|  |  |
| --- | --- |
| Biostatica  Door Alistair Vardy, Mark Schrauwen, Bart van Trigt en Herre Faber. | Matlab Wk3.2 |

Inhoudsopgave

[Versiebeheer 2](#_Toc497839386)

[1 Inleiding 3](#_Toc497839387)

[2 Staptijden bepalen m.b.v. voetstapschakelaars 4](#_Toc497839388)

[2.1 Het signaal 5](#_Toc497839389)

[2.1.1 Het doel 6](#_Toc497839390)

[2.2 Data opschonen 6](#_Toc497839391)

[2.2.1 Een deel van een vector selecteren m.b.v. een relationele operator 6](#_Toc497839392)

[2.2.2 Selecteren van waardes m.b.v. logical() 7](#_Toc497839393)

[2.2.3 De functie ones() en zeros() 8](#_Toc497839394)

[2.2.4 Het aanpassen van waardes van een vector m.b.v. een relationele operator 9](#_Toc497839395)

[2.2.5 De diff() functie 10](#_Toc497839396)

[2.2.6 Waar wordt diff() voor gebruikt? 11](#_Toc497839397)

[2.3 Vragen en opdrachten 12](#_Toc497839398)

[2.4 Antwoorden en uitwerkingen 15](#_Toc497839399)

[3 Toepassingen - Lichaamszwaartepunt 19](#_Toc497839400)

[3.1 Het plotten van de markers 19](#_Toc497839401)

[3.2 Totale zwaartepunt 21](#_Toc497839402)

[3.3 Tijd-as 23](#_Toc497839403)

[3.4 Hoogte van het zwaartepunt tegen de tijd 23](#_Toc497839404)

[4 Hoeken van segmenten 24](#_Toc497839405)

[4.1 Goniometrische functies in Matlab – graden vs. radialen 24](#_Toc497839406)

[4.2 Definitie van een gewrichtshoek 24](#_Toc497839407)

Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Beschrijving | Door |
| 0.0 | 11-07-2017 | Eerste versie. | Alistair Vardy |
| 0.1 | 19-09-2017 | Inleiding gemaakt. Diverse aanvullingen. Stuk over het selecteren van vectoren. | Mark Schrauwen |
| 0.2 | 21-09-2017 | Staptijden uitgebreid met opdrachten en uitwerkingen. | Mark Schrauwen |
| 0.3 | 27-09-2017 | Opmerkingen van studenten verwerkt. | Mark Schrauwen |
| 0.4 | 23-10-2017 | Aanvullingen. | Bart van Trigt |
| 0.5 | 09-11-2017 | Reader doorgelezen en verbeter. Hoofdstuk 3 verbeterd en aangepast. | Herre Faber |
| 0.6 | 09-11-2017 | Een opdracht die de student moet maken m.b.t. lichaamszwaartepunt verbeterd. | Mark Schrauwen |
| 0.7 | 20-11-2017 | Opmerkingen van E. van de Kerkhof verwerkt. | Mark Schrauwen |
| 0.8 | 29-11-2017 | H2 gemaakt. | Mark Schrauwen |

# Inleiding

Het soepel kunnen werken met indices, matrices en vectoren is erg belangrijk.

Als je zoiets leest, moet je je altijd afvragen: ‘waarom is dat *erg* belangrijk’

Een antwoord is: omdat wij bij Bewegingstechnologie vaak met data werken, data afkomstig van sensoren en meetsystemen. Wij willen meten aan de mens! Data staat zelden in een keer in het juiste formaat. Data is zelden in een keer geschikt om een conclusie uit te trekken of om te interpreteren. Data moet worden bewerkt en geanalyseerd.

Matlab is een handig programma omdat het verwerken van data met Matlab ongekend eenvoudig gaat. De auteur[[1]](#footnote-1) durft te beweren dat er geen programma bestaat dat zo goed in staat is data te verwerken als Matlab.

Data ingelezen m.b.v. Matlab komt vaak in een vector of in een matrix te staan. Zoals je eerder hebt geleerd, is een vector een variabele met verschillende elementen. In elk element van een vector kan een getal staan. Een matrix is een vector bestaande uit vectoren. Matlab komt met allerlei faciliteiten om matrices, en dus ook vectoren, gemakkelijk te verwerken. Concluderend, Matlab is een geschikt programma om data te verwerken. Het verwerken kan je met scripts automatiseren. Als je van 20 proefpersonen 5 verschillende metingen hebt gedaan, waarbij elke meting 3 x is herhaald, moet je 300 databestanden doorwerken. Dat zou handmatig heel veel tijd kosten. Met Matlab heb je ineen dag tijd een script geschreven dat zoiets helemaal automatisch voor je doet.

Waarom gebruiken Bewegingstechnologen Matlab? Uiteindelijk… om tijd te besparen. Matlab is niet het doel, maar het hulpmiddel. Matlab dwingt je echter wel om logisch na te denken en dat is natuurlijk nooit verkeerd.

In dit hoofdstuk behandelen we niet veel nieuwe informatie. We gaan wat dieper in op vectoren en technieken in de context van BT-gerelateerde vraagstukken. We gaan ons nog niet specifiek bezighouden met het inlezen van data. Dat komt volgende week. In dit document houden we ons voornamelijk bezig met het verwerken van data.

**Zie je een fout? Of heb je een aanbeveling, dan passen we dat z.s.m. mogelijk aan klik daarvoor op** [**deze weblink.**](http://www.bewegingstechnologie.com/weblinks/curr17/Biostatica/bitbucket)

**Stuur eventueel een e-mail naar** [**mjschrau@hhs.nl**](mailto:mjschrau@hhs.nl)

# Een klein beetje herhaling

We hebben op dit punt de belangrijkste onderdelen van Matlab behandeld. Er is nog veel meer dat we zouden kunnen behandelen. Slecht een fractie van wat we zouden kunnen vertellen komt aan bod in jouw opleiding. Dat niet erg. Alles wat we je hadden kunnen vertellen, is (globaal) opgebouwd uit: functies, beslissingen en herhalingen. Kortom, de basiselementen ken je nu. De fundering is gelegd.

We kregen een hele goede vraag van één van jullie studiegenoten:

Mag je een if-statement ook combineren met een for-lus?

Dat klinkt als een triviale vraag, maar is dat zeker niet. Als je al kunt programmeren denk je bij jezelf “ja natuurlijk”! Zo dachten wij dat ook tijdens het ontwikkelen van dit cursus-materiaal. Maar wij weten dat doordat iemand dat ons heeft verteld of omdat we daar proefondervindelijk zijn achter gekomen (ervaring). Het is echter niet vanzelfsprekend als je begint met programmeren. Daarom is de gestelde vraag een hele goede vraag.

Je mag alles wat we tot nu hebben behandeld combineren met elkaar. Zo mag je een if-statement combineren met for-lussen die stoppen in switch-cases waar je weer een functie van maakt die je weer in een if-else-statement stopt waar je weer een functie van maakt, enzovoort. Je mag alles combineren met elkaar. Dat maakt dat programmeren (en het ontwikkelen van software) moeilijk is. De basiselementen zijn goed te begrijpen, maar de hoeveelheid combinaties van die basiselementen zijn oneindig (en dat maakt programmeren moeilijk).

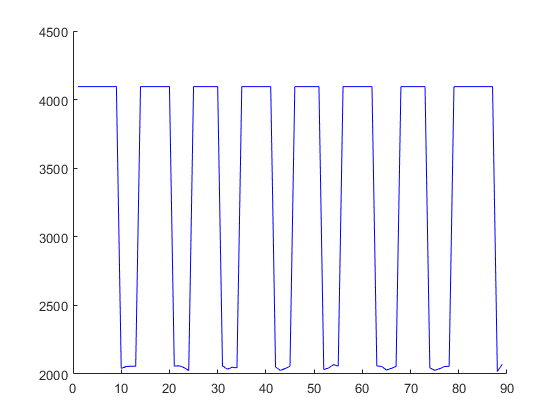
# Staptijden bepalen m.b.v. voetstapschakelaars

Een stap kunnen we op verschillende manieren registreren. Je telefoon of een horloge doet dit met behulp van een accelerometer. Je kunt ook videobeelden maken en daaruit de contactmomenten bepalen. De methode die we hier gebruiken zijn voetstapschakelaars. Deze sensoren maken gebruik van de druk tussen de voet en de grond. Als er voldoende druk wordt gedetecteerd, wordt het signaal ‘hoog’. Als er geen of te weinig druk op de voetstapschakelaar staat, wordt het signaal ‘laag’ (zie Figuur 1). Dit signaal wordt vervolgens opgeslagen in een data-bestand (vaak een tekst-bestand[[2]](#footnote-2)).

## Het signaal

Hieronder zie je een voorbeeld van zo’n voetstapsignaal van één voet.

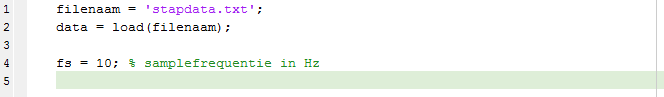
Zie jij aan de data in Figuur 1 wanneer de proefpersoon een stap zet of los van de grond komt? Let op dat waardes boven de 4000 niet automatisch hoeven te betekenen dat er dan voetcontact is. Het kan net zo goed andersom zijn. Toch kan je, door naar de grafiek te kijken afleiden hoe het zit…



Figuur : Voetstap data.

[Download het file stapdata.txt](http://www.bewegingstechnologie.com/weblinks/curr17/voorbeelddata/stapdata.txt) en zet de file in een nieuwe folder (met een duidelijke naam). Onthoud de locatie van de folder.

Ga in Matlab m.b.v. de Current Folder naar de zojuist aangemaakte folder.

Zet de volgende commando’s in een script:

Figuur : het inlezen van voetstapdata m.b.v. Matlab.

In Figuur 2 staat code om de data die je zojuist hebt gedownload in te lezen. Wat hier exact gebeurt, is nu nog niet van belang. In lesweek 4 ga je dieper in op het inlezen van data m.b.v. load(). Het enige dat van belang is, is dat de variabele data de gegevens uit het tekstfile bevat.

### Het doel

Het doel van dit onderdeel is het aanpassen van het signaal, zodat we alleen nog nullen (voet los) en enen (voet aan de grond) hebben. Op dit moment varieert het signaal in Figuur 1 ergens tussen 4100 en 2000.

## Data opschonen

De data in Figuur 1 schommelt een beetje. Hoewel wij mensen in staat zijn af te leiden uit de grafiek wat er aan de hand is, is dat niet wat we willen. Wij willen een script dat voor ons kan bepalen wanneer een proefpersoon een stap heeft gezet en dat vertaalt naar een 1 (voetcontact) of een 0 (geen voetcontact).

Zie jij waar het signaal schommelt?

Je kunt deze uitdaging op verschillende manier aanpakken. Je kunt m.b.v. een for-loop en een if-else-construct code schrijven die voor elk element van de data bekijkt of het een stap is of niet.

Je kunt in Matlab ook heel handig een deel van een vector selecteren op basis van een relationele operator. Hoe dit handig is voor de bovenstaande opdracht zie je hieronder.

### Een deel van een vector selecteren m.b.v. een relationele operator

In deze subparagraaf gaan we kijken hoe we een deel van een vector kunnen selecteren en tegelijkertijd de geselecteerde waardes kunnen aanpassen. Zodoende, kun je heel snel het eerder beschreven doel bereiken. Hieronder volgt de uitleg m.b.v. een voorbeeld dat niet gerelateerd is aan voetstapdata. Jij moet dus zelf de volgende uitleg begrijpen en toepassen op de voetstapdata.

Maak een nieuw script aan.

Typ de onderstaande code over:

%% een vector genaamd 'signal'

t = 0:0.01:pi;

signal = sin(2\*pi\*t);

Zo ziet de sinus er uit in een grafiek (zie Figuur 3):

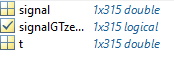


Figuur : de sinus in een grafiek weergegeven.

Test wat er gebeurt als je onderstaande code met een relationele operator uitvoert:

signalGTzero = signal > 0

**De workspace ziet er nu zo uit:**

****

Figuur : de workspace bij het uitvoeren van voorgaande code.

Merk op dat de variabele signalGTzero van het type ‘ logical ‘ is.

Merk op dat de relationele operator tot gevolg heeft dat een bewerking op een vector een logische vector signalGTzero teruggeeft. Dat is een vector die bestaat uit logische enen en nullen. Deze bevat evenveel elementen als de vector signal. Overal waar is voldaan aan signal > 0 wordt een logische 1 gezet in signalGTzero. Overal waar signal <= 0 wordt een logische 0 gezet in signalGTzero. Terzijde: het verschil met het type ‘double’ is dat een logische vector **alleen** enen en nullen mag bevatten. Een vector met het type ‘double’ mag ook andere numerieke waardes bevatten. Matlab maakt dus verschil tussen een logische 0 en een numerieke nul en tussen een logische 1 en een numerieke 1.

### Selecteren van waardes m.b.v. logical()

Dan nu een andere extra stap. Later gaan we de informatie in deze paragraaf samenvoegen met paragraaf 3.2.1.

Je kunt een deel van een vector selecteren op de volgende, wellicht niet intuïtieve, manier:

stukData = signal(**logical**([ 1 1 1 1 1 1 1 ]))

In dit stukje code worden de eerste zeven elementen van de vector signal in stukData gezet.

Je ziet een rij met zeven 1-en. Dat zijn numerieke enen. Met behulp van de logical() functie maken we van die vector logische enen. Die logische 1-en worden door Matlab als index opgevat, omdat ze tussen haakjes achter de variabele signal staan. Matlab interpreteert die zeven logische enen nu als allemaal afzonderlijke kleine opdrachtjes: staat er een logische 1 op de eerste plek? Neem dan de waarde en zet hem in stukData. Staat er een logische nul? Doe er dan niets mee. Je ziet in het voorgaande code voorbeeld dat je op deze manier de eerste zeven elementen van de signal vector selecteert.

Het volgende stukje code laat zien dat alle elementen met een nul, niet worden afgedrukt.

stukData2 = signal(logical([1 0 1 0 1 0 1]))

Onderzoek het verschil tussen de laatste twee code voorbeelden.

Als het goed is, is je opgevallen dat het toevoegen van nullen ervoor zorgt dat waardes niet worden afgedrukt in het Command Window. Anders gezegd, als in het voorgaande code voorbeeld alle enen in de vector tussen blokhaken nullen zouden zijn, dan wordt er niets afgedrukt.

Probeer dit!

### De functie ones() en zeros()

We kunnen ook alle elementen van de vector signal weergeven op de volgende manier:

stukData3 = signal(logical(**ones**(1,length(signal))))

Natuurlijk heeft dit niet zoveel zin. Want je kunt ook alle elementen van een vector opvragen door het typen van de naam van de vector: signal. In het voorgaande voorbeeld genereert de functie ones() voor ons een hele hoop enen. Dat is een standaard Matlab functie.

Test zelf m.b.v. Matlab de functie ones() en beantwoorde volgende vragen:

Hoe genereer je een kolom met enen?

Hoe genereer je een rij met enen?

We kunnen ook een vector met alleen maar nullen genereren. Dat doen we m.b.v. de vector zeros(). In het voorgaande voorbeeld worden er net zoveel enen aangemaakt als het aantal elementen in de vector signal (de lengte van de vector signal).

Wat wordt er afgedrukt als je in plaats van enen alleen nullen gebruikt? Zie in het volgende voorbeeld:

signal(logical(**zeros**(1,length(signal))))

Nu genereert Matlab een lege vector. Ok, waar was dit allemaal voor bedoeld? Je hebt gezien hoe je met relationele operatoren elk element van een vector kunt testen. Bijvoorbeeld: heeft elk element van de vector een waarde groter dan x? Zie de vorige paragraaf. Je hebt in deze paragraaf gezien hoe je een deel van een vector kunt selecteren m.b.v. logical waardes.

In de volgende paragraaf komt alles samen.

### Het aanpassen van waardes van een vector m.b.v. een relationele operator

Stel we willen alleen waardes van een vector gebruiken die positief zijn. Dan kunnen we de relationele operator gebruiken in combinatie met het toekennen van een gewenste variabele. Dat is cryptisch beschreven, daarom krijg je hieronder direct te zien hoe je dat doet:

signal(signal<0) = 0;

Merk op dat de variabele signal het signaal van een sinus bevatte. Het resultaat van deze actie is te zien in de onderstaande plot:



Figuur : het resultaat van signal(signal<0)=0;

Hopelijk zie je hoe de vorige twee paragrafen in het voorgaande codevoorbeeld samen komen. Je selecteert eerst de (on)gewenste waardes van de vector signal en aan die waardes ken je een gewenste waarde toe. De relationele operator genereert logische enen en nullen en zodoende selecteer je delen van een vector. Aan die selecties ken je vervolgens een waarde toe (in het voorgaande voorbeeld ken je een nul toe).

Wat gebeurt er met alle andere waardes?

Alle andere waardes blijven intact. Anders zou Figuur 5 geen sinusvormen meer bevatten. Kortom daar doet de voorgaande code niets mee. In dit voorbeeld krijgen de ongewenste elementen de waarde nul. Uiteindelijk hebben we zodoende alle negatieve waarden van de vector verwijderd.

### De diff() functie

Zoals al vaker aangegeven is Matlab voorzien van veel functionaliteit. In deze subparagraaf behandelen we een nieuw stukje Matlab functionaliteit. De diff()functie.

Typ in het Command Window: doc diff.

Lees de documentatie van de diff functie.

Zoals vaker is de uitleg niet heel erg gemakkelijk te begrijpen… Daar is iets beters voor:

Lees de code voorbeelden in de documentatie

Hoewel de standaard documentatie van Matlab vaak heel erg goed is, zeker als je er aan gewend bent, zullen we toch een paar eenvoudige voorbeelden van de diff functie geven.

Lees de onderstaande code:

clearvars

close all

clc

% Het aanmaken van een vector met lineair toenemende waardes

stapgrootte = 0.01

linVector = 0:stapgrootte:stapgrootte\*20;

% Toepassing van de diff() functie

diffVector = diff(linVector)

Maak een nieuw script en voer de bovenstaande code uit.

Pas de waarde van de variabele stapgrootte aan (maak er bijvoorbeeld 0.37 van) en voer de code opnieuw uit.

Door het zelf testen van de functie diff() heb je, als het goed is, gezien dat de functie diff() het verschil tussen elk opeenvolgend element geeft. Of zoals Matlab het zegt:

[Y](file:///C:\Program%20Files\MATLAB\R2017a\help\matlab\ref\diff.html?overload=%28matlab%29%2Fdiff+false#outputarg_Y) = diff([X](file:///C:\Program%20Files\MATLAB\R2017a\help\matlab\ref\diff.html?overload=%28matlab%29%2Fdiff+false#inputarg_X)) calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1.

Alhoewel je het natuurlijk eigenlijk zelf moet opzoeken in Google, volgen hier enkele vertalingen (voor deze ene keer). Adjacent = opvolgend, array = rij/kolom. Along the first array dimension whose size does not equal 1, betekent: langs de eerste dimensie die meer dan 1 element bevat. In dit voorbeeld bestaat linVector uit 1 rij en 21 kolommen. De verschillen tussen de elementen in horizontale richting (in de richting van de rij) worden dus gegeven.

Wat is de lengte van de variabele diffVector in vergelijking met de lengte van linVector? Bedenk een verklaring.

Zoals je in de Workspace kunt zien, is de lengte van de variabele van diffVector 1 waarde kleiner dan de lengte van linVector. Dat komt doordat als je het verschil berekent tussen elke opeenvolgende waarde je alleen verschillen (differences) teruggkrijgt:

Vector: 0 1 2 3 % n elementen

Verschil tussen elk element: 1 1 1 % n-1 elementen

Test dit zelf met de volgende code, copy en paste het in het Command Window:

vec = 0:2; lengteDiff = length(diff(vec)), lengteVector = length(vec)

### Waar wordt diff() voor gebruikt?

De functie diff() kan handig zijn als je *overgangen* wilt bepalen. Wat zijn overgangen? Bijvoorbeeld de overgang van een moment dat iemand op een voetstapschakelaar staat naar dat iemand er niet meer op staat. Zo’n snelle verandering wordt ook wel een overgang (zie Figuur 6en Figuur 7) genoemd.

Bij rugby mag je met de ‘bal’ in de hand naar voren rennen. Maar als je hem naar een medespeler gooit, moet je naar de zijkant of naar achteren gooien, in ieder geval niet naar voren, want dat is tegen de regels. Stel, er zit een sensor in de bal, die iedere 0,01 seconde de positie van de bal in de lengterichting van het veld meet en je hebt een ander detectiemechanisme dat ook iedere 0,01 seconde bepaalt of een speler de bal vast heeft of niet. Om het ingewikkeld te maken: een speler mag wel de bal naar voren schoppen, maar dat aspect negeren we nu. Dit levert voor 1 van de 2 teams (we noemen het team A) twee vectoren op:

1. balVast: een logical rijvector. True: een speler heeft de bal vast, False: Geen enkele speler heeft de bal vast, hij bevindt zich dus in een vlucht omdat hij net is gegooid.
2. Pos: positie van de bal in de lengterichting van het veld. Het eigen doel van team A heeft positie nul, het doel van het andere team heeft positie +100.

Schrijf de code om te bepalen of team A zich aan de regels houdt. In dit voorbeeld weten we zeker dat team B er niet aan te pas is gekomen. Gegeven:

balVast = logical([1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1]);

Pos = [30 31 33 35 38 36 34 32 30 28 26 24 22 22 23 24];

Als het werkt verander dan 26 in 29. In dag geval met er een overtreding worden gemeld. Het antwoord staat hieronder. Gebruik dat alleen ter controle of als je er echt niet uitkomt.

Antwoord:

balVast = logical([1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1]);

Pos = [30 31 33 35 38 36 34 32 30 28 26 24 22 22 23 24];

relevantPos = Pos(not(balVast));

naarVoren = diff(relevantPos);

[x,i] = find(naarVoren > 0);

if isempty(i) == true

disp('Geen overtreding')

else

disp('Overtreding!')

end

## Vragen en opdrachten

|  |  |
| --- | --- |
|  | Pak de voetstapdata er bij. Gebruik de relationele operator om van elke meetwaarde die een ingedrukte voetstapschakelaar representeert 1 te maken. **Elke meetwaarde die het niet indrukken van de voetstapschakelaars representeert wordt een nul.**  Ter info. Als je de voetstapdata hebt ingelezen heb je een tabel (matrix) met drie kolommen. De eerste kolom bevat de tijd uitgedrukt in seconde. De tweede kolom bevat voetstapdata van het linkerbeen en de derde kolom van het rechterbeen. |
|  | Gebruik opnieuw de oorspronkelijke voetstap data en maak een nieuw script. **Verwerk nu m.b.v. één of meerdere for-loops en één of meerdere if-statements de voetstapdata om deze op te schonen zoals in de vorige opdracht.**  Ter info. Je mag in deze opdracht geen relationele operatoren gebruiken die direct in een vector worden toegepast. |
|  | Welke van de twee voorgaande methoden levert de meest leesbare code op? |
|  | Welke van de twee voorgaande methoden heeft jouw voorkeur? |
|  | **Breid de analyse van de voetstapdata verder uit** door het schrijven van Matlab code**, zodat je de heel-strike en toe-off weergeeft zoals in de volgende figuren:**    Figuur : de rechter voetstapdata met toe-off en heel-strike.    Figuur : de linker voetstapdata met toe-off en heel-strike. |
|  | Iemand zwaait met zijn ene arm heen en weer. De schouderhoek alfa (in graden) wordt beschreven met de formule: . Zijn andere arm zwaait heen en weer met: . De tijd t loopt van 0 tot 2 seconden met stapjes van 0,01 seconde. **Maak een nieuw script en vul de variabelen alfa en beta op de beschreven manier.** |
|  | **Toon in een grafiek alfa (in rood) en beta (in blauw) tegen de tijd. Maak de grafiek netjes met labels langs de assen en een titel. Gebruik de functie legend om aan te geven welke grafiek bij welke variabele hoort.** |
|  | **Zorg ervoor dat de grafieken bij alle hoekwaarden onder de 0 graden gestippeld zijn (kleur maakt niet uit):**    Figuur : Schouderhoek tijdens het zwaaien van de armen. |
|  | De horizontale positie van de voet is, steeds bij hielcontact, opgemeten in een assenstelsel. Die voetposities worden mooi weergegeven met een e-macht. **Gebruik de volgende code in een nieuw script om de voetposities te genereren:**  clearvars  close all  t = 0:10;  voetPos = 5\*exp(0.1\*t);  **Kijk in de Matlab help wat de functie exp doet. Vul nu de code aan, zodat er een grafiek wordt getoond van de schredelengtes op de gegeven tijdstippen** (let op dat de waardes op de y-as kloppen!):    Figuur : Schredelengtes behorende bij de door de exp functie gegeven voetposities.  Dit is een staafdiagram. Als je niet weet welke functie je daarvoor moet gebruiken laat Google dan ‘staaf’ vertalen. |

## Antwoorden en uitwerkingen

|  |  |
| --- | --- |
|  | %% Data inladen  filenaam = 'stapdata.txt';  load(filenaam);  tijdsas = stapdata(:,1);  links = stapdata(:,2);  rechts = stapdata(:,3);    %% Data plotten  figure  hold on  plot(tijdsas,links,'r');  plot(tijdsas,rechts,'g');  hold off    %% Data aanpassen  % we willen het signaal aanpassen zodat we alleen waarden hebben die of 0  % of 1 zijn.  links2 = links;  rechts2 = rechts;  links2(links < 3000) = 0;  rechts2(rechts < 3000) = 0;  links2(links >= 3000) = 1;  rechts2(rechts >= 3000) = 1;    plot(tijdsas,links2);  title('Voetstapdata linkerbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  figure; %nieuw figuur aanmaken voor een plot  plot(tijdsas,rechts2,'r');  title('Voetstapdata rechterbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap'); |
|  | %% Data inladen  filenaam = 'stapdata.txt';  load(filenaam);  tijdsas = stapdata(:,1);  links = stapdata(:,2);  rechts = stapdata(:,3);    %% Data plotten  figure  hold on  plot(tijdsas,links,'r');  plot(tijdsas,rechts,'g');  hold off    % Bepaal eerst of het grote getal de stap of de zwaaifase is.  % Bepaal voor jezelf waar de toe-off en heel strike momenten zijn    %% Data aanpassen  % we willen het signaal aanpassen zodat we alleen waarden hebben die of 0  % of 1 zijn.  % we nemen aan dat de lengte van de vector links = rechts.  for nVal = 1:length(links)  if links(nVal) < 3000  links2(nVal) = 0;  else  links2(nVal) = 1;  end  if rechts(nVal) < 3000  rechts2(nVal) = 0;  else  rechts2(nVal) = 1;  end  end    plot(tijdsas,links2);  title('Voetstapdata linkerbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  figure; %nieuw figuur aanmaken voor een plot  plot(tijdsas,rechts2,'r');  title('Voetstapdata rechterbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap'); |
|  | Dat is persoonlijke voorkeur. Als je minder code snel beter leesbaar vind, dan heeft het eerste voorbeeld de voorkeur. Als je for-lussen en if-statements beter leesbaar vind, dan heeft dat de voorkeur. Het slim gebruiken van de Matlab mogelijkheden heeft vaak de voorkeur wat weer pleit voor het eerste voorbeeld. |
|  | Tja, daar kan alleen jij een antwoord opgeven. |
|  | % Data analyse op basis van voetschakelaardata  clearvars  close all  clc    %% Data inladen  filenaam = 'stapdata.txt';  load(filenaam);  tijdsas = stapdata(:,1);  links = stapdata(:,2);  rechts = stapdata(:,3);    %% Data plotten  figure  hold on  plot(tijdsas,links,'r');  plot(tijdsas,rechts,'g');  hold off    % Bepaal eerst of het grote getal de stap of de zwaaifase is.  % Bepaal voor jezelf waar de toe-off en heel strike momenten zijn    %% Data aanpassen  % we willen het signaal aanpassen zodat we alleen waarden hebben die of 0  % of 1 zijn.  links2 = links;  rechts2 = rechts;  links2(links < 3000) = 0;  rechts2(rechts < 3000) = 0;  links2(links >= 3000) = 1;  rechts2(rechts >= 3000) = 1;    plot(tijdsas,links2);  title('Voetstapdata linkerbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  figure; %nieuw figuur aanmaken voor een plot  plot(tijdsas,rechts2,'r');  title('Voetstapdata rechterbeen')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  %% Identificeren van stappen  % we bekijken alleen de linkerzijde  %  % Het doel is het maken van het volgende plaatje:    indHS\_L = find(diff(links2)==1);  indTO\_L = find(diff(links2)==-1);  indHS\_R = find(diff(rechts2)==1);  indTO\_R = find(diff(rechts2)==-1);  figure; %nieuw figuur aanmaken voor een plot  hold on  plot(tijdsas,links2,'b');  plot(tijdsas(indHS\_L),links2(indHS\_L),'ro');  plot(tijdsas(indTO\_L),links2(indTO\_L),'go');  title('Voetstapdata linkerbeen met Toe-off en Heel-strike')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  hold off  figure; %nieuw figuur aanmaken voor een plot  hold on  plot(tijdsas,rechts2,'r');  plot(tijdsas(indHS\_R),rechts2(indHS\_R),'ro');  plot(tijdsas(indTO\_R),rechts2(indTO\_R),'go');  title('Voetstapdata rechterbeen met Toe-off en Heel-strike')  xlabel('Tijd (seconde)');  ylabel('Detectie van een voetstap');  hold off  6, 7 en 8.  clearvars  close all  clc  t = 0:0.01:2;  alfa = 45\*cos(pi\*t);  beta = 55\*cos(1.1\*pi\*t);  figure  plot(t,alfa,'r',t,beta,'b')  xlabel('Tijd (s)')  ylabel('Hoek (graden)')  title('Schouderhoek')    ind1 = alfa < 0;  ind2 = beta < 0;  hold on  plot(t(ind1),alfa(ind1),'y--')  plot(t(ind2),beta(ind2),'y--')  legend('Alfa','Beta','Location','North')  9.  clearvars  close all  t = 0:10;  voetPos = 5\*exp(0.1\*t);  plot(voetPos)  schreden = diff(voetPos)  figure  bar(schreden)  xlabel('Tijd')  ylabel('Schredelengte') |

# Toepassingen - Lichaamszwaartepunt

In de volgende toepassing gaan we een sprongbeweging analyseren. Een proefpersoon is beplakt met vijf markers op deelzwaartepunten: op het hoofd (9 kg), de romp (40 kg), het bovenbeen (20 kg), het onderbeen (14 kg) en de voet (4 kg). De x- en y-coördinaten van de deelzwaartepunten staan in een bestand opgeslagen. De data is opgenomen met 120 beeldjes per seconde.

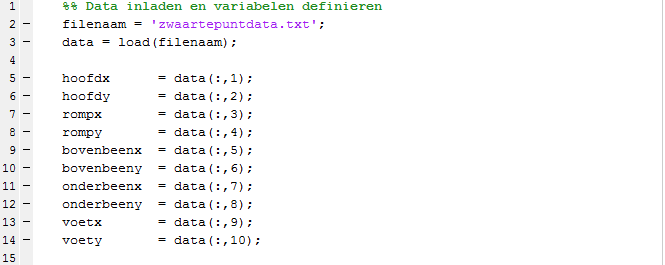
We willen uiteindelijk weten wat de hoogte is van het lichaamszwaartepunt als functie van tijd.

We pakken dit vraagstuk als volgt aan:

1. Maak, voor een goed overzicht, een plaatje van alle markers op alle gemeten tijdstippen.
2. Bepaal de coördinaten van het totale lichaamszwaartepunt
3. Plot de y-coördinaat van het totale lichaamszwaartepunt tegen de tijd

## Het plotten van de markers

Het plotten van de markers kan op een aantal verschillende manieren. We laten er hier één zien die prima werkt als je slechts 5 markers hebt, maar onhandig als je (veel) meer markers hebt. Eerst laden we de data in en slaan we de x- en y-coördinaten van alle afzonderlijke markers op:

****

Figuur : Matlab om de x- en y-coördinaten van de zwaartepunten in aparte variabelen te zetten.

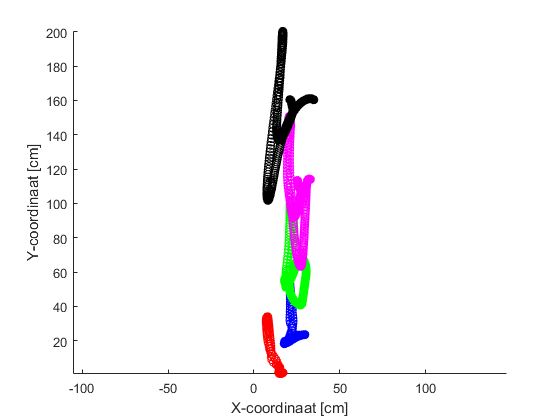
**Plot nu de x- en y-coördinaten van alle markers tegen elkaar (x op de x-as, y op de y-as) en geef elke marker een andere kleur zodat je het volgende plaatje krijgt.**



Figuur : x- en y-coördinaten van de afzonderlijke deelzwaartepunten verbonden door lijnen.

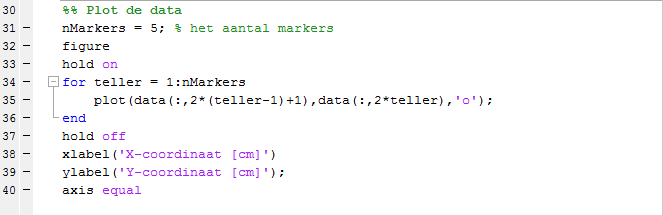
Met het commando axis equal zorg je ervoor dat de assen dezelfde schaal hebben, **gebruik dat**.

**Alle punten zijn verbonden met lijnen en dat willen we niet. Pas nu de code aan zodat je alleen rondjes om de punten krijgt.**



Figuur : x- en y-coördinaten van afzonderlijke deelzwaartepunten.

Om dit voor elkaar te krijgen, moet je vijf plot commando’s aanpassen. Je kunt je voorstellen dat dit foutgevoelig is als je (veel) meer markers hebt. Hieronder zie je code die hetzelfde doet, maar waarin slechts één keer een plot commando is gegeven binnen een for-loop:



Figuur : Matlab code

Probeer te achterhalen waarom de juiste kolommen opgehaald worden als de teller loopt van 1 tot 5 in regel 35 (tip: gebruik de debugger):

plot(data(:,2\*(teller-1)+1),data(:,2\*teller),'o');

**Vervang regel 31 (nMarkers = 5;) door een commando waarin geen getal meer voorkomt, terwijl het programma toch goed blijft werken.**

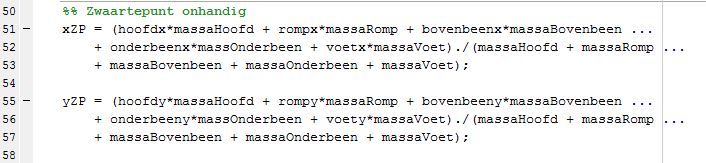
## Totale zwaartepunt

We hebben de volgende formules voor de x- en y-coördinaat van het lichaamszwaartepunt:



Waar  en  respectievelijk de x- en y-coördinaten van de deelzwaartepunten zijn. De i geeft het volgnummer cq. index aan van het deelzwaartepunt.

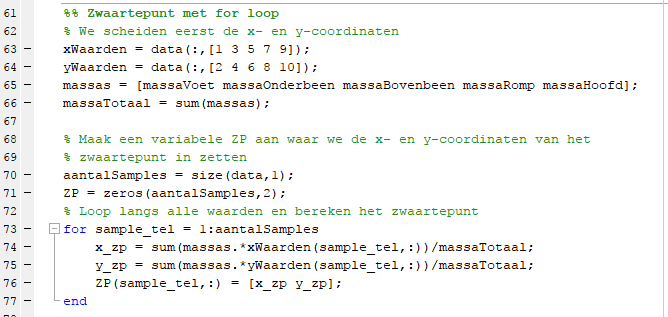
We hebben markers op een heleboel tijdstippen. We kunnen ervoor kiezen om alles in één keer uit te rekenen of voor elk tijdstip afzonderlijk. Beide manieren hebben hun voor- en nadelen. We kunnen het zwaartepunt op verschillende manieren uitrekenen. Wat we willen vermijden is code van de volgende vorm:



Figuur : onoverzichtelijke programmacode

Behalve dat je er scheel van wordt als je er te lang naar kijkt, kun je je voorstellen dat je heel eenvoudig een foutje maakt en dat het heel lastig is om die fout te vinden (er staat ergens een foutje, **probeer deze maar eens te vinden**). Je kunt typefouten maken in de namen van de variabelen, fouten in de vermenigvuldiging, haakjes, etc.

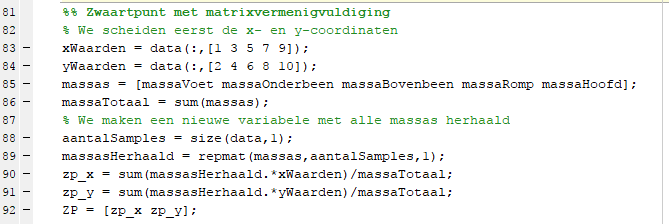
Een andere methode is om gebruik te maken van een for-loop en voor elk tijdstip het totale zwaartepunt te berekenen:



Figuur : overzichtelijke programmacode

Het bezwaar hier is dat, hoewel het overzichtelijke code is, het een trage berekening oplevert.

Tot slot kunnen we gebruik maken van een matrixvermenigvuldiging. Dit is altijd sneller bij het runnen c.q. uitvoeren van het programma, maar ziet er soms complexer uit:



Figuur : de ultieme oplossing: alles met matrixvermenigvuldigingen

In regel 89 wordt met behulp van de functie *repmat* (kijk in de help van Matlab als je hem niet snapt) een matrix gemaakt met alle massa’s herhaald onder elkaar gezet. Op deze manier kunnen we op regels 90 en 91 deze matrix vermenigvuldigen met de x- en y-coördinaten (**het puntje is essentieel, waarom ook alweer?**). Door vervolgens te sommeren en te delen door de totale massa, krijgen we voor elke sample het zwaartepunt.

Bij het delen hebben we geen punt-operator nodig, waarom niet?

**Kies de methode waar je je het prettigst bij voelt en waar de kans op fouten klein is. Schroom niet om extra tussenstappen te maken zodat de code voor jezelf en andere programmeurs begrijpelijk blijft.**

## Tijd-as

Je gaat nu de hoogte van het totale zwaartepunt plotten. Daar willen we een tijd-as voor plotten. **Gebruik dezelfde techniek als bij de voetstapdata en maak een tijd-as. Dit heb je daarbij nodig: de markers zijn met een snelheid van 120 beeldjes per seconden opgenomen.**

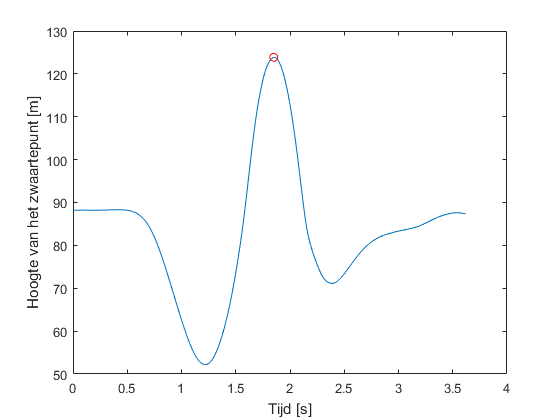
## Hoogte van het zwaartepunt tegen de tijd

We willen de hoogte (dus de y-coördinaat) van het zwaartepunt tegen de tijd plotten. Omdat we deze eerder hebben berekend en ook een tijd-as hebben, kunnen we dit eenvoudig plotten.

**Plot de hoogte van het lichaamszwaartepunt tegen de tijd.**

**Geef aan op welk *tijdstip* de hoogte maximaal is. Gebruik hier het commando max dat eerder aan de orde is gekomen.**

**Zet op dat maximum een rood rondje. Je krijgt dan het volgende plaatje:**



Figuur : Met de functie max wordt het hoogste punt van het totale lichaamszwaartepunt gelokaliseerd.

Hoe hoog heeft deze persoon nou gesprongen? Je zou 1.25 meter kunnen zeggen, maar dat is niet bepaald een eerlijke maat…

# Hoeken van segmenten

## Goniometrische functies in Matlab – graden vs. radialen

Voor we beginnen met dit onderwerp is het essentieel om het verschil tussen graden en radialen te benadrukken. Als we de functies sin, cos en tan gebruiken (maar ook hun inverse asin, acos en atan), dan gaat Matlab ervan uit dat we radialen gebruiken. Je kunt graden naar radialen omrekenen door te vermenigvuldigen met pi en te delen door 180. Als alternatief kan je gebruik maken van de functies sind, cosd, tand, asind, acosd en atand. Deze verwachten graden als input.

## Definitie van een gewrichtshoek

Je kunt op verschillende manieren een gewrichtshoek definiëren. Een handige manier is de hoek die het segment maakt t.o.v. de horizontaal:

αromp

αfemur

αtibia

αvoet

Figuur : Met de functie max wordt het hoogste punt van het totale lichaamszwaartepunt gelokaliseerd.

Voor de hoekdefinities is de draairichting tegen de klok in en de hoek wordt gemeten t.o.v. de positieve x-as (die naar rechts is gericht). De positieve y-as is overigens naar boven gericht. Deze definities zijn overeenkomstig die in Matlab worden gehanteerd. Dat is niet vanzelfsprekend. Er zijn genoeg programmeertalen waarbij de y-as naar beneden positief is benoemd. De ontwerpers van deze programmeertalen zouden zich hiervoor diep moeten schamen, maar dit terzijde.

Gegeven: de teen heeft coördinaten (0,0), de enkel (-20,10), de tibialengte is 40 cm en de tibiahoek is 60 graden. **Geef de Matlab code om de voethoek en de coördinaten van de knie uit te rekenen.**

## Uitwerking

clearvars

close all

clc

tibLengte = 40;

enkel = [-20; 10];

tibHoek = 60;

voethoek = atand(enkel(2,1)/enkel(1,1))+180

knie = enkel+tibLengte\*[cosd(tibHoek); sind(tibHoek)]

**Breid het programma uit om de coördinaten van de schouder uit te rekenen.**

Gegeven:

Femurlengte: 45 cm

Romplengte: 70 cm

Femurhoek: 1,5\*pi rad

Romphoek: 80 graden

## Uitwerking

clearvars

close all

clc

tibLengte = 40;

enkel = [-20; 10];

tibHoek = 60;

voethoek = atand(enkel(2,1)/enkel(1,1))+180

knie = enkel+tibLengte\*[cosd(tibHoek); sind(tibHoek)]

femurLengte = 45;

femurHoek = 1.5\*pi;

heup = knie+femurLengte\*[cos(femurHoek); sin(femurHoek)]

rompLengte = 70;

rompHoek = 80;

schouder = heup+rompLengte\*[cosd(rompHoek); sind(rompHoek)]

1. Mark Schrauwen [↑](#footnote-ref-1)
2. Een bestand met door een mens leesbare tekst. [↑](#footnote-ref-2)