|  |  |
| --- | --- |
| Biostatica  Door Mark Schrauwen | Matlab Wk3.1 |

Inhoudsopgave

[Versiebeheer 2](#_Toc495594738)

[1 Introductie 3](#_Toc495594739)

[2 Herhalen 4](#_Toc495594740)

[2.1 For-lus 4](#_Toc495594741)

[2.1.1 Het optellen van de x opeenvolgende waardes van een vector 4](#_Toc495594742)

[2.1.2 Het opslaan van de waardes in een for-lus 7](#_Toc495594743)

[2.1.3 Waar zijn we ook al weer mee bezig? 9](#_Toc495594744)

[2.1.4 Uitwerking 11](#_Toc495594745)

[2.2 While-lus 12](#_Toc495594746)

[2.2.1 Afsluitende opmerkingen 13](#_Toc495594747)

[2.3 Vragen en opdrachten 14](#_Toc495594748)

[2.4 Antwoorden 15](#_Toc495594749)

[3 Tekenen in Matlab 17](#_Toc495594750)

[3.1 Een eerste aanzet 17](#_Toc495594751)

[3.2 Het verfraaien van een plot 19](#_Toc495594752)

[3.2.1 Kleuren van de lijnen 19](#_Toc495594753)

[3.3 Het afsluiten van een plot 21](#_Toc495594754)

[3.4 Meerdere plots in een plot 21](#_Toc495594755)

[3.5 Verschillende plots in een aparte GUI 22](#_Toc495594756)

[3.6 Het verfraaien van een plot 23](#_Toc495594757)

[3.7 Het opgeven van een plotbereik 24](#_Toc495594758)

[3.8 Andere type plots 24](#_Toc495594759)

[3.8.1 Area plot 24](#_Toc495594760)

[3.8.2 Histogram 25](#_Toc495594761)

[3.9 Vragen en Opdrachten 27](#_Toc495594762)

[3.10 Antwoorden 27](#_Toc495594763)

[4 Bronnen 29](#_Toc495594764)

# 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 | 02-10-2017 | Commentaar Bart verwerkt + andere kleine verbeteringen. | Mark Schrauwen |
| 0.3 | 07-10-17 | Aanvullingen gedaan. Bezig met het aanmaken van nieuwe opgaven. | 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 herhaalstructuren in Matlab. De logische en relationele operatoren van vorige week spelen hierbij een belangrijke rol, dus zorg ervoor dat je die onderdelen onder de knie hebt. 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 berekeningen voor 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.

**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 meestal ‘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 gereedschap is voor de Bewegingstechnoloog’.

Bij het herhalen van activiteiten in Matlab weet je bijna altijd hoe vaak je een actie moet herhalen.

Stel dat je elk element van een vector moet ophogen. De waarde waarmee één element 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 herhaald moet worden.

Voorstel voor bt relevant voorbeeld. Stel dat je de spierlengte van de m. vastus medialis kunt uitrekenen met behulp van de kniehoek. Alleen is het natuurlijk zo dat de kniehoek geen vast gegeven is: tijdens het bewegen verandert hij voortdurend. Dus de vraag luidt nu: hoe kan je voor alle mogelijke kniehoeken (0°, 0,1°, 0,2°,…160° knieflexie) de spierlengte van de m. vastus medialis bepalen. Dan moet je dus 1601 keer dezelfde berekening herhalen! Matlab kan dit soort ‘herhaaldingetjes’ heel snel en efficiënt uitvoeren.

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

Als bewegingstechnoloog krijg je nogal eens te maken met blessures aan pezen. Daarom is de sterkte van een pees belangrijk en interessant. Stel, je meet de treksterkte van een pees door hem in te klemmen in een machine. Een pees bestaat uit talloze bindweefselvezels. De machine trekt aan de pees totdat hij scheurt. Tijdens dit proces scheurt op een gegeven moment de eerste vezel door. Daardoor wordt de pees slapper en scheuren de volgende seconde de volgende twee vezels door. Daardoor wordt hij nog slapper en scheuren de volgende seconde de volgende drie vezels door, enzovoorts. De reeks van het aantal nieuw doorgescheurde vezels luidt: 1, 2, 3, 4 etc. De reeks van het totaal aantal gescheurde vezels is dan:  
0+1=1, 1+2=3, 3+3=6, 6+4=10, 10+5=15, 15+6=21 etc. Dus je neemt steeds de uitkomst van de vorige optelling plus het nieuw aantal gescheurde vezels. De logische vraag die je kunt stellen is: na hoeveel tijd c.q. na hoeveel van deze herhalingen is de pees volledig doorgescheurd als hij uit bijvoorbeeld 1000 vezels bestaat. Dit is typisch een voorbeeld dat je oplost in een herhaalstructuur in Matlab.

In dit voorbeeld moet een programma worden ontworpen dat 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.

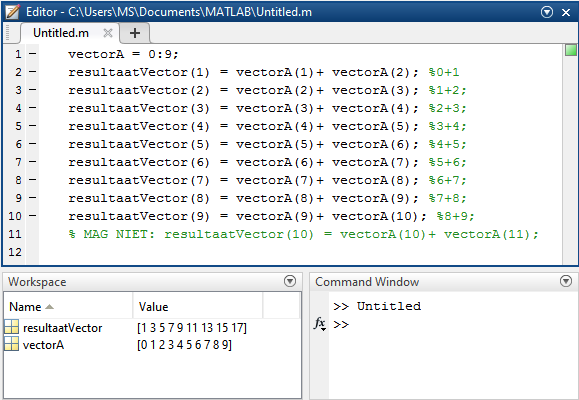
Stel dat we elke opeenvolgend getal willen optellen. Dan krijgen we: 0+1, 1+2, 2+3, … , 8+9. Merk op, dat de resultaat vector één element kleiner is dan de oorspronkelijke vector. Immers er volgt op 9 niet een nieuw getal. We tellen 9 niet opnieuw op bij een ander getal. Dat getal wordt overgeslagen 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 script dan weet je dat een for-lus kunt gaan gebruiken.

Maar wat is een for-lus?

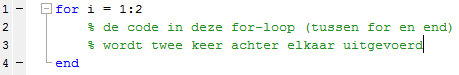
Een for-lus is een programmeerconstructie die de programmeur (jij) in staat stelt gemakkelijk een aantal herhalingen achter elkaar uit te voeren.

Hoe gebruik je een for-lus?



Figuur 1: 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 de lus eindigt met ‘end’. Het voorbeeld laat echter nog niet zien wat er daadwerkelijk gebeurd.

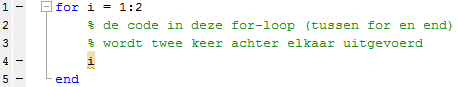


Figuur 2: dit is hoe een for-lus in Matlab eruit ziet.

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 niets in de for-lus. Met ‘in de for-lus’ wordt bedoeld: coderegels tussen de regel die het woord ‘for bevat en de regel die het woord ‘end’ bevat.

Voer het voorbeeld in Figuur 3 uit.

****

Figuur 3: het voorbeeld in Figuur 2 is behoorlijk uitgebreid.

Wat valt op?

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

Snap je waar deze waardes vandaan komen? Zo niet, wees niet bang en lees verder.

Als je in Matlab een regel met een variabele 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 1 en 2 naar respectievelijk 2 en 8 en voer de code uit.

Zie Figuur 4. 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.

Zet op regel 1 een breakpoint en voer de code opnieuw uit.



Figuur 4: 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 5: 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 is niet hetzelfde als wat de for-lus oplevert!

|  |
| --- |
| *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. De laatste keer heeft i de waarde 8. Tel daar 2 bij op en je krijgt, zoals ook te zien in het Command Window, de waarde 10.* |

### Het opslaan van de waardes in een for-lus

Vaak willen we bij het gebruik van een for-lus waardes opslaan in een variabele. Dat is bijna altijd een vector. Zoals je eerder hebt geleerd in deze cursus, bestaat een vector uit verschillende variabelen. Elke individuele variabele van een vector noemen we een element. De waarde van een element kunnen we veranderen door er een andere waarde naar toe te schrijven.

Hoe slaan we nu de waardes, aangemaakt in een for-lus, op?

Door gebruik te maken van een variabele en een index. Zie Figuur 6.



Figuur 6: het opslaan van de in een for-lus aangemaakte waardes.

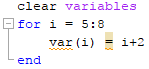
Voor de code in Figuur 6 zelf uit. Matlab zal een zogenaamde warning laten zien, maar deze kan je voorlopig negeren.

Je hebt nu alles wat je hebt aangemaakt in de for-lus opgeslagen in een vector genaamd var. Dit voorbeeld zegt niet zoveel en is puur bedoeld om te laten zien hoe je een for-lus kunt gebruiken. Meer betekenisvolle voorbeelden volgen later.

Wat gebeurt er als je in het voorgaande voorbeeld de waarde 1 verandert naar 5?

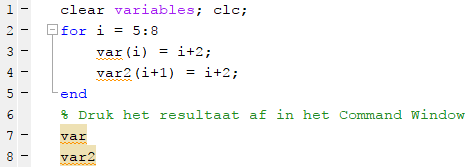
Test dit zelf!

Het resultaat van deze laatste actie is te zien in Figuur 7. Merk op dat we beginnen met de uitdrukking: clear variables. Deze uitdrukking maakt de Workspace leeg[[1]](#footnote-1). Zodoende weet je zeker dat je niet met de oude waardes van de variabele var aan het werken bent.



Figuur 7: we veranderen de waarde 1 naar 5.

Door i = 5:8 te schrijven, maakt Matlab zelf de niet gebruikte elementen van de vector aan. De elementen 1 tot en met 4 worden gevuld met nullen. Dat is iets om rekening mee te houden. Elk eerste element van een vector in Matlab begint met index 1. Als je dat negeert, door bij 5 te beginnen, maakt Matlab zelf de ontbrekende indices aan.



Figuur 8: Een waarde opslaan op een andere index.

**Neem de code in Figuur 8 over en voer deze uit. Zet een breakpoint op regel 3.**

De code in Figuur 8 laat zien dat je niet verplicht ben om een waarde op te slaan op plekje var(i). Je kunt een index ook ophogen of verlagen. Als je de resultaten van Figuur 7 vergelijkt met Figuur 8 zie je dat het resultaat van Figuur 8 langer is.

Snap je waarom?

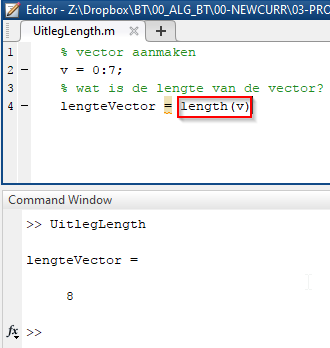
### Waar zijn we ook al weer mee bezig?

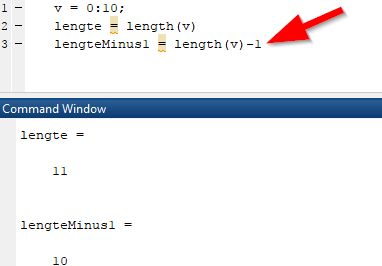
We willen het voorbeeld in Figuur 1 zo schrijven zodat het een stuk slimmer en korter is met behulp van een for-lus.

Hier volgen een aantal tips die je helpen om het gegeven voorbeeld slim op te pakken. De eerste handige standaard Matlab functie is *length()*. Deze functie kan de lengte van een vector (of matrix) opvragen[[2]](#footnote-2), dat wil zeggen: het aantal elementen in de vector opvragen.

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

Figuur 9: een voorbeeld van de functie length().

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

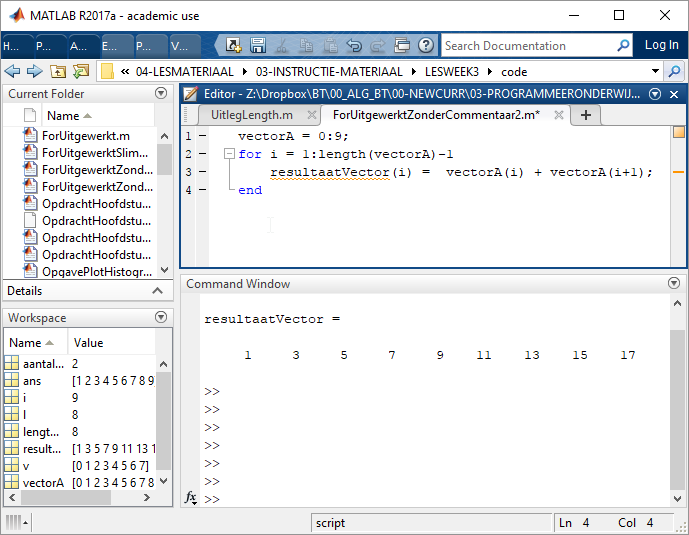


Figuur 10: een voorbeeld dat laat zien dat je ook direct een berekening kunt uitvoeren op de functie length(). In dit geval de berekening length() minus de waarde 1`.

**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 11 is te zien hoe het probleem in Figuur 1 veel slimmer en duidelijk is geschreven.



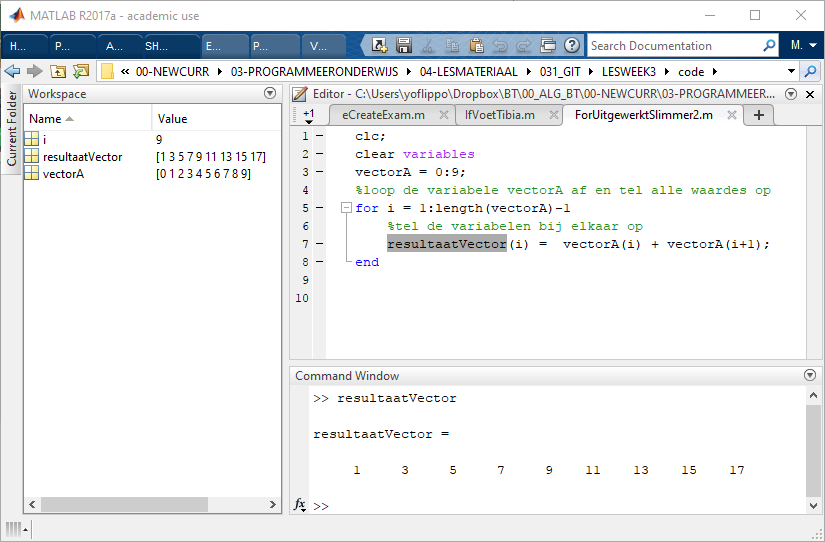
Figuur 11: het uitgewerkte voorbeeld, neem de code goed door. Je ziet dat deze code een stuk efficiënter is dan de code in Figuur 1.

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

Zet een breakpoint op regel 3

Voer het programma stap voor stap uit (druk telkens op F10)

Wat zijn de verschillen met de vorige versie?

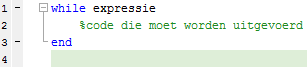
Merk op dat dit programma een heel stuk korter is dan het programma in Figuur 1. Dat komt doordat we nu gebruik maken van de for-lus. De code in Figuur 11 is niet heel erg duidelijk en dat komt door het gebrek aan commentaar. Daarom hebben we de code in Figuur 11 aangepast (zie Figuur 12).

Figuur 12: de code van Figuur 11 maar dan wat duidelijk a.d.h.v. commentaar

## While-lus

Naast de for-lus bestaat er ook de zogenaamde while-lus in Matlab. Een for-lus wordt gebruikt als je het aantal herhalingen **vantevoren weet**. Een while-lus wordt gebruikt als je het aantal herhalingen **niet vantevoren weet**.

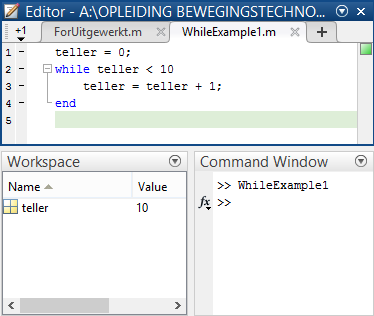
Je kunt probleemloos je hele leven programma’s ontwerpen zonder gebruik te maken van while-loops. Dat geldt voor bijna elke programmeertaal. Er zijn echter veel scenario’s denkbaar waarin de while-lus simpelweg handiger is. Dat zijn vaak situaties waarin een gebruiker om input moet worden gevraagd, of situaties waarin data moet worden ingelezen vanuit een apparaat. In figuur 13 zie je de algemene gedaante van de while loop.



Figuur 13: het ontwerp van een while-loop

Zoals eerder verteld, worden in Matlab, en in andere programmeertalen, herhalingen uitgevoerd zolang de logische-expressie *true* is. Dat werkt ook zo voor de while-loop. De while-lus blijft de code herhalen zolang (Eng: while) de expressie true is en stopt pas als de expressie false.

Merk op dat een for-lus geen expressie heeft. Een for-lus heeft een aantal herhalingen dat gelijk is aan de vector die je er aan toekent. Immers, 1:8 is een vector van 8 getallen.



Figuur 14: een nieuw voorbeeld waarin de while-lus wordt gebruikt.

Bekijk de code in Figuur 14.

Voer de code in en ga er met de debugger stap voor stap doorheen. Bekijk na iedere stap de waarde van de variabele teller.

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

Misschien had je 9 verwacht, dat is immers kleiner dan 10 zoals regel 2 vereist. De oplettende lezer ziet echter dat het antwoord op deze vraag in Figuur 14 is te zien, namelijk 10.

Debug het programma nogmaals en kijk wat er op het einde van de loop gebeurt met de variabele teller.

De while-lus wordt net zo lang uitgevoerd als dat de logische-expressie in dit geval: teller < 10 waar is.

Verwijder regel 3 en voer de loop weer uit met de debugger. Hoe vaak wordt hij nu uitgevoerd?

Indien de gebruiker vergeet de waarde teller op te hogen binnenin de while-lus, dan blijft het programma oneindig lang doorgaan. Niet vergeten dus!

### Afsluitende opmerkingen

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

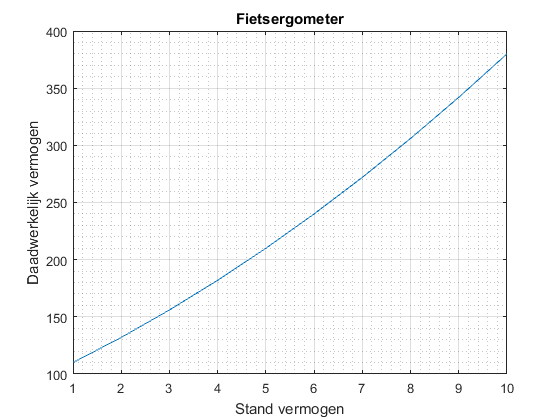
In de vragen/opdrachten gedeelte van dit hoofdstuk zal je enkele while-lus gerelateerde vragen tegenkomen en ook kun je dit type vragen verwachten op het tentamen.

## 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 Figuur 15.   
   [](http://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiGkMSOoonXAhUMKlAKHa8pCPYQjRwIBw&url=http://www.tunturi.com/nl/product/go-ligfiets-30/&psig=AOvVaw2sQtIQ41blmR97qEBRnz5C&ust=1508934652845752)

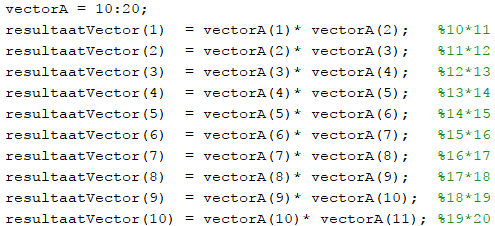
Figuur 15: een fietsergometer.

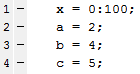
Dit gaat over een fietsergometer (hometrainer). De belasting van de ergometer staat uitgezet in een grafiek (Figuur 16).



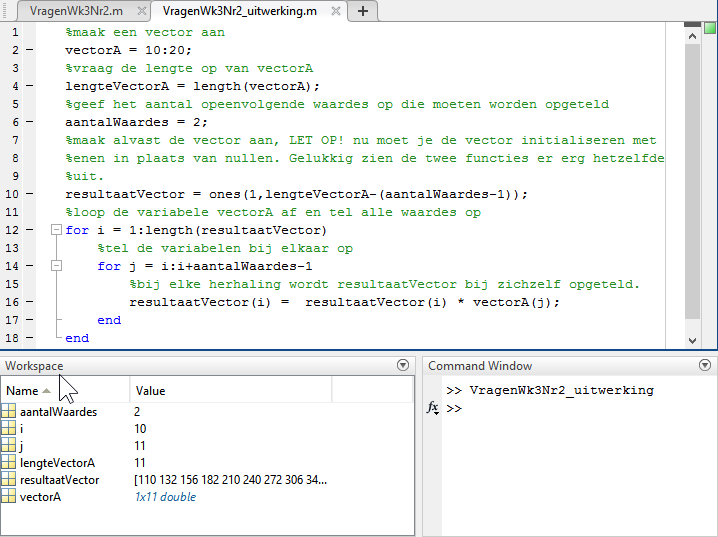
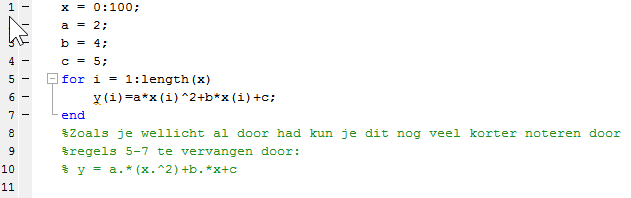
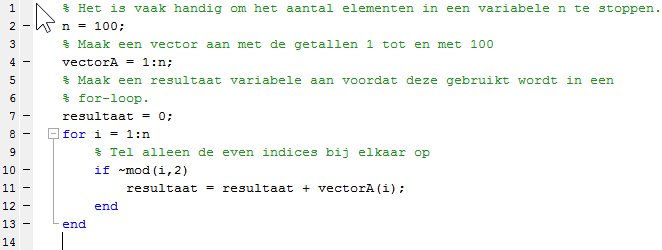
Figuur 16: vermogen en daadwerkelijk vermogen van een fietsergometer.

Op de x-as staat de stand van de knop waarmee de gebruiker de zwaarte (het vermogen) van de oefening instelt. Op de y-as staat het daadwerkelijk te trappen vermogen. De code om deze grafiek te maken staat in Figuur 16. De index van resultaatVector staat op de x-as en de inhoud van resultaatVector staat op de y-as. 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. Zorg dat het niet uitmaakt met welke waardes vectorA is geïnitialiseerd.



1. Zie de nevenstaande code: . Bereken met behulp van een for-lus y=a\*x2+b\*x+c;
2. 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(getal,2)[[3]](#footnote-3). De vector loopt van 1 tot en met 100.
3. Herhaal opdracht 5, maar gebruik in plaats van een for-lus een while-lus.

## 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.

# 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 (grafieken) kunt maken.

## Een eerste aanzet

Wanneer zou je in Matlab data moeten plotten?

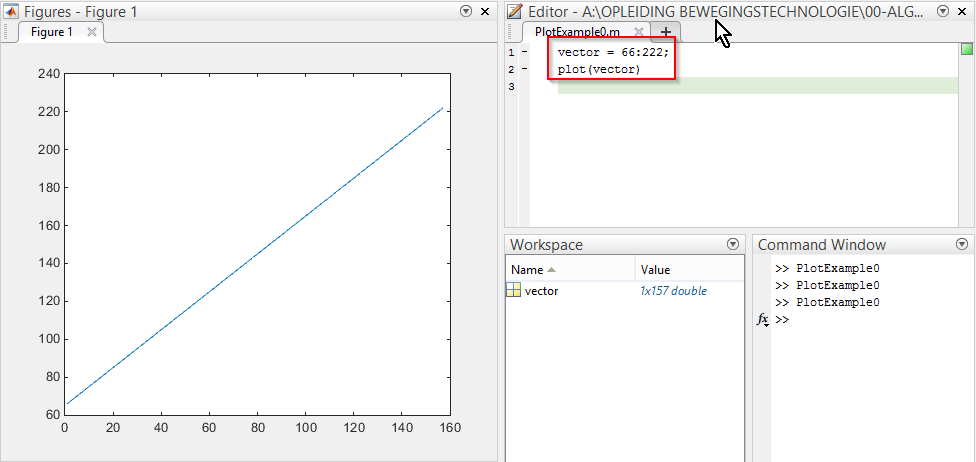
Het visualiseren van data is belangrijk Zo weet de Bewegingstechnoloog of er bijvoorbeeld tijdens een meting de waarde van een bepaalde sensor boven de 100 is geweest. Het visualiseren kan ook handig zijn om het gedrag van bepaalde functies of wiskundige vergelijkingen te onderzoeken. In de praktijk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

Je kunt met slechts een paar regels code in Matlab al snel een plot maken. Zie Figuur 15.

Probeer dit nu zelf in Matlab te doen.

Merk op dat in dit voorbeeld de y-waardes van de grafiek worden gespecificeerd.

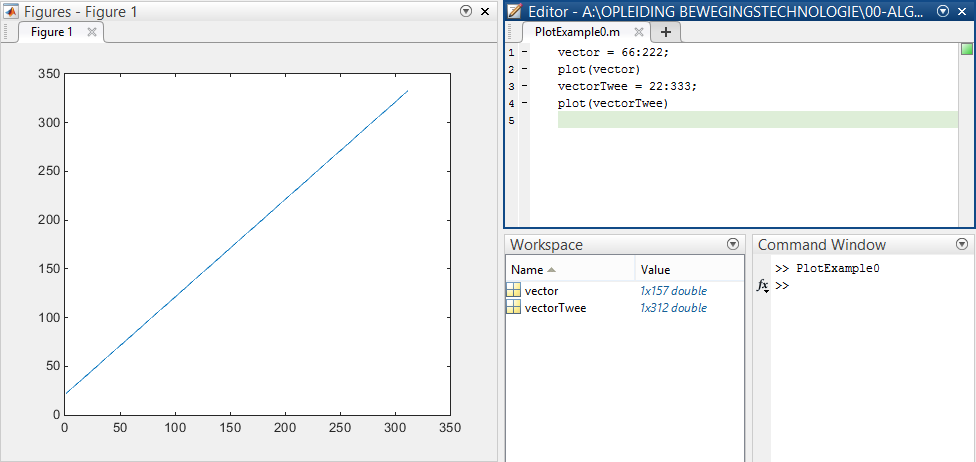
Waar denk je dat de x-waardes vandaan komen?



Figuur 17: 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 16.



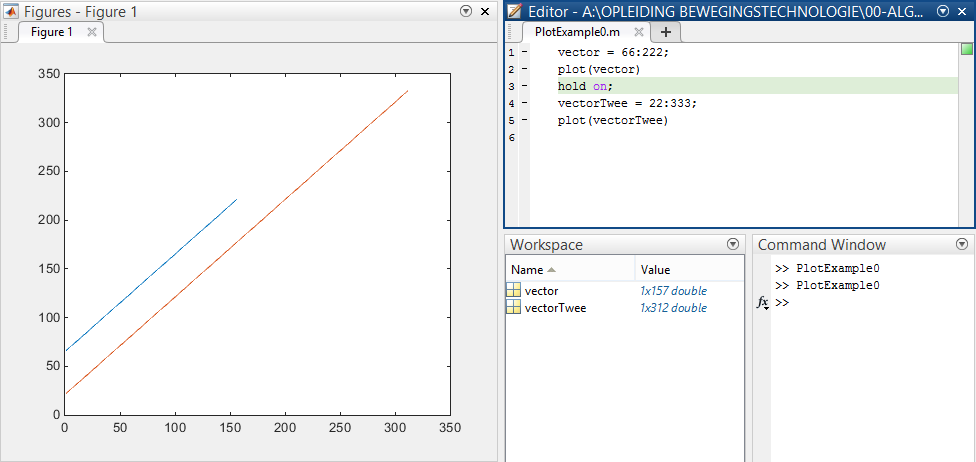
Figuur 18: De eerste poging om twee vormen te tekenen in één grafiek. Merk op dat de assen andere waardes hebben in vergelijking met Figuur 15.

|  |
| --- |
| *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 17:

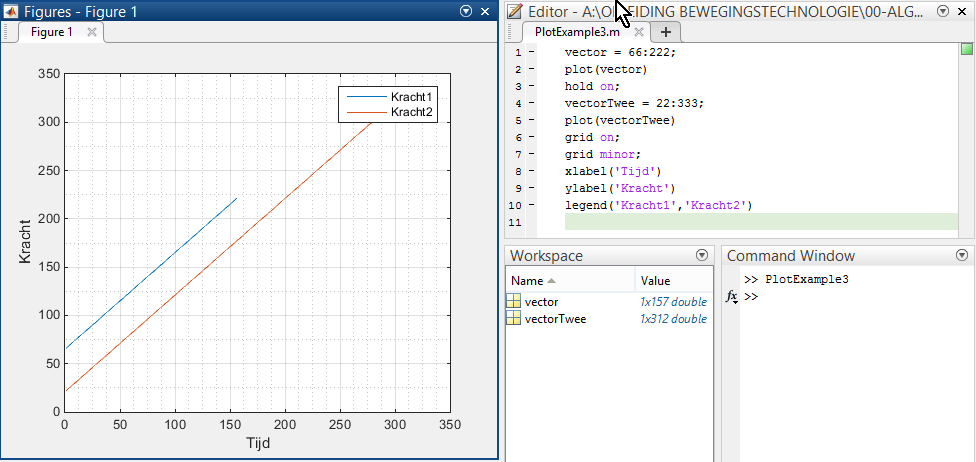


Figuur 19: Een voorbeeld van twee vectoren afbeeld in een plot dankzij de hold on; code.

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 in het Command Window 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 18).

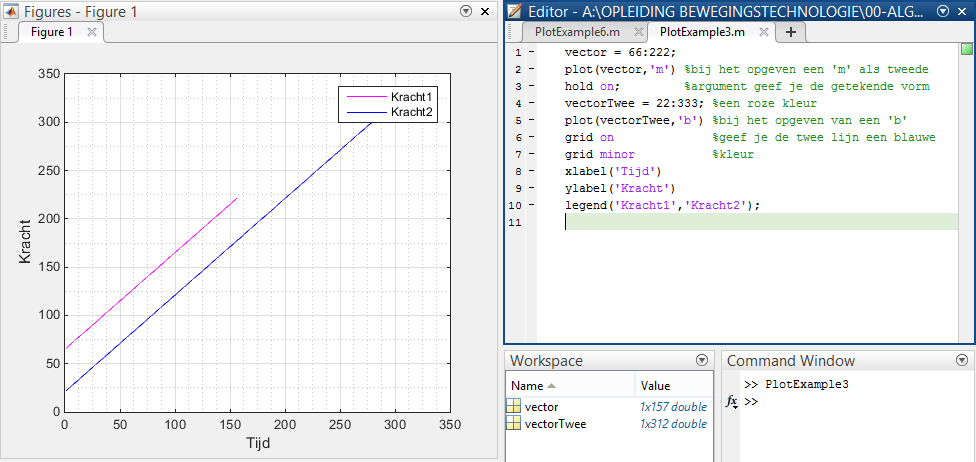


Figuur 20: De plot van Figuur 17 verfraaid en beter leesbaar gemaakt.

Met behulp van regel 6 en 7 in Figuur 18 voegen we een grid toe aan de plot. De grid maakt het aflezen van waardes van een grafiek gemakkelijk. Ook willen we graag de x-as en de y-as voorzien van een naam. Dat doen we in regel 8 en 9 in Figuur 18. Je ziet wederom dat Matlab al haar functionaliteit inpakt m.b.v. functies (xlabel en ylabel zijn functies zonder output). Als laatste kunnen we in een plot() ook nog opgeven welke signalen wat voorstellen. In regel 10 van Figuur 18 kunnen we een legenda maken m.b.v. de functie legend().

### Kleuren van de lijnen

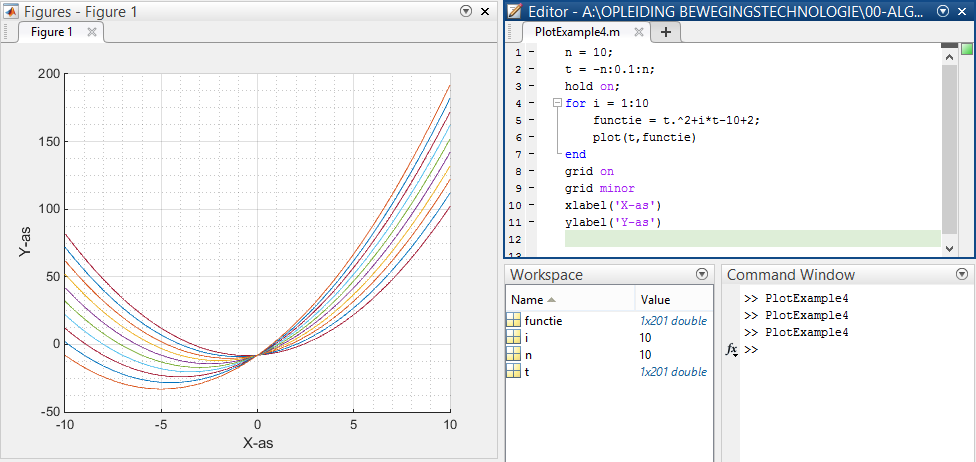
Je hebt inmiddels gemerkt dat Matlab zelf kleuren toekent aan de lijnen die worden getekend in een plot. Je hebt hier echter zelf ook wat over te zeggen (zie Figuur 19).



Figuur 21: je kunt ook zelf de kleuren in een plot definiëren

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

Zorg dat je de code in Figuur 20 begrijpt!

Verander hier en daar wat en probeer te voorspellen wat er zal gebeuren voordat je het programma runt ter controle.

## Het afsluiten van een plot

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

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

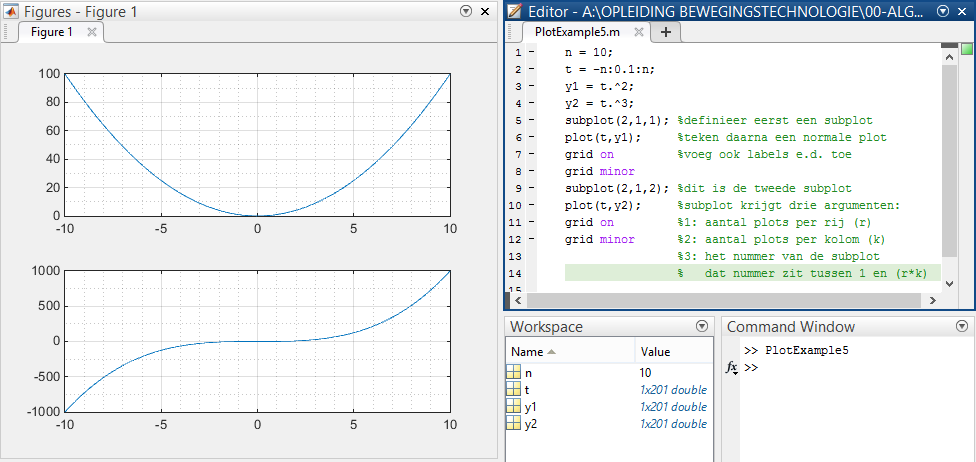
Een plot kan worden afgesloten met de code close all.

Maak op basis van voorgaande informatie zelf in Matlab een plot.

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

## Meerdere plots in een plot

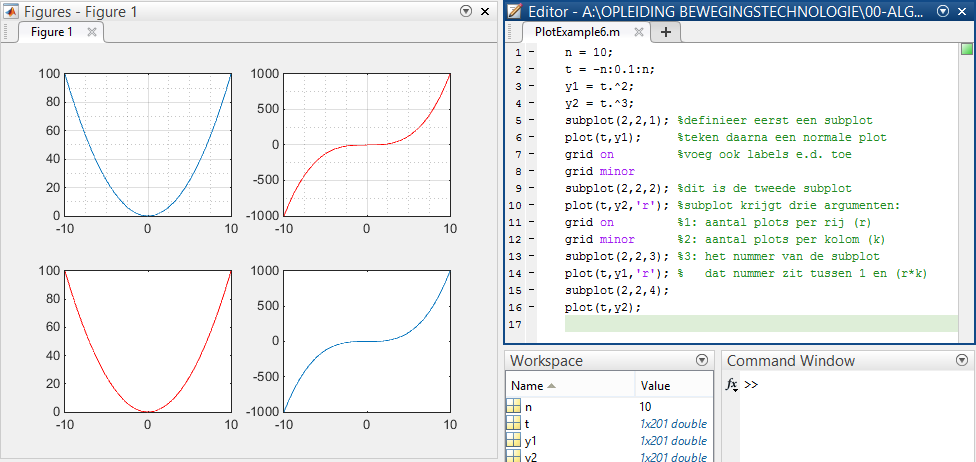
Tot nu heb je telkens één plot gemaakt in één *Graphical User Interface* (GUI). Het is ook mogelijk om meerdere plots in één GUI te plaatsen. Dan kan m.b.v. de functie subplot(r,k,n). Deze functie krijgt drie argumenten. Het eerste argument (r) wordt gebruikt om het aantal subplots in rijen weer te geven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



Figuur 23: een voorbeeld van subplots, meerdere plots in één GUI.

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

Het onderstaande figuur (Figuur 22) geeft antwoord op die vraag:



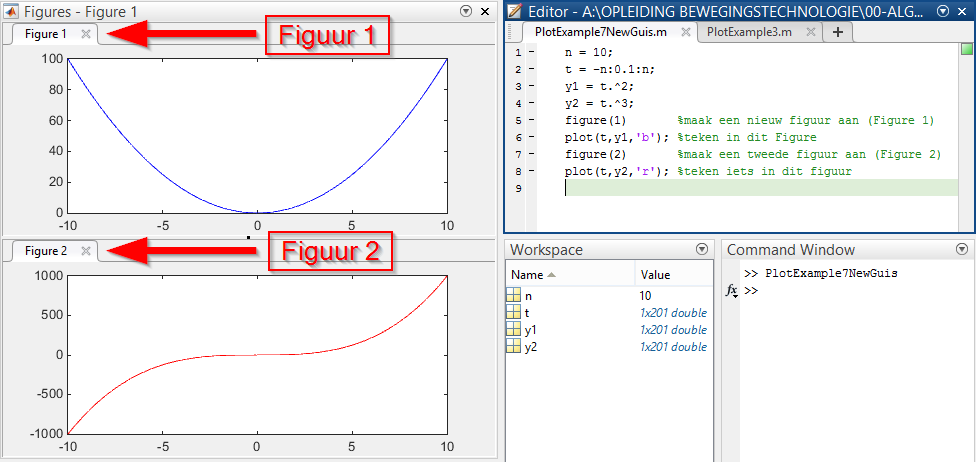
Figuur 24: meerder plots in één figuur.

|  |
| --- |
| *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:

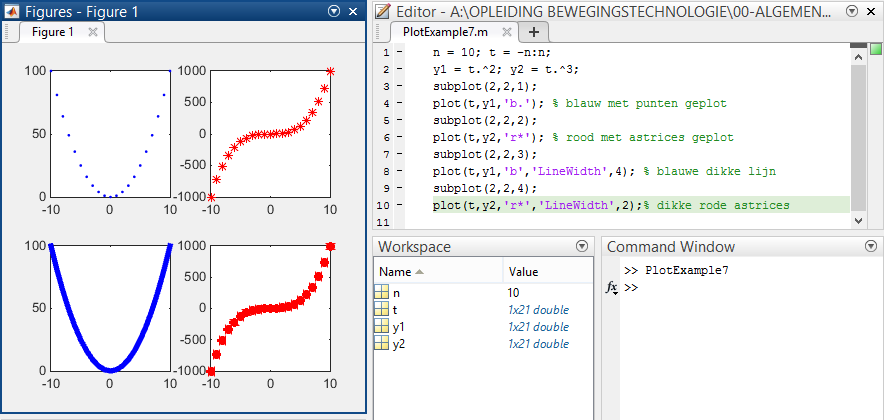


Figuur 25: in plaats van subplots is in dit voorbeeld gebruik gemaakt van twee verschillende GUI’s.

Het verschil tussen Figuur 21 en Figuur 23 is dat nu elke grafiek in een eigen window staat. Dat betekent dat één van deze windows kan worden gesloten zonder dat het andere window daar last van ondervindt.

## Het verfraaien van een plot

Inmiddels hebben we al een paar manieren geleerd om een plot te verfraaien. Het nut van zo’n verfraaiing is vaak om verschillen duidelijker te maken. Zo heb je eerder 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 26: 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 24 ziet, kun 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 24 en test dit zelf uit.

Merk op dat de plot functie een verschillend aantal inputs kan hebben. De volgorde van die inputs is overigens meestal wel belangrijk. Je hebt inmiddels opgemaakt uit Figuur 24 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 24 en test dit zelf uit.

## Het opgeven van een plotbereik

Het is je misschien niet opgevallen, maar tot nu toe werd, als je iets had geplot, de hele vector weergegeven. Alle data beschikbaar in een vector werd middels een plot zichtbaar gemaakt. Dat betekent dat Matlab zo goed als mogelijk de best passende weergave maakt van jouw data. 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 een aantal functies voor verzonnen: xlim en ylim.

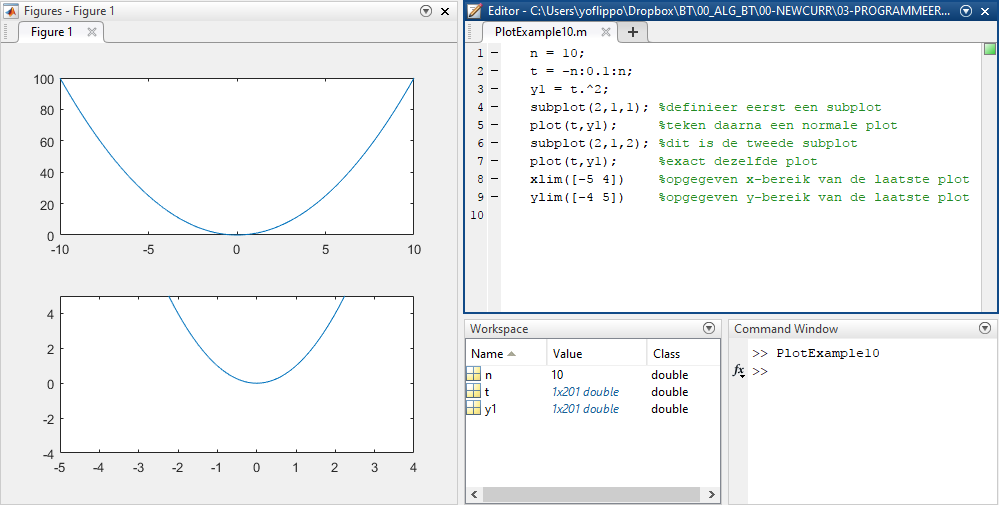
Als je eenmaal een plot hebt gemaakt, dan voeg je twee regels toe onderaan deze code:

xlim([-variabelex variabelex2])

Deze regel limiteert de waardes van de x-as en laat alleen een grafiek zien van –variabelex tot variabelex2. Hetzelfde geldt voor de y-as:

ylim([-variabeley variabeley2])

Natuurlijk kun je in plaats van een variabelex of variabeley ook direct een getal invullen, maar meestal duidt dat op een ongestructureerde manier van programmeren.



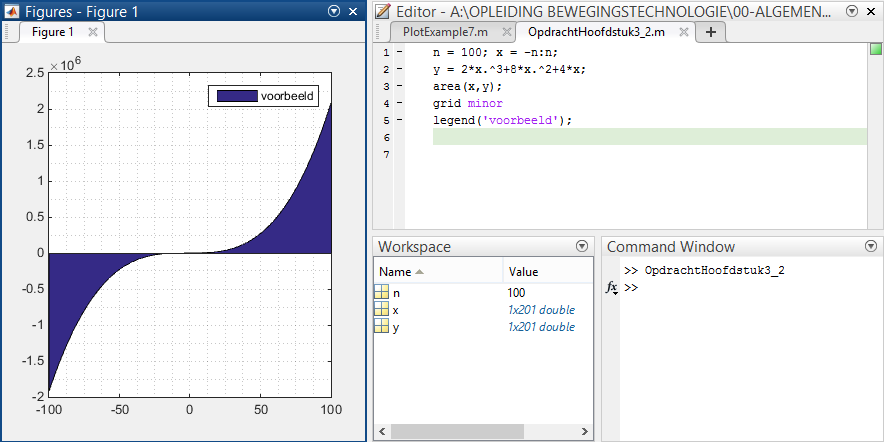
Figuur 27: twee dezelfde grafieken worden geplot, maar m.b.v. een nieuw plotbereik zien de grafieken er toch anders uit.

## Andere type plots

Er zijn nog meer verschillende type plots die in Matlab kunnen worden gemaakt. We gaan ze niet allemaal behandelen. Bovendien staat alles wat je wilt weten m.b.t. Matlab mogelijkheden online (https://nl.mathworks.com/help/matlab/). Hieronder volgen nog enkele veel voorkomende plots.

### Area plot

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

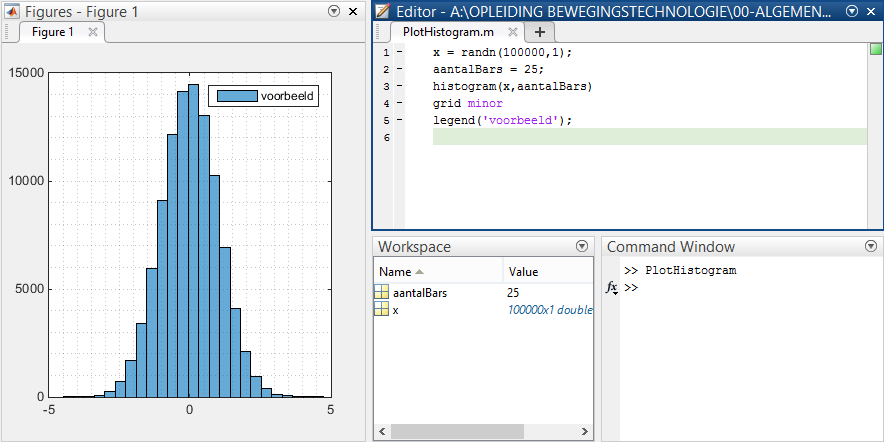


Figuur 28: Een voorbeeld van een area plot.

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

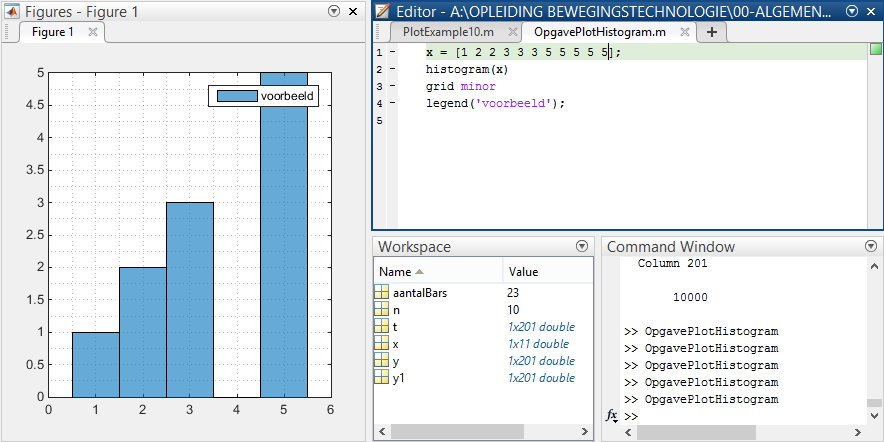
### Histogram

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 29: 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.

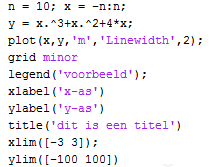


Figuur 30: het plotten van verschillende balken op basis van ingegeven input. Dit voorbeeld laat mooi zien dat het getal dat het meest voorkomt in de vector wordt gerepresenteerd door de langste balk.

## 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 en 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.
3. Welke kleuren kun je aan de lijnen in een plot geven?

## 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. De Matlab help-functie geeft je de benodigde informatie. Zo kun je de lijnen een gele of zwarte kleur geven.

# Bronnen

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

1. Pas op met een alternatieve vorm van clear variables namelijk clear all. Deze uitdrukking wordt door Matlab afgeraden te gebruiken omdat hij traag werkt en allerlei negatieve consequenties heeft. Je mag hem alleen gebruiken als je heel goed weet wat je aan het doen bent. [↑](#footnote-ref-1)
2. Zoals behandeld in de eerste reader. [↑](#footnote-ref-2)
3. Er wordt hier niet uitgelegd wat deze code exact betekent. Dat neemt niet weg dat je er goed aan doet om deze code even te testen. [↑](#footnote-ref-3)