigma$	Sigma	<code>\Sigma</code>
$\Upsilon$	Upsilon	<code>\Upsilon</code>
$\Phi$	Phi	<code>\Phi</code>
$\Psi$	Psi	<code>\Psi</code>
$\Omega$	Omega	<code>\Omega</code>

# Mogelijkheden van SymPy

- |  |   |   |
|--|---|---|
| <ul style="list-style-type: none"><li>• Elementaire rekenkunde: Ondersteuning voor operatoren zoals +, -, *, /, ** (macht).</li><li>• Vereenvoudiging</li><li>• Uitbreiding</li><li>• Functies: Goniometrisch,</li></ul> | <ul style="list-style-type: none"><li>hyperbolisch, exponentieel, wortels, logaritmen, absolute waarde, sferische harmonischen, faculteiten en gammafuncties, zetafuncties, veeltermen, speciale functies, enz.</li></ul> | <ul style="list-style-type: none"><li>• Substitutie</li><li>• Getallen: Willekeurig nauwkeurige gehele getallen, rationale getallen en zwevende-kommagetallen.</li><li>• Niet-commutatieve symbolen</li><li>• Patroonherkenning</li></ul> |
|--|---|---|

## Veeltermen

- |   |  |  |
|---|--|--|
| <ul style="list-style-type: none"><li>• Elementaire rekenkunde: deling, grootste gemene deler, enz.</li></ul> | <ul style="list-style-type: none"><li>• Factoren</li><li>• Vrije factoren decompositie</li><li>• Gröbner-bases</li></ul> | <ul style="list-style-type: none"><li>• Decompilatie van deelbreuken</li><li>• Resultanten</li></ul> |
|---|--|--|

## Calculus

- |   |  |  |
|---|--|--|
| <ul style="list-style-type: none"><li>• Grenzen: <math>\lim_{x \rightarrow 0} x \log(x) = 0</math></li><li>• Afgeleiden</li></ul> | <ul style="list-style-type: none"><li>• Integratie: Het gebruikt de uitgebreide Risch-Norman</li></ul> | <ul style="list-style-type: none"><li>heuristiek.</li><li>• Taylor (Laurent) reeks</li></ul> |
|---|--|--|

## Oplossen van vergelijkingen

- |  |  |   |
|--|--|---|
| <ul style="list-style-type: none"><li>• Veeltermvergelijkingen</li><li>• Algebraïsche vergelijkingen</li></ul> | <ul style="list-style-type: none"><li>• Differentiaalvergelijkingen</li><li>• Verschilvergelijkingen</li></ul> | <ul style="list-style-type: none"><li>• Stelsels van vergelijkingen</li></ul> |
|--|--|---|

## Combinatoriek

- |   |  |   |
|---|--|---|
| <ul style="list-style-type: none"><li>• Permutaties</li><li>• Combinaties</li></ul> | <ul style="list-style-type: none"><li>• Partities</li><li>• Deelverzamelingen</li><li>• Permutatiegroepen:</li></ul> | <ul style="list-style-type: none"><li>Polyhedraal, Rubik's kubus, symmetrisch, enz.</li><li>• Prufer- en Gray-codes</li></ul> |
|---|--|---|

## Discrete Wiskunde

- |   |  |   |
|---|--|---|
| <ul style="list-style-type: none"><li>• Binomiale coëfficiënten</li><li>• Sommaties</li></ul> | <ul style="list-style-type: none"><li>• Producten</li><li>• Getaltheorie: Genereren van priemgetallen,</li></ul> | <ul style="list-style-type: none"><li>priemgetaltesten, integerfactorisatie, enz.</li><li>• Logische expressies</li></ul> |
|---|--|---|

## Matrices

- |   |  |  |
|---|--|--|
| <ul style="list-style-type: none"><li>• Elementaire rekenkunde</li><li>• Eigenwaarden/eigenvectoren</li></ul> | <ul style="list-style-type: none"><li>• Determinanten</li><li>• Inversie</li></ul> | <ul style="list-style-type: none"><li>• Oplossen van vergelijkingen</li><li>• Abstracte expressies</li></ul> |
|---|--|--|

## Geometrische Algebra

- |  |  |   |
|--|--|---|
| <ul style="list-style-type: none"><li>• Geometrie<ul style="list-style-type: none"><li>– Punten, lijnen, stralen, segmenten, ellipsen,</li></ul></li></ul> | <ul style="list-style-type: none"><li>cirkels, veelhoeken, enz.</li><li>– Snijpunten</li></ul> | <ul style="list-style-type: none"><li>– Raaklijnen</li><li>– Similariteit</li></ul> |
|--|--|---|

## Plotten

- |  |   |  |
|--|---|--|
| <ul style="list-style-type: none"><li>• Coördinatenmodi</li><li>• Plotten van geometrische</li></ul> | <ul style="list-style-type: none"><li>objecten</li><li>• 2D en 3D</li></ul> | <ul style="list-style-type: none"><li>• Interactieve interface</li><li>• Kleuren</li></ul> |
|--|---|--|

## Natuurkunde

- Eenheden
- Mechanica

- Kwantummechanica
- Gaussische optica

- Pauli-algebra

## Statistiek

- Normale verdelingen
- Uniforme verdelingen
- Kansberekening

## Weergave

- Mooie weergave:  
ASCII/Unicode-weergave,
- LaTeX
- Genereren van code: C,  
Fortran, Python

## Referenties

- [1] Douglas C. Giancoli. *Natuurkunde*. Pearson, 2014.
- [2] Robert Johansson. *Numerical Python*. Apress, 2015.
- [3] Bayen Kong, Siau. *Python Programmign and Numerical Methods*. Academic Press, 2021.
- [4] Eric Matthes. *Crash Course Porgrammeren in Python*. No Starch Press, 2018.
- [5] Amit Saha. *Doing Math With Python*. No Starch Press, 2015.