|  |  |
| --- | --- |
| Biostatica  Door Mark Schrauwen en Alistair Vardy | Matlab Wk3.1 |

Inhoudsopgave

[Versiebeheer 2](#_Toc492380240)

[1 Introductie 3](#_Toc492380241)

[2 Herhalen 4](#_Toc492380242)

[2.1 For-lus 4](#_Toc492380243)

[2.1.1 Het optellen van de x opeenvolgende waardes van een vector 4](#_Toc492380244)

[2.1.2 Waar zijn we ook alweer mee bezig? 7](#_Toc492380245)

[2.1.3 Uitwerking 9](#_Toc492380246)

[2.1.4 Nog slimmer 11](#_Toc492380247)

[2.1.5 Teveel for-lus gebruiken 12](#_Toc492380248)

[2.2 While-loop 13](#_Toc492380249)

[2.2.1 Afsluitende opmerkingen 14](#_Toc492380250)

[2.3 Vragen en opdrachten 15](#_Toc492380251)

[2.4 Antwoorden 16](#_Toc492380252)

[3 Tekenen in Matlab 18](#_Toc492380253)

[3.1 Een eerste aanzet 18](#_Toc492380254)

[3.2 Het verfraaien van een plot 20](#_Toc492380255)

[3.2.1 Kleuren van de lijnen 20](#_Toc492380256)

[3.3 Het afsluiten van een plot 21](#_Toc492380257)

[3.4 Meerdere plots in een plot 22](#_Toc492380258)

[3.5 Verschillende plots in een aparte GUI 24](#_Toc492380259)

[3.6 Het verfraaien van een plot 24](#_Toc492380260)

[3.7 Het opgeven van een plotbereik 25](#_Toc492380261)

[3.8 Andere veelvoorkomende type plots 26](#_Toc492380262)

[3.8.1 Area plot 26](#_Toc492380263)

[3.8.2 Histogram 26](#_Toc492380264)

[3.9 Vragen en Opdrachten 28](#_Toc492380265)

[3.10 Antwoorden 28](#_Toc492380266)

[4 Bronnen 30](#_Toc492380267)

# Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Beschrijving | Door |
| 0.0 | 11-07-2017 | Eerste versie | Mark Schrauwen |
| 0.1 | 02-10-2017 | Review versie | Bart van Trigt |
| 0.2 | 10-02-2017 | Commentaar Bart verwerkt + andere kleine verbeteringen. | Mark Schrauwen |

# Introductie

Vorige week heb je een aantal zaken geleerd. Je bent bekend geraakt met logische operatoren en relationele operatoren. Het is duidelijk geworden dat je in Matlab vaak beslissingen en herhalingen uitvoert. Je hebt geleerd hoe je beslissingen in Matlab kunt maken m.b.v. de if-else-statement. Daarnaast heb je geleerd dat het if-gedeelte van een if-else-statement wordt uitgevoerd als de logische-expressie (de test) true is. Als dat niet zo is dan wordt het else-gedeelte uitgevoerd.

Ook heb je gezien dat sommige beslissingen niet op basis van een binaire waarde zijn maar op basis van verschillende waardes. Deze beslissingen kun je afhandelen m.b.v. een switch-statement.

Deze week gaan we verder met herhalingen in Matlab. Ook krijg je te zien hoe we in Matlab kunnen tekenen. Je leert deze week hoe je data uit een meting kunt visualiseren. Je zult opnieuw in aanraking komen met verschillende Matlab functies. Merk op, dat de door Matlab aangeleverde functies de bouwstenen vormen van veel van de programma’s die je gaat maken.

Deze week wordt ook nog speciale aandacht gegeven aan matrices en vectoren. Deze type variabelen hebben Matlab populair gemaakt. Matlab kan namelijk heel goed, in tegenstelling tot andere programmeertalen, omgaan met matrices en vectoren. Er zijn dan ook allerlei functies beschikbaar die het omgaan met deze variabelen vergemakkelijkt. Er zal in dat gedeelte onder andere worden uitgelegd waarom het doen van berekeningen m.b.v. matrices zo populair is.

Als laatste leer je deze week hoe je om moet gaan met de door ons aangeleverde functies. Dat zijn functies die niet zijn geschreven door Matlab. Aan het einde van deze lesweek krijg je wederom een individuele opdracht waarin je de kennis opgedaan moet toepassen.

**Zie je een fout? Of heb je een aanbeveling dan horen we dat graag! Stuur dan een e-mail naar** [**mjschrau@hhs.nl**](mailto:mjschrau@hhs.nl) **en wij passen het dan z.s.m. aan.**

# Herhalen

Herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen, herhalen en herhalen.

Om een stuk Matlab code daadwerkelijk een programma te laten zijn moeten er beslissingen worden gemaakt en/of herhalingen worden uitgevoerd. Inmiddels weet je hoe beslissingen worden gemaakt. Nu gaan we kennis maken met herhalingen.

## For-lus

Als je acties moet herhalen dan weet je vaak ‘hoe vaak’ je deze acties moet uitvoeren. Als je strafregels moet schrijven dan krijg je vaak een getal te horen. Bijvoorbeeld, schrijf 100 x op ‘ik mag niet vergeten dat Matlab een belangrijk vak is voor een Bewegingstechnoloog’.

Bij het herhalen van activiteiten in Matlab weet je bijna altijd hoe vaak je een actie moet herhalen. Stel dat je een element van een Matlab vector op basis van een bepaalde waarde moet ophogen; je mag enkele de oneven indices van een vector ophogen met de waarde ‘1’. Hoe doe je dit dan gemakkelijk?

Stel dat je elk element van een Matlab vector moet ophogen. De waarde waarmee één element van een vector wordt opgehoogd hangt af van de waarde van dat ene element. Bijvoorbeeld bij element 2 van een vector moet de waarde 2 worden opgeteld. Bij element 3 moet de waarde 3 worden opgeteld. Dan kun je deze problemen, niet gemakkelijk, oplossen m.b.v. een vector operator. Op zulke momenten komt automatisch een for-lus om de hoek kijken.

Stel dat je van een vector de eerste 3 elementen moet optellen en in een nieuwe vector moet stoppen. Hoe doe je dit met een operator? Dat is niet mogelijk.

Een for-lus wordt gebruikt als je weet hoe vaak iets herhaalt moet worden.

### Het optellen van de x opeenvolgende waardes van een vector

In dit voorbeeld moet een programma worden ontworpen dat een drietal opeenvolgende waardes van een vector bij elkaar optelt. Stel dat we een variabele genaamd *vectorA* hebben waarin de waardes van 0 tot en met 9 staan.

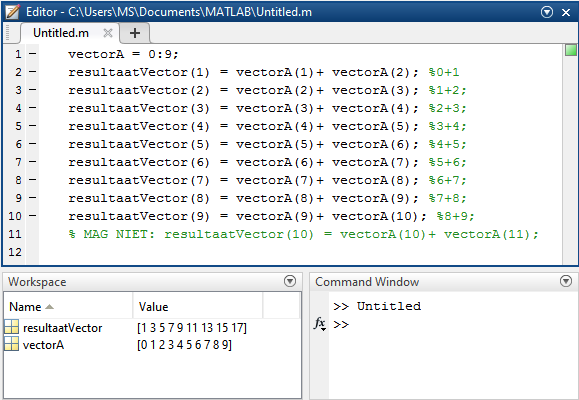
Stel dat we elke twee opeenvolgende waardes willen optellen. Dan krijgen we: 0+1, 1+2, 2+3, … , 8+9. Merk op, dat de resultaat vector één element kleiner is dan de oorspronkelijke vector. Immers er volgt op 9 niet een nieuw getal. We tellen 9 niet opnieuw op bij een ander getal. Dat getal wordt overgeslagen en dus is er één getal minder in de nieuwe vector.

De code van het vorige voorbeeld is te zien in Figuur 1. Er vallen direct een aantal zaken op aan deze code. Zo is de code erg lang. Er zijn maar liefst 10 regels nodig. Wat nog meer opvalt, er zit veel structuur en regelmaat in de code. Als je zoiets ziet in een zelfgeschreven dan weet je dat een for-lus kunt gaan gebruiken.

*Maar wat is een for-lus?*

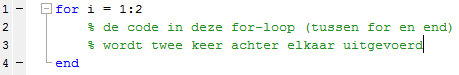
Een for-lus is een programmeerconstructie die de programmeur (jij) in staat stelt gemakkelijk een x-aantal herhalingen achter elkaar uit te voeren.

*Hoe gebruik je een for-lus?*



Figuur : Een naïeve manier om een aantal herhalingen uit te voeren in Matlab. In dit voorbeeld worden twee opeenvolgende getallen van een vector bij elkaar opgeteld in Matlab. Waarom kan de laatste regel niet worden uitgevoerd?

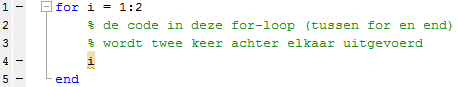
In Figuur 2 is te zien hoe in Matlab een for-lus moet worden geschreven. Zoals je kan zien begin je met het woord ‘for’ en elke loop eindigt met ‘end’. Het voorbeeld laat echter nog niet zien wat er daadwerkelijk gebeurd.



Figuur : dit is hoe een for-lus in Matlab eruit ziet.

**Typ de code in Figuur 2 over en voer de code uit**

Er gebeurt nog helemaal niets. Dat is in overeenstemming met de code want er staat nog niet in de for-lus.

****

Figuur : het voorbeeld in Figuur 2 is behoorlijk uitgebreid.

**Voer het voorbeeld in Figuur 3 uit**

*Wat valt op?*

Je hebt gezien dat er in de Command Window (CW) twee waardes zijn afgedrukt namelijk 1 en 2.

*Snap je waar deze waardes vandaan komen?*

Als je in Matlab een variabele in een regel uitvoert dan wordt de waarde die de variabele bevat afgedrukt in het CW (dat noemen we een echo). In dit specifieke geval worden de waardes 1 en 2 afgedrukt.

**Verander de waardes van 1 en 2 naar respectievelijk 2 en 8 en voer de code uit**

Je merkt op dat er een relatie is tussen de regel i = 2:8 en de i die in de for-lus staat. Elke herhaling wordt de variabele i opgehoogd met de waarde 1. Dus volgens het laatste voorbeeld wat jij hebt uitgevoerd, is de waarde van de variabele i exact 7 keer afgedrukt in het CW.



Figuur : een focus op de variabele i. Deze variabele krijgt elke herhaling een andere waarde. De eerste keer is de waarde 2 de tweede keer dat de for-lus wordt uitgevoerd is de waarde 3. Dit wordt herhaald tot en met dat i de waarde 8 heeft gekregen.

*Wat wordt er afgedrukt in het CW als de code in* Figuur 5 *wordt uitgevoerd? Bedenk dit eerst zelf! Kijk voor het antwoord op de volgende pagina.*



Figuur : De code is nu aangevuld. Wat wordt er nu afgedrukt in het CW?

Om te zien welke waardes de variabele i krijgt in de for-lus kun je 2:8 uitvoeren in het CW.

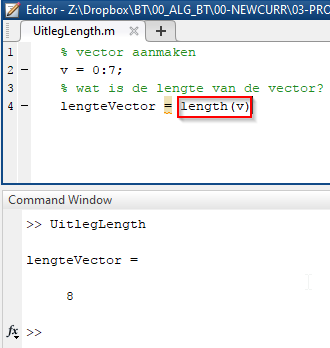
|  |
| --- |
| *Dit wordt afgedrukt als de code in Figuur 5 wordt uitgevoerd:*  *ans =*  *4*  *.*  *.*  *.*  *ans =*  *10*  *De for-lus herhaalt zich (2 tot en met 8) 7 keer. Elke herhaling wordt de variabele i met 1 opgehoogd: 2, 3, … , 8. De eerste keer dat de for-lus wordt uitgevoerd is i gelijk aan de waarde 2. Tel bij die waarde 2 op en je krijgt vier.* |

### Waar zijn we ook al weer mee bezig?

We willen het voorbeeld in Figuur 1 zo schrijven zodat het een stuk slimmer en korter is met behulp van een for-lus.

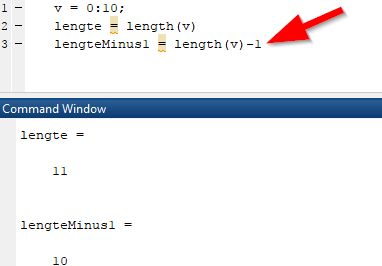
Hier volgen een aantal tips die je helpen om het voorbeeld, dat is gegeven slim op te pakken. De eerste handige standaard Matlab functie is *length()*. Deze functie kan de lengte van een vector (of matrix) opvragen.

Stel je hebt een vector bestaande uit de getallen 0 tot en met 10. Je wilt weten wat de lengte is van deze vector. Dan krijg je het onderstaande voorbeeld:



Figuur : een voorbeeld van de functie length().

De functie length() geeft een getal terug. Dat getal representeert de lengte van een vector. Je kunt dus ook direct een berekening doen met deze functie:

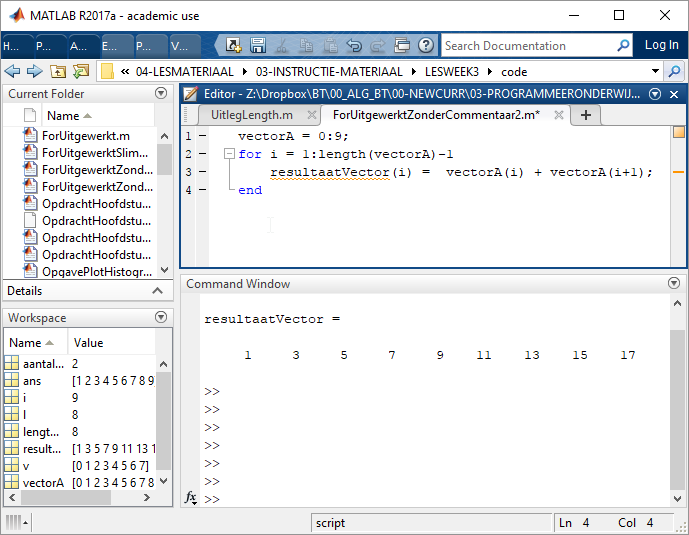


Figuur : een voorbeeld dat laat zien dat je ook direct een berekening kunt uitvoeren op de functie length(). In dit geval de berekening length() minus de waarde 1`.

**Probeer nu zelf in Matlab m.b.v. een for-lus het probleem van** Figuur 1 **op te lossen. De uitwerking staat op de volgende pagina.**

### Uitwerking

In Figuur 8 is te zien hoe het probleem in Figuur 1 veel slimmer en duidelijk is geschreven. Er zijn echter



Figuur : het uitgewerkte voorbeeld, neem de code goed door. Je ziet dat deze code een stuk efficiënter is dan de code in Figuur 1.

**Typ dit voorbeeld over (als je dit nog niet hebt gedaan ☹ )**

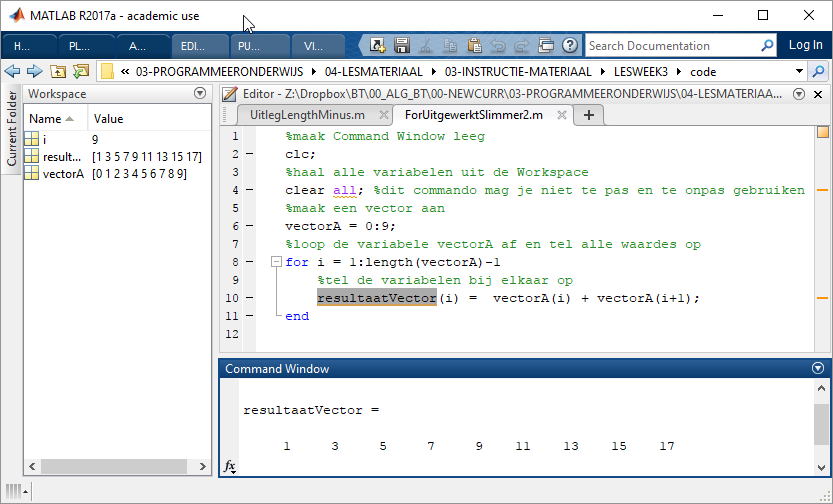
**Zet een breakpoint op regel 3**

**Voor het programma stap voor stap uit (druk telkens op F11)**

*Wat zijn de verschillen met de vorige versie?*

Merk op dat dit programma een heel stuk korter is dan het programma in Figuur 1. Dat komt omdat we nu gebruik maken van de for-lus. Er zitten echter nog wel een aantal kleine onderdelen in de code van Figuur 8 die enige verklaring behoeven.

Sterker nog, de code in Figuur 8 is niet heel erg duidelijk en dat komt door het gebrek aan commentaar. Daarom hebben we de code in Figuur 8 aangepast (zie Figuur 9)

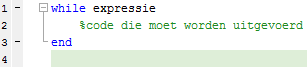


Figuur : de code van Figuur 8 maar dan wat duidelijk a.d.h.v. commentaar

## While-loop

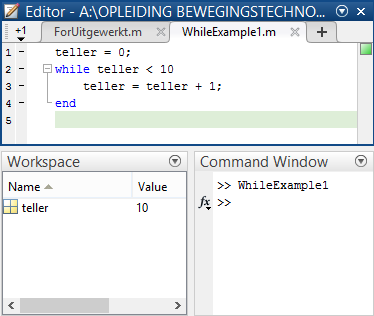
De for-lus is inmiddels bekend. Maar er is ook nog een andere Matlab constructie om herhalingen uit te voeren. Niet iedereen realiseert zich het verschil tussen een for-lus en een while-loop. Een for-lus wordt gebruikt als je het aantal herhalingen **weet**. Een while-loop wordt gebruikt als je het aantal herhalingen **niet weet**.

Je kunt probleemloos je hele leven programma’s ontwerpen zonder gebruik te maken van while-loops. Dat geldt voor bijna elke programmeertaal. Echter, zijn er veel scenario’s denkbaar waarin de while-loop simpelweg handiger is. Dat zijn vaak situaties waarin een gebruiker om input moet worden gevraagd, of situaties waarin data moet worden ingelezen vanuit een apparaat. Dat maakt het weer moelijker om goede voorbeelden te verzinnen. Daarom beperken we ons enkel tot het gebruik van de while-loop.



Figuur : het ontwerp van een while-loop

Zoals eerder verteld worden in Matlab en andere programmeertalen herhalingen uitgevoerd zolang de logische-expressie *true* is. Dat werkt ook zo voor de while-loop.



Figuur : een nieuw voorbeeld waarin de while-loop wordt gebruikt.

**Bekijk de code in Figuur 12.**

*Wat is de waarde in de variabele teller als de while-loop helemaal klaar is met uitvoeren?*

De oplettende lezer ziet dat het antwoord op de vorige vraag in Figuur 12 is te zien, namelijk 10.

De while-loop wordt net zo lang uitgevoerd als dat de logische-expressie in dit geval: teller < 10 waar is. Indien de gebruiker vergeet de waarde teller op te hogen binnenin de while-loop dat blijft het programma oneindig lang doorgaan.

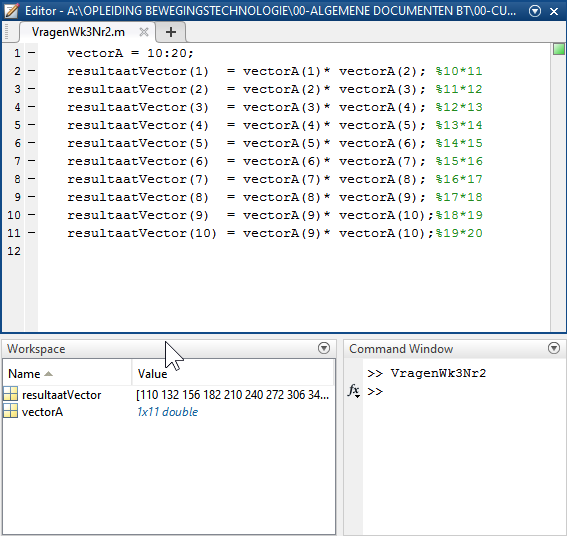
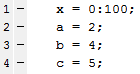
### Afsluitende opmerkingen

De for-lus heeft niet zoveel aandacht gekregen als de while-loop. Dat hoeft ook niet. De werking van beide herhalingsmogelijkheden lijken ook veel op elkaar. Je weet immers dat er wordt herhaald zolang de expressie *true* is. Dat lijkt heel erg op de wijze waarop een if-construct werkt. Je weet dat de if-code van een if-else-statement wordt uitgevoerd als de expressie *true* is.

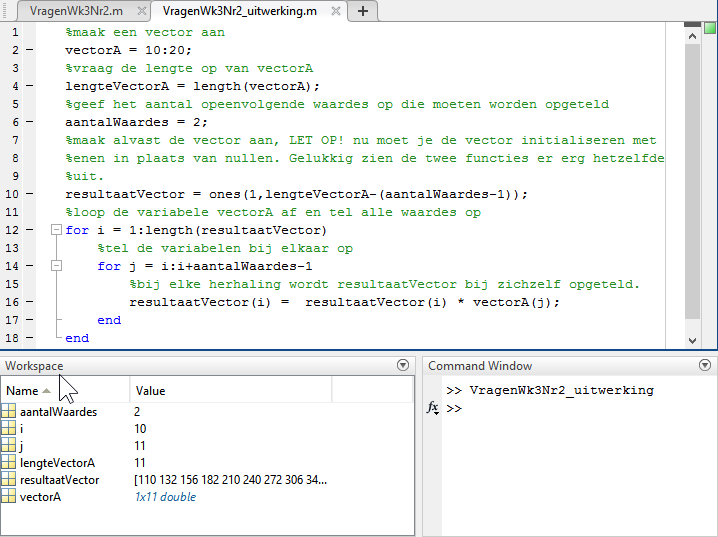
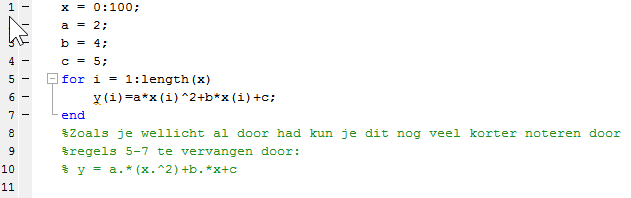
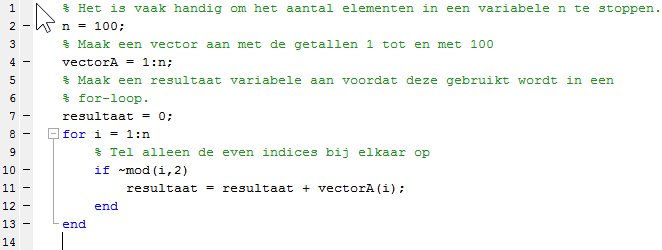
Dat betekent dat het in een while-loop veel gemakkelijker is om relationele en logische operatoren te gebruiken. Dat is een ander argument waarom while-loops van toepassing kunnen komen.

In de vragen/opdrachten gedeelte van dit hoofdstuk zal je ook enkele while-loop gerelateerde vragen tegenkomen en ook kun je dit type vragen verwachten op het tentamen.

## Vragen en opdrachten

1. Bekijk de code in Figuur 3. Waarom wordt in deze code in de CW de waardes 1,2 afgedrukt?
2. Is het mogelijk om de code van Figuur 3 uit te breiden, zodat de i met stappen van 0.1 wordt afgedrukt?
3. Zie onderstaande afbeelding. Dit voorbeeld lijkt heel erg op de code in Figuur 3. Maar sommige elementen zijn toch echt anders. Pas de code aan zodat de berekeningen in onderstaand figuur worden uitgevoerd m.b.v. een for-lus. Zorgt dat het niet uitmaakt met welke waardes *vectorA* is geinitialiseerd.
4. Zie de nevenstaande code: . Bereken met behulp van een for-lus y=a\*x2+b\*x+c;
5. Schrijf een code die alleen de even indices (in plaats van oneven) van een vector bij elkaar optelt. Tip: om te bepalen of een index even of oneven is gebruik je: ~mod(i,2) . De vector loopt van 1 tot en met 100.

## Antwoorden

1. De eerste regel van de for- loop i = 1:2 bepaalt hoe vaak de code wordt uitgevoerd. In dit geval van i = 1 tot en met i = 2. Elke herhaling wordt de variabele i geprint. Elke herhaling (iteratie) wordt i opgehoogd. Dus de eerste keer is de i gelijk aan 1. De tweede keer is de i gelijk aan 2.
2. Ja dat is mogelijk. De code i = 1:2 is zoals we in week 1 hebben gezien het genereren van een vector waarbij de elementen telkens met 1 toenemen. Deze code kan worden uitgebreid naar i = 1:0.1:2. Je kunt de for-lus in Matlab zien als een code construct die *ieder* element van een vector doorloopt. Ongeacht de waardes van die vector. Echter zal je deze code niet heel vaak tegenkomen omdat een for-lus meestal wordt gebruikt om door de elementen van een vector heen te lopen.
3. Dit is de uitwerking. Merk op dat het heel erg lijkt op de eerder aangeleverde uitwerkingen. De grootste verandering zit in het vervangen van de +-operator en de \*-operator en het gebruik van ones() in plaats van zeros().
4. 
5. Hieronder staat de uitwerking. Je hebt een if-statement nodig om bij elke herhaling een beslissing te kunnen maken.
6. Dat je binnenin een for-lus één of meerdere for-lus gebruikt.

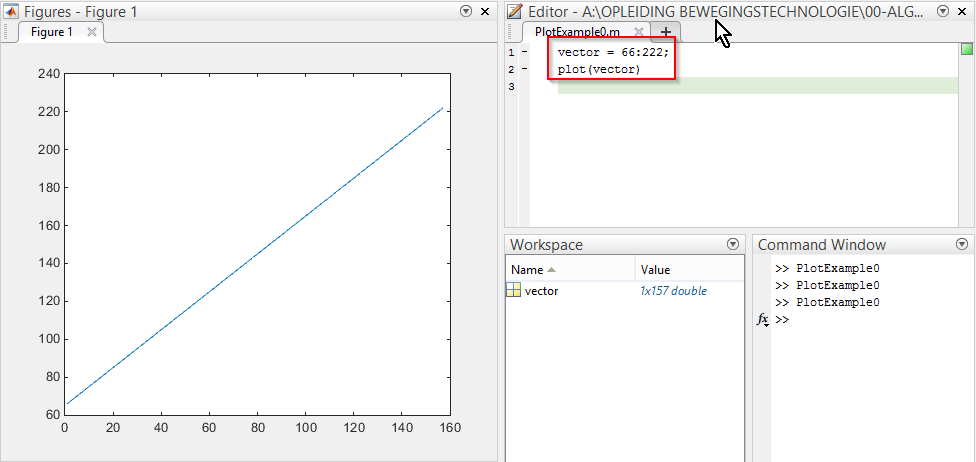
# Tekenen in Matlab

Matlab is meer dan alleen een programmeeromgeving. Matlab kan bijvoorbeeld verbinding maken met bepaalde aansturingen van robots. Matlab kan verbinding maken met de Arduino (een ontwikkelomgeving waar je in blok 3 kennis mee gaat maken). Maar Matlab kan ook goed data visualiseren. In dit hoofdstuk gaan we kijken naar hoe je met Matlab plots kunt maken.

## Een eerste aanzet

*Wanneer zou je in Matlab data moeten plotten?*

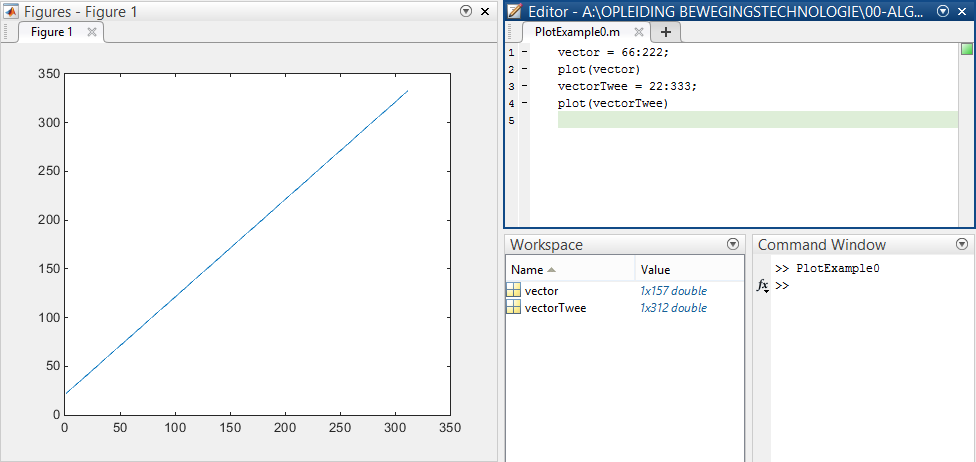
Het visualiseren van data is belangrijk als mensen data moeten inspecteren. Zo weet de student of er bijvoorbeeld tijdens een meting de waarde van een bepaalde sensor boven 100 geweest. Het visualiseren kan ook handig zijn om het gedrag van bepaalde functies te onderzoeken of wiskundige vergelijkingen. In de praktijk zal je heel vaak plots maken om snel inzicht te krijgen in hoe een bepaalde dataset zich gedraagt.



Figuur : voorbeeld van een plot in Matlab. Merk op dat je maar een paar regels code nodig hebt.

|  |
| --- |
| *Maar wat moeten we doen als we twee lijnen willen tekenen?* |

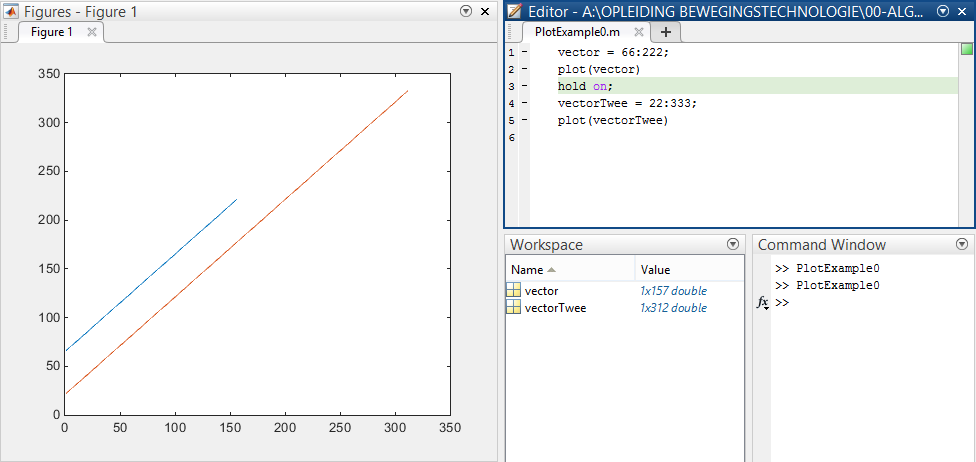
Een naïeve maar voor de hand liggende manier om dat te doen wordt weergegeven in Figuur 14.



Figuur : De eerste poging om twee vormen te tekenen in één grafiek. Merk op dat de assen andere waardes hebben in vergelijking met Figuur 13.

|  |
| --- |
| *Wat gaat hier mis?* |

Het probleem is dat het plotten van de variabele *vectorTwee* de plot van de variabele *vector* heeft overschreven. Deze plot is dus niet meer getekend in de plot. Elke keer als de functie plot() wordt aangeroepen wordt standaard de vorige oude plot overschreven. *Hoe kan dit worden voorkomen?* Het antwoord op die vraag is te zien in Figuur 15:

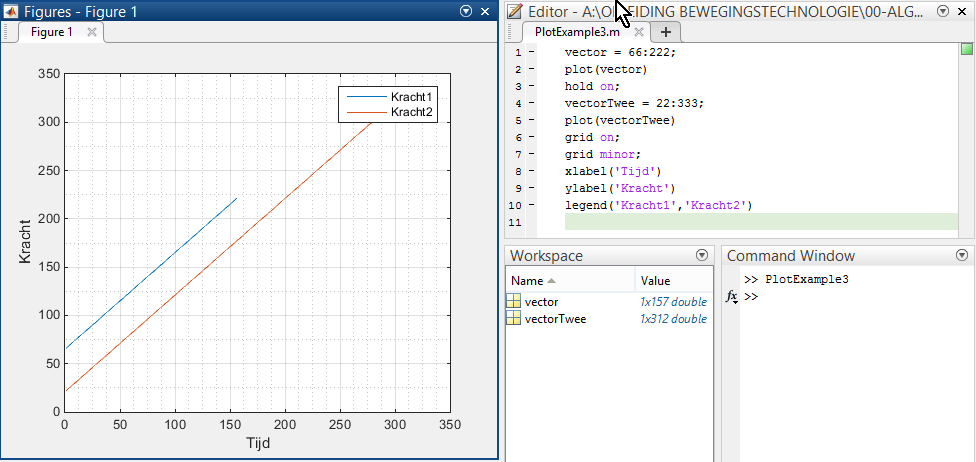


Figuur : Een voorbeeld van twee vectoren afbeeld in een plot dankzij de hold on; code.

De truc is om Matlab te vertellen dat de eerste variabele *vector* moet worden onthouden en niet mag worden overschreven. Dat doen we aan de hand van de **hold on;** code. Je kunt deze Matlab functionaliteit onderzoeken m.b.v. **help hold**.

## Het verfraaien van een plot

De voorgaande plots zijn erg onduidelijk. Wat laat de x-as zien? Wat laat de y-as zien? Wat betekenen de twee verschillende vectoren? Dat kunnen we nu niet goed zien. Met een aantal extra regels code kunnen we een plot beter leesbaar en begrijpelijk maken (zie Figuur 16).

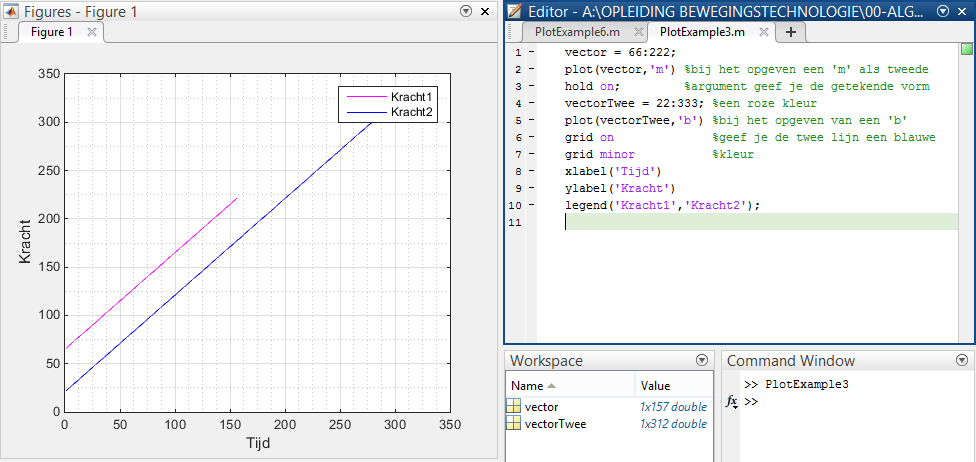


Figuur : De plot van Figuur 15 verfraaid en beter leesbaar gemaakt.

Met behulp van regel 6 en 7 in Figuur 16 voegen we een grid toe aan de plot. Het grid maakt het aflezen van waardes van een vector gemakkelijk. Ook willen we graag de x-as en de y-as een voorzien van een naam. Dat doen we in regel 8 en 9 in Figuur 16. Je ziet wederom dat Matlab al haar functionaliteit inpakt m.b.v. functies. Als laatste kunnen we in een plot() ook nog opgeven welke signalen wat voorstellen. In regel 10 van Figuur 16 kunnen we een legenda maken m.b.v. de functie legend().

### Kleuren van de lijnen

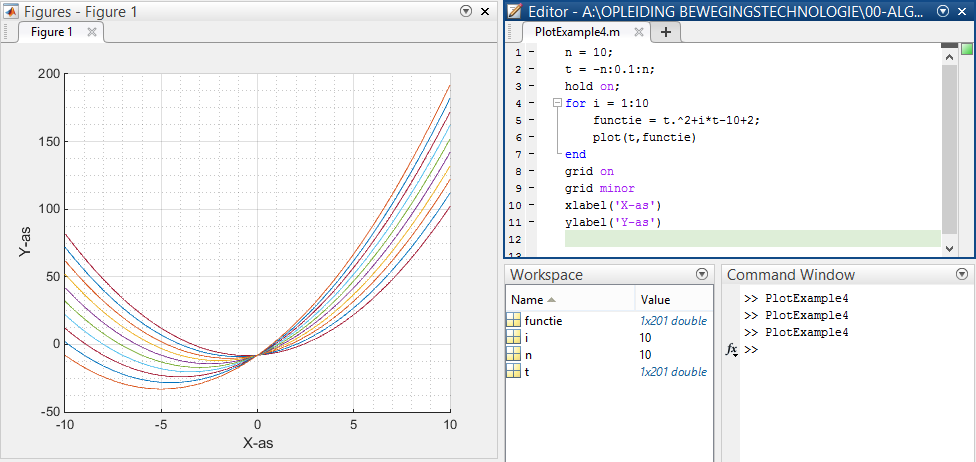
Je hebt inmiddels gemerkt dat Matlab zelf kleuren toekend aan de lijnen die worden getekend in een plot. Je hebt hier echter zelf ook wat over te zeggen (zie Figuur 17).



Figuur : je kunt ook zelf de kleuren in een plot definieren

**Maak zelf een plot met verschillende zelf aangemaakte signalen en een legenda**

Het is nu mogelijk om de kennis opgedaan in de vorige weken toe te passen om bijvoorbeeld meerdere kwadratische functies in één plot te tekenen:



Figuur : M.b.v. een for-lus worden er gemakkelijk 10 verschillende kwadratische functies getekend.

**Zorg dat je de code in Figuur 18 begrijpt!**

## Het afsluiten van een plot

Zoals je eerder hebt gezien kan het CW worden leeggemaakt m.b.v. de clear all; code. Maar het uitvoeren van deze code zorgt er niet voor de een aangemaakte plot wordt weggehaald.

|  |
| --- |
| *Hoe doen we dit dan wel?* |

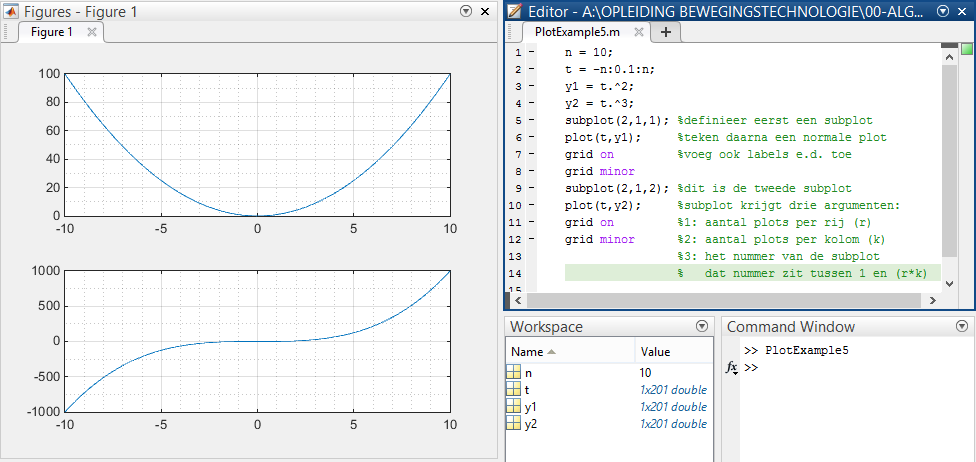
Een plot kan worden afgesloten met de code close all;.

**Maak op basis van voorgaande informatie zelf in Matlab een plot aan**

**Sluit deze plot af m.b.v. *close all***

## Meerdere plots in een plot

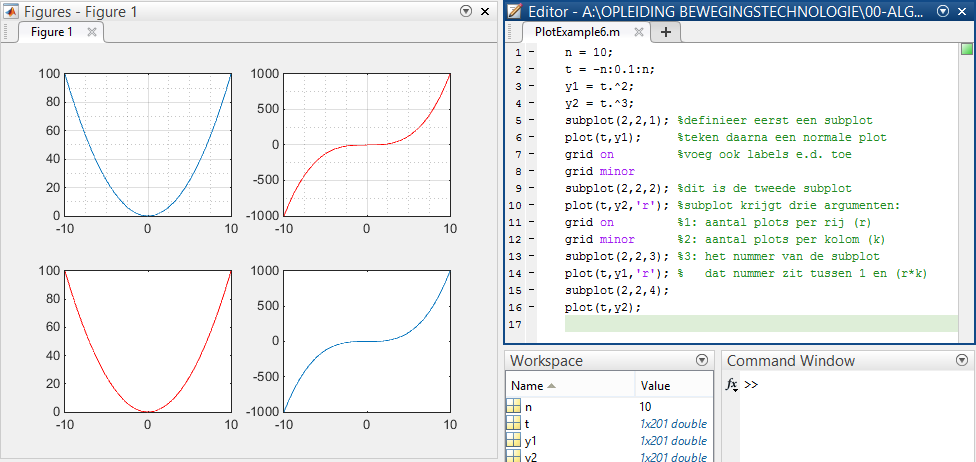
Tot nu heb je telkens één plot gemaakt in één GUI. Het is ook mogelijk om meerder plots in één GUI te plaatsen. Dan kan m.b.v. de functie subplot(r,k,n). Deze functie krijgt drie argumenten. Het eerste argument (r) wordt gebruikt om het aantal subplots in rijen weer te geven in een GUI. Het tweede argument (k) wordt gebruikt om het aantal subplots in de kolommen weer te geven. Het laatste argument (n) daarmee selecteer je een subplot waarin moet worden getekend. Het aantal subplots hangt af van r en k. Dus om een bepaalde subplot te selecteren zijn dit de opties: 1 tot (r\*k).



Figuur : een voorbeeld van subplots, meerdere plots in één GUI.

*Is het ook mogelijk om meer dan twee subplots te tekenen?*

Het onderstaande figuur (Figuur 20) geeft antwoord op die vraag:



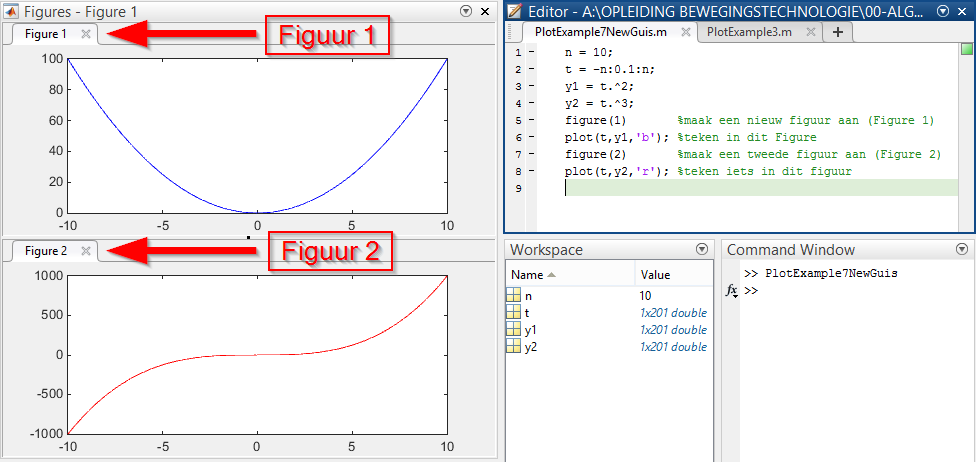
Figuur : meerder plots in één figuur.

|  |
| --- |
| *Kun je ook een 3 bij 3 plot maken (een subplot met 9 grafieken)?* |

**Probeer dit zelf te maken op basis van de voorgaande uitleg.**

## Verschillende plots in een aparte GUI

Soms is het wenselijk om signalen te plotten in hun eigen GUI. Dat is met Matlab ook mogelijk. In het onderstaande voorbeeld is te zien hoe je dit doet:

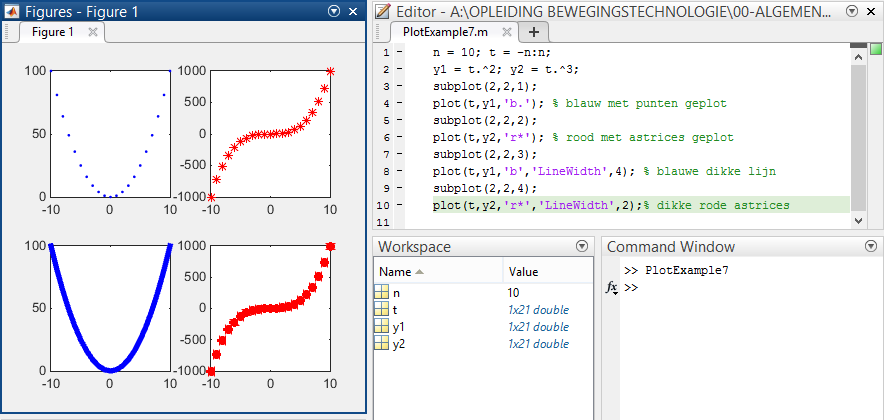


Figuur : in plaats van subplots is in dit voorbeeld gebruik gemaakt van twee verschillende GUI’s.

Het verschil tussen Figuur 19 en Figuur 21 is dat nu elke grafiek in een eigen GUI staat. Dat betekent dat één van deze GUI’s kan worden gesloten zonder dat de andere GUI’s daar last van ondervinden.

## Het verfraaien van een plot

Inmiddels hebben we al een paar manieren geleerd om een plot te verfraaien. Zo heb je specifieke lijnen een eigen kleur gegeven. Ook heb je een grid toegevoegd aan een plot en heb je een legenda toegevoegd. Maar er zijn nog meer technieken om een plot verder te verfraaien. Of anders gezegd, er zijn nog meer manieren om data te plotten.



Figuur : Een aantal subplots met verschillende verfraaiingen. Lees zelf de code goed door om te begrijpen welke veranderingen in de code zijn doorgevoerd.

Je kunt zoals je in Figuur 22 ziet verschillende verfraaiingen aan je plot toevoegen. Zo kun je de lijndikte van een plot veranderen.

*Hoe moet dit?* **Leidt het antwoord af uit de code in Figuur 22 en test dit zelf uit.**

Je hebt inmiddels opgemaakt uit Figuur 22 dat je niet altijd lijnen hoeft te tekenen. Je kunt ook met bepaalde symbolen werken.

*Hoe kun je in plaats van een lijn enkel puntjes tekenen in een plot?* **Leidt het antwoord af uit de code in Figuur 22 en test dit zelf uit.**

## Het opgeven van een plotbereik

Het is je misschien niet opgevallen, maar tot nu toe als je iets hebt geplot dan werd de hele vector weergegeven. Maar is dat altijd wenselijk? Nee, natuurlijk niet. Soms wil je een heel specifiek deel van een plot weergeven. Maar hoe je dat?

Matlab heeft daar natuurlijk een aantal functies voor verzonnen: xlim en ylim.

Als je eenmaal een plot hebt gemaakt, dan voeg je twee regels toe onderaan deze code:

xlim([-variabelex variabelex])

Deze regel limiteert de x-as en laat alleen een grafiek zien van –variabelex tot variabelex. Hetzelfde geldt voor de y-as:

ylim([-variabeley variabeley])

Natuurlijk kun je in plaats van een variabele ook direct een getal invullen.



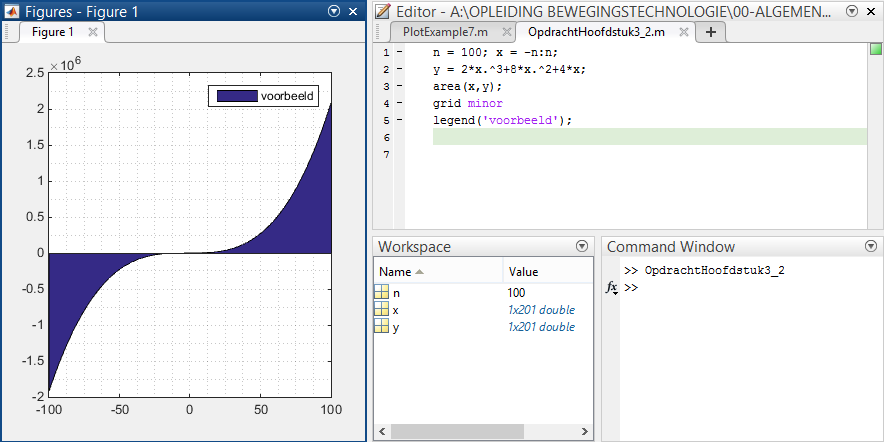
Figuur : twee dezelfde grafieken worden geplot, maar m.b.v. een nieuw plotbereik zien de grafieken er toch anders uit.

## Andere veelvoorkomende type plots

Er zijn nog meer verschillende type plots die in Matlab kunnen worden gemaakt. Het gaat echter voorbij de bedoelingen van dit document om ze allemaal te behandelen. Bovendien staat alles wat je wilt weten m.b.t. Matlab mogelijkheden online.

### Area plot

Soms wil je een plot tekenen waarbij het oppervlak onder een grafiek is weergegeven. Dit kan met behulp van de area():

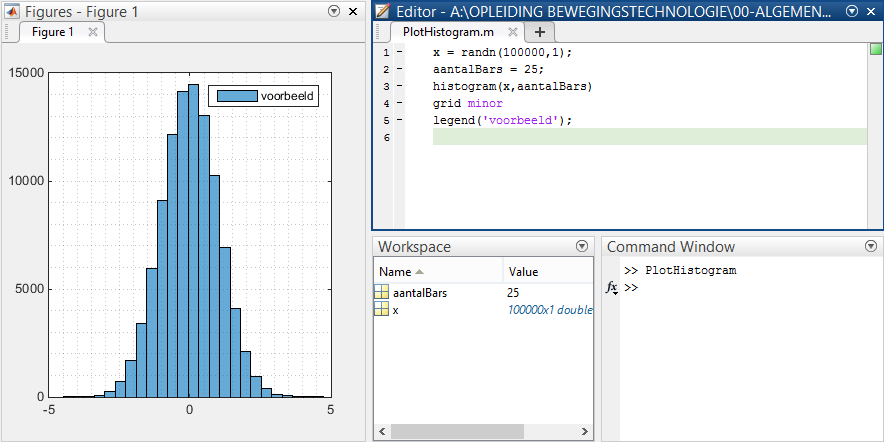


Figuur : Een voorbeeld van een area plot.

**Maak nu zelf een area plot van een willekeurige wiskundige functie**

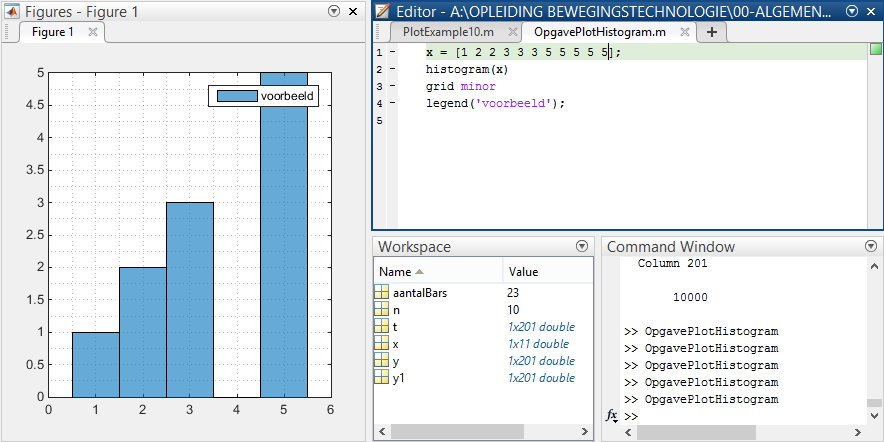
### Histogram

Histogrammen kunnen om allerlei redenen handig zijn. Zo kun je kansverdeling, frequentieverdeling weergeven of andere maten visualiseren. In het volgende voorbeeld wordt getoond hoe je een histogram kunt tekenen:



Figuur : Een voorbeeld van een histogram

In de histogram functie van Matlab kun je één vector met verschillende waardes weergeven in een histogram. Ook kun je nog opgeven hoeveel balken (bars) je wilt gebruiken voor de plot. Let op! Een balk (bar) van een histogram krijgt zijn lengte op basis van de hoeveelheid dezelfde waardes in een vector.

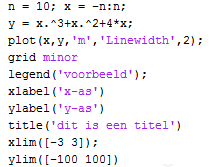


Figuur : het plotten van verschillende balken op basis van ingegeven input. Dit voorbeeld laat mooi zien dat het getal dat het meest voorkomt in de vector wordt gerepresenteerd door de langste balk.

## Vragen en Opdrachten

1. Maak een plot van de volgende vergelijking 2\*x^3+x^2+4\*x+4 en voeg de volgende verfraaiingen toe:
   1. Maak de lijn dikte 2
   2. Geef de magenta kleur aan de lijn
   3. Voeg een minor grid toe
   4. Voeg een legenda toe met daarin de volgende naam voor de lijn: voorbeeld
   5. Voeg een titel, x-as, y-as toe aan de grafiek.
2. Plot de grafiek van de voorgaande opdracht in het x-bereik van -3 tot 3 en in het y-bereik van -100 tot 100.

## Antwoorden

1. Dit is het resultaat van de eisen
2. De code moet worden aangevuld met twee regels. Hieronder staat het resultaat van de aanpassing plus de voorbeeld code:
3. **Wk3.1: Herhaling vorige week; terminologie, booleaanse logica, set path**
4. **Wk3.1: Tekenen in een plot(). Tekenen van een lijn of cirkel.**
5. **Wk3.1: Herhalen in matlab: for en while lussen**
6. **Wk3.1: Een opdracht waarin het voorgaande allemaal wordt gecombineerd.**
7. Wk3.2: Speciale aandacht naar matrices en vectoren (datatypes)
8. Wk3.2: Het gebruiken van een door ons aangeleverde functie (connecties met: set path en functies)

# Bronnen

* https://nl.mathworks.com/help/matlab/matlab\_prog/loop-control-statements.html?requestedDomain=www.mathworks.com#responsive\_offcanvas

1. Wk2.1: Het testen van de kennis van vorige week
2. Wk2.1: Wat is een functie in Matlab? Voorbeelden van veel gebruikte functies: smooth(), max(), min(), abs(), sin(), sind(), asin(), etc.
3. Wk2.1: Het maken van een moeilijkere functie in Matlab (met matrix output) (*apart onderdeel matrices en vectoren*)
4. Wk2.1: Matlab terminologie. Denk aan: colon, operator, indices, etc.
5. Wk2.1: Matlab help/doc functionaliteit
6. **Wk2.2: Booleaanse logica een hele lichte introductie (0,1, TRUE, FALSE, AND, OR) + voorbeelden en oefeningen.**
7. **Wk2.2: Het maken van beslissing: if-else (maak connectie met booleaanse logica)**
8. **Wk2.2: Het maken van een beslissing in een functies**
9. **Wk2.2: Set path. Wat is het? Waarom heb je het nodig? *(variabel, ergens anders, oefening waar bij het mis gaat + connectie workspace / map overzicht GUI)***
10. **Wk3.1: Herhaling vorige week; terminologie, booleaanse logica, set path**
11. **Wk3.1: Tekenen in een plot(). Tekenen van een lijn of cirkel.**
12. **Wk3.1: Herhalen in matlab: for en while lussen**
13. **Wk3.1: Een opdracht waarin het voorgaande allemaal wordt gecombineerd.**
14. Wk3.2: Speciale aandacht naar matrices en vectoren (datatypes)
15. Wk3.2: Het gebruiken van een door ons aangeleverde functie (connecties met: set path en functies)
16. Wk4.1: Toolboxes
17. Wk4.1: Andere Matab functies (denk aan: differentieren/integreren)
18. Wk4.1: Uitzoeken op basis van help/doc wat een bepaalde Matlab functie doet
19. Wk4.1: Error interpreteren (stack trace) herstel een programma
20. **Wk4.2: Het lezen en schrijven naar bestanden**
21. **Wk4.2: Werken met een .mat bestand. Wat is het?**
22. **Wk4.2: Een eindopdracht waarin al het voorgaande terugkomt, b.v.k. automatisch @random gegeneerd.**
23. **Wk4.2: Matlab etiquette.**