|  |  |
| --- | --- |
| Biostatica  Door Alistair Vardy en Mark Schrauwen | Matlab Wk3.2 |

Inhoudsopgave

[Versiebeheer 2](#_Toc491869460)

[1 Inleiding 3](#_Toc491869461)

[2 Staptijden bepalen m.b.v. voetstapschakelaars 3](#_Toc491869462)

[2.1 Het signaal 3](#_Toc491869463)

[2.2 Data opschonen 4](#_Toc491869464)

[2.3 De tijds-as 4](#_Toc491869465)

[2.4 Tijden van de voetstappen 5](#_Toc491869466)

[3 Lichaamszwaartepunt 6](#_Toc491869467)

[3.1 Het plotten van de data 6](#_Toc491869468)

[3.2 Zwaartepunt van markers 7](#_Toc491869469)

[3.3 Tijds-as 9](#_Toc491869470)

[3.4 Hoogte van het zwaartepunt tegen de tijd 9](#_Toc491869471)

[4 Hoeken van segmenten 11](#_Toc491869472)

[4.1 Goniometrische functies in Matlab – graden vs. radialen 11](#_Toc491869473)

[4.2 Definitie van een gewrichtshoek 11](#_Toc491869474)

[5 Numeriek differentierenen 12](#_Toc491869475)

[5.1 Analytisch differentiëren 12](#_Toc491869476)

[5.2 Gemeten data 12](#_Toc491869477)

[5.3 Numeriek differentiëren 12](#_Toc491869478)

[5.4 Effecten van ruis en meetfouten op de numerieke afgeleide 13](#_Toc491869479)

# 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 |

# 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’ ? De volgende paragraven gaan daar op in.

Een antwoord is: omdat wij bij Bewegingstechnologie vaak met data werken. Data afkomstig van sensoren en meetsystemen. Wij willen meten aan de mens! Kwantitatieve data versus een kwalitatieve analyse. Data staat zelden in een keer in het juiste formaat. Data is zelden in een keer geschikt om een conclusie uit te trekken of erg nog 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 generieke 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. Op elk element van een vector kan een getal staan. Een matrix is een vector bestaande uit vectoren. Matlab staat voor Matrix Laboratory. Uit de naam kun je afleiden dat Matlab is gemaakt om handig en slim om te gaan met matrices. Matlab komt met allerlei faciliteiten om matrices, en dus ook vectoren, gemakkelijk te verwerken.

Concluderend, Matlab is een programma geschikt 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 kunnen kosten. Met Matlab heb je op een dag tijd een script geschreven die dat helemaal automatisch voor je doet.

Waarom gebruiken Bewegingstechnologen Matlab? Uiteindelijk… om tijd te besparen. Matlab is niet het doel, maar het hulpmiddel.

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. In dit document houden we ons voornamelijk bezig met het verwerken van data.

**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.**

# Staptijden bepalen m.b.v. voetstapschakelaars

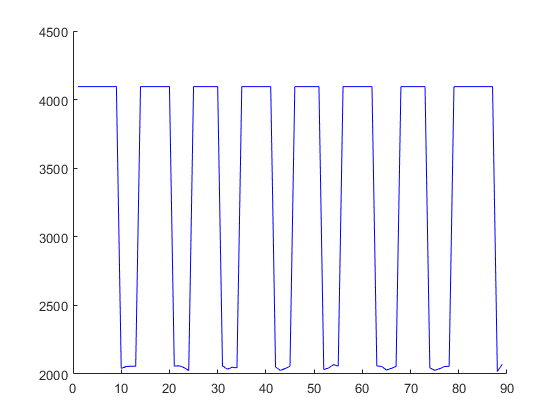
Een stap kunnen we op verschillende manieren meten. Je telefoon of een horloge doet dit met behulp van een accelerometer. Je kunt videobeelden maken en daaruit de contactmomenten bepalen. De methode die we hier gebruiken zijn voetstapschakelaars. Deze sensoren maken gebruik van de druk van de voet. Als er voldoende druk van de voet op de grond wordt gedetecteerd door een voetstapschakelaar 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).

## Het signaal

Hieronder zie je een voorbeeld van zo’n voetstapsignaal.

Zie jij aan de data wanneer de proefpersoon een stap zet?

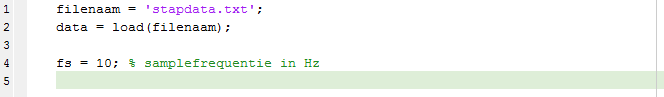
Zie jij aan de data wanneer de voet los is van de grond?



Figuur 1: Voetstap data.

[Download de file stapdata.txt](http://www.bewegingstechnologie.com/weblinks/curr17/Biostatica/Wk3_2_staptijden) en zet de file in de nieuwe folder (met een duidelijk naam).

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

Zet de volgende commando’s in een script:

Figuur 2: 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. Data.

### Het doel

Het doel van dit onderdeel is het aanpassen van het signaal zodat we alleen nog nullen en enen hebben. Op dit moment varieert het signaal in Figuur 1 ergens tussen 4100 en 2000. Je moet daarvoor bepalen wat het moment is waarop de voet de grond raakt en wanneer de voet weer los komt. Uiteraard zijn deze momenten niet zuiver te bepalen doordat de voet afrolt. Echter, we kunnen wel een goede schatting maken op basis van de voetstapdata.

## 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 of een 0.

Zie jij waar het signaal schommelt?

Je hebt in de voorgaande weken gezien hoe je beslissingen maakt m.b.v. Matlab. Je weet hoe je met vectoren kunt omgaan. Je gaat dadelijk code schrijven om de data in Figuur 1 aan te passen zodat als een proefpersoon op de voetstapschakelaar staat het signaal 1 wordt. Als de proefpersoon los is van de grond moet het signaal 0 zijn.

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 als volgt.

### Een deel 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 ongerelateerd 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 genaam 'signal'

t = 0:0.01:pi;

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

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

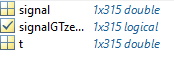


Figuur 3: de sinus in een grafiek weergegeven.

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

signalGTzero = signal > 0

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

****

Figuur 4: 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 teruggeeft. Dat is een vector die bestaat uit enen en nullen.

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

Dan nu een andere vooralsnog niet gerelateerde stap. Later gaan we de informatie in deze paragraaf samenvoegen met paragraaf 2.2.1.

Je kunt een deel van een vector selecteren op de volgende manier:

signal(logical([ 1 1 1 1 1 1 1 ]))

In dit stukje code worden de eerste zeven elementen van de vector signal weergegeven.

Hoeveel enen staan er in het voorgaande stukje code?

Met behulp van de logical() functie maken we van een vector met enen (en nullen) een vector met dezelfde enen en nullen maar dan met het type logical. Je zit 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.

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

Onderzoek het verschil tussen de laatste twee code voorbeelden.

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

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

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

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

Test zelf m.b.v. Matlab de functies 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;

Het resultaat van deze actie is te zien in de onderstaande plot:



Figuur 5: het resultaat van signal(signal<0)=0;

Hopelijk zie je hoe de vorige twee paragraven in het voorgaande code voorbeeld samen komen. Je selecteert eerst de (on)gewenste waardes van de vector signal en aan die waardes ken je een gewenste waarde toe.

Wat gebeurt er met alle andere waardes?

Alle andere waardes blijven intact. Anders zou Figuur 5 geen vormen meer van een sinus bevatten. Kortom daar doet de voorgaande code niets mee. In dit voorbeeld is krijgt de ongewenste vector waarde de waarde nul. Uiteindelijk hebben we zodoende elke negatieve waarde van een vector verwijderd. Het laatste codevoorbeeld is dus een permanent onomkeerbare acties op de vector signal.

### De diff() functie

Zoals al veel vaker aangegeven Matlab komt voorzien met 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 heel vaak is de uitleg zelf 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 geven.

Lees de onderstaande code:

clear all; 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 code uit.

Wat valt op aan de inhoud van de variabele diffVector?

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

Wat valt op aan de inhoud van de variabele diffVector?

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.

Wat is de lengte van de variabele diffVector in vergelijking met de lengte van linVector?

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 natuurlijk doordat als je het verschil berekent tussen elke opeenvolgende waarde je alleen verschillen (differences) teruggkrijgt. In een vector met de getallen 0 tot en met 2 heb je 3 getallen en 2 verschillen, namelijk: 1-0 en 2-1.

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.

## 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.  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. een of meerdere for-loops en 1 of meerdere if-statements de voetstapdata op te schonen zoals in de vorige opdracht. 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 |
|  | Breidt 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 6: de rechter voetstapdata met toe-off en heel-strike.    Figuur 7: de linker voetstapdata met toe-off en heel-strike. |
|  |  |
|  |  |
|  |  |
|  |  |

## 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  clear all; 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 |

We willen daarom het signaal aanpassen.

**Bepaal een grenswaarde en zorg ervoor dat alle waarden boven die grenswaarde gelijk worden aan 1 en alle waarden daaronder gelijk aan 0. Maak hiervoor gebruik van de theorie uit hoofdstuk 2.**

**Noem het nieuwe signaal voetstappenSchoon.**

Vervolgens willen we weten wanneer de voet de grond raakt. Dit is het geval als het nieuwe signaal van nul naar een springt.

**Bekijk de help tekst van de commando’s diff en find en bepaal op welke indices de voet de grond raakt.**

**Voer het commando verschil = diff(voetstappen) uit**

**Hoe groot is de waarde van deze vector als deze van nul naar 1 springt?**

**Gebruik het commando find om de indices te vinden waar dit het geval is.**

## De tijds-as

We weten nu op welke index de voet de grond raakt, maar nog niet hoeveel seconden ertussen zit.

We willen het vorige figuur plotten, maar nu met *tijd* op de x-as. Hiervoor is een tijds-as nodig. Een tijds-as begint meestal op tijdstip t=0. Tussen elke meetwaarde zit een bepaalde tijd. Als we zoals hier 10 datapunten per seconden meter, dan zit er 0.1 s tussen de datapunten.

Stel we zouden slechts 10 datapunten hebben. Dan zou de tijds-as er als volgt uit moeten komen te zien.

Tabel 1: xxx

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Tijd | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| Tijd algemeen | 0\*dt | 1\*dt | 2\*dt | 3\*dt | 4\*dt | 5\*dt | 6\*dt | 7\*dt | 8\*dt | 9\*dt |

Voor onze data weten we altijd de samplefrequentie (fs). Hiermee kunnen we de tijdstap dt bepalen door de relatie dt = 1/fs.

**Om een vector te maken die van 1 tot 10 loopt kunnen we het commando 1:10 gebruiken**

**Maak nu een vector in Matlab die van 0 tot 9 loopt.**

**Definieer nu de sample frequentie met het commando fs = 10;**

**Maak hieruit de tijdstap met het commando dt = 1/fs;**

**Vermenigvuldig de vector die loopt van 0 tot 9 met de tijdstap en zorg dat het resultaat hetzelfde is als in de tabel.**

Het is niet wenselijk om voor een tijds-as de tijdstap en de lengte van de vector er expliciet in te zetten. Netter is aan het begin de samplefrequentie te definiëren en daarmee de tijdstap. De lengte van de tijds-as haal je uit het aantal samples.

In ons geval is het totaal aantal samples is gelijk aan de lengte van de staptijden.

Hiervoor kunnen we het commando length gebruiken. Als de data niet in de vorm van een vector is, maar bijvoorbeeld twee kolommen met dat voor de linker en rechtervoet, dan kunnen we het commando size gebruiken. Dit commando retourneert een vector met als eerste waarde het aantal rijen en als tweede het aantal kollommen. Als we alleen het aantal rijen willen (dit is de eerste dimensie van de matrix), dan kunnen we het commando size ook als volgt aanroepen:

aantalRijen = size(data,1);

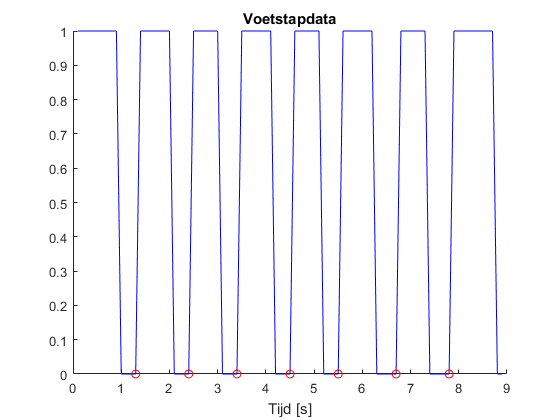
**Maak nu een tijds-as voor de gegeven voetstapdata waarin geen getallen expliciet genoemd worden. Zo kun je altijd eenvoudig een tijds-as maken zonder dat je van tevoren hoeft te weten hoeveel samples je hebt.**

## Tijden van de voetstappen

We willen nu weten op welke tijdstippen de voet is neergekomen. We hebben inmiddels de indices van deze momenten en een tijds-as

**Bepaal de tijdstippen waarop de voet neerkomt**

Als we het opgeschoonde signaal pakken en tegen de tijd uitzetten, dan kunnen we daarin de momenten waar de voet neerkomt weergeven met bijvoorbeeld een rondje.



Figuur 8: voetstapdata.

**Probeer dit figuur te reproduceren**

**Bepaal de tijdstippen waarom de voet is neergekomen**

**Gebruik het commando diff om de staptijden (de tijd tussen momenten waar de voet neer is komen) te berekenen.**

# Lichaamszwaartepunt

De eerste vraag is het plotten van markers van een sprong. We hebben 5 markers op deelzwaartepunten geplaatst; 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 staan in file 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:

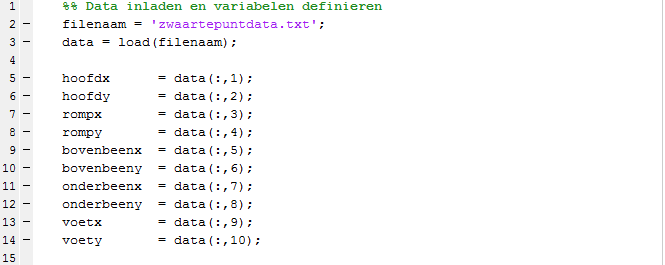
1. Maak een plaatje van alle markers.

Het is altijd goed om een overzicht van de data te hebben zodat je weet dat de meting goed is gegaan en dat er geen problemen zijn met de data.

1. Bepaal de coördinaten van het lichaamszwaartepunt
2. Plot de y-coördinaat van het lichaamszwaartepunt tegen de tijd

## Het plotten van de data

Het plotten van de markers kan op een aantal verschillende manieren. We laten er hier een zien die prima werkt als je slechts 5 markers hebt, maar foutgevoelig is 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 9: Matlab xxx

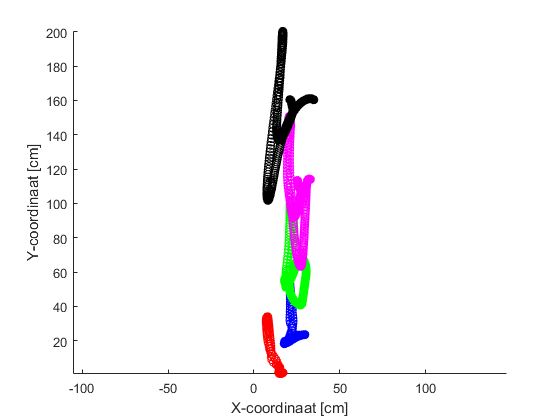
**Plot nu de x- en y-coördinaten van alle markers tegen elkaar (x op de x-as, y op de y-as) en geef elke marker een andere kleur zodat je het volgende plaatje krijgt.**



Figuur 10: xxx

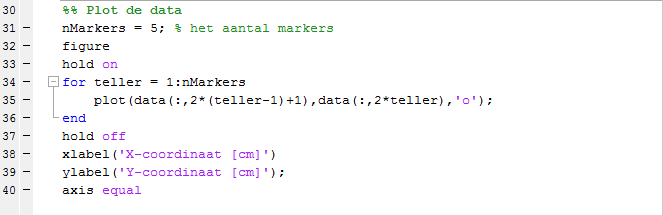
Met het commando axis equal zorg je ervoor dat de assen dezelfde schaal hebben

**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 11: xxx

Je moet nu 5 keer iets aanpassen. Je kunt je nu voorstellen dat dit foutgevoelig is als je veel meer markers hebt. Hieronder zie je code die hetzelfde doet, maar waarin slechts 1 keer een plot commando is gegeven binnen een for-loop:



Figuur 12: Matlab code

Probeer te achterhalen waarom de juiste kollommen opgehaald worden als teller loopt van 1 tot 5 in regel 35:

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

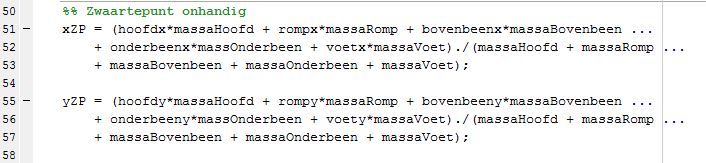
## Zwaartepunt van markers

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



Waar  en  respectievelijk de x- en y-coördinaten van de deelzwaartepunten zijn.

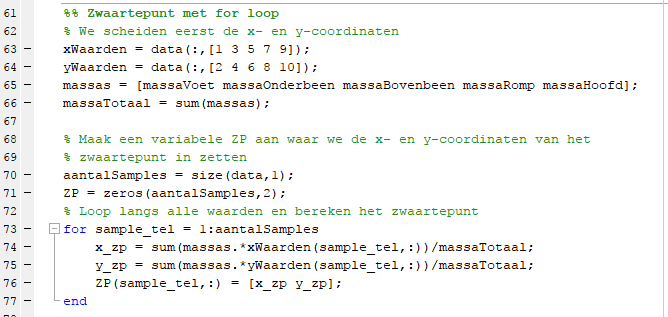
We hebben markers op een hele hoop tijdstippen. We kunnen er voor kiezen om alles in een keer uit te rekenen of voor elk tijdstip afzonderlijk. Elke heeft voor- en nadelen. We kunnen het zwaartepunt op verschillende manieren uitrekenen. Wat we willen vermijden is code van de volgende vorm



Figuur 13: xxx

Je kunt 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.

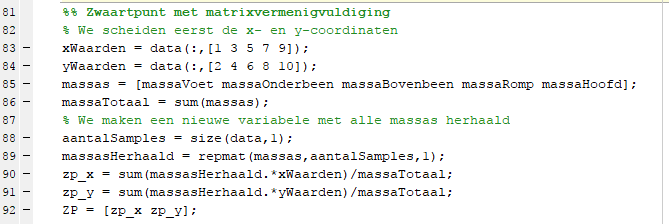
Een andere methode is gebruik maken van en for-loop en voor elk samplemoment het zwaartepunt te berekenen



Figuur 14: xxx

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

Tot slot kunnen we gebruik maken van een matrixvermenigvuldiging. Dit is altijd sneller, maar ziet er soms complexer uit.



Figuur 15: xxx

In regel 89 wordt een matrix gemaakt met alle massa’s herhaald onder elkaar gezet. Op deze manier kunnen we op regels 90 en 91 de matrix met alle massa’s vermenigvuldigen met de x- en y-coördinaten. Door vervolgens te sommeren en te delen door de totale massa, krijgen we voor elke sample het zwaartepunt.

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 je voor jezelf en andere lezers begrijpelijk blijft.

## Tijds-as

**We gaan zo de hoogte van het zwaartepunt plotten. Daar willen we een tijds-as voor hebben. Gebruik dezelfde techniek als bij de voetstapdata en maak een tijds-as. De markers zijn met een snelheid van 120 beeldjes per seconden opgenomen.**

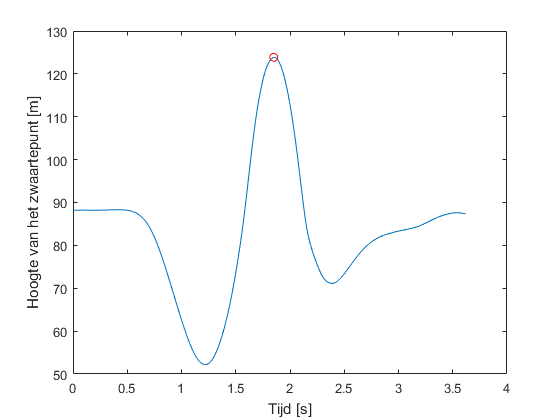
## Hoogte van het zwaartepunt tegen de tijd

We willen de hoogte van het zwaartepunt tegen de tijd plotten. De hoogte van het zwaartepunt is gelijk aan de y-coördinaat hiervan. Omdat we deze eerder hebben berekend en ook een tijds-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 16: xxx

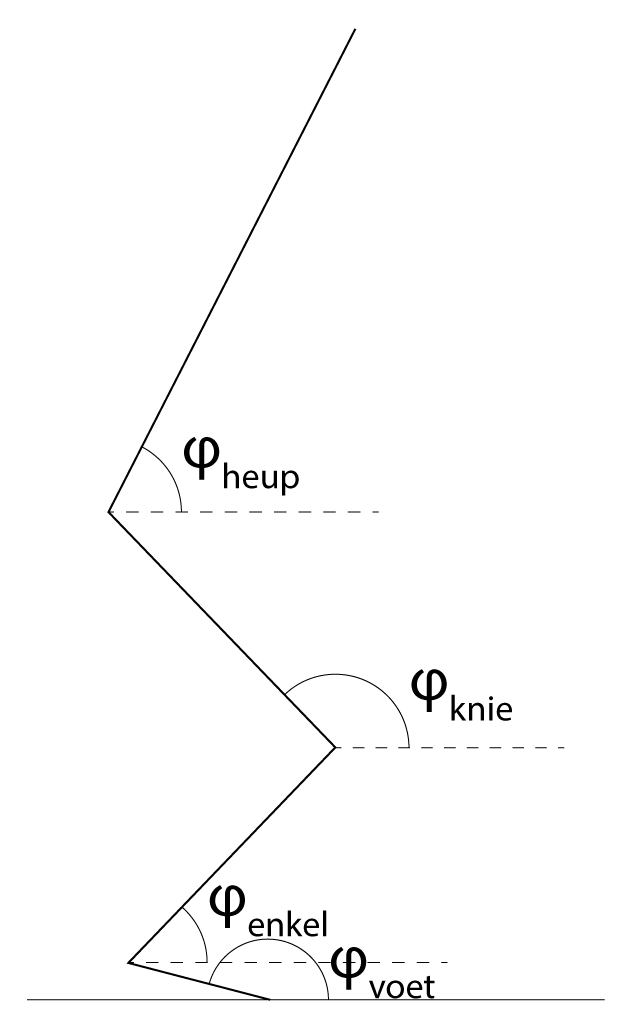
# Hoeken van segmenten

## 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 inversen asin, acos en atan), dan gaat Matlab er vanuit dat we radialen gebruiken. Je kunt graden naar radialen omrekenen of gebruik maken van de functies sind, cosd, tand, asind, acosd en atand.

## 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:



Hier wordt een zogenaamd linksdraaiend assenstelsel gebruikt, de positieve draairichting is linksom en de hoek wordt gemeten t.o.v. de horizontaal, de positieve x-as is naar rechts gericht en de positieve y-as naar boven.

# Numeriek differentierenen

We gaan hier heel kort in op het onderwerk numeriek differentiëren. Dit zal tijdens de wiskundelessen veel uitvoeriger aan bod komen.

## Analytisch differentiëren

We behandelen hier kort het begrip numeriek differentiëren. De meeste van jullie zijn bekend met het concept differentiëren. Bijvoorbeeld, de afgeleide van de functie is gelijk aan . Dit noemen we ook wel analytisch differentiëren. Hier starten we met een ruwe schatting van de verandering van over een stuk tijd . We krijgen dan het differentiequotiënt:



Als we het stuk tijd nu steeds kleiner maken, dan vinden we uiteindelijk de momentane verandering van per tijdseenheid, oftewel de afgeleide :



## Gemeten data

Deze methode werkt zeer goed als we een functievoorschrift hebben. Echter, in veel gevallen hebben we te maken met gemeten data. Als we data meten en digitaal opslaan hebben we niet op elke tijdstip een waarde. We bemonsteren (samplen) slechts op bepaalde tijdstippen een meetwaarde. De snelheid waarmee dit gebeurd wordt de samplefrequentie (fs) genoemd. Het tijdsinterval tussen twee meetwaarden is gelijk aan dt = 1/fs.

We kunnen de limiet voor de afgeleide niet nemen omdat de hoeveelheid tijd nooit kleiner kan worden dan het tijdsinterval tussen twee meetwaarden.

## Numeriek differentiëren

De meest eenvoudig vorm van numeriek differentiëren is door een eerst orde benadering de maken. Waar we bij analytisch differentiëren een limiet nemen, kiezen we een bepaalde waarde van :



Door de waarde van klein te kiezen maken we een steeds kleinere fout. Echter, als je  extreem klein kiest dan zal de fout die je maakt weer groter worden. Je kunt de gemaakte fout verder verkleinen door niet een eerste orde, maar tweede of hogere orde numerieke afgeleide te nemen.

Eerste orde numeriek differentiëren in Matlab kan m.b.v. het commando gradient in Matlab. Matlab weet echter niet hoeveel tijd er tussen twee meetpunten zit en gaat er zonder verdere informatie van uit dat dit 1 seconde is. Er zijn twee manieren om dit te verhelpen:

1. Je deelt zelf door 

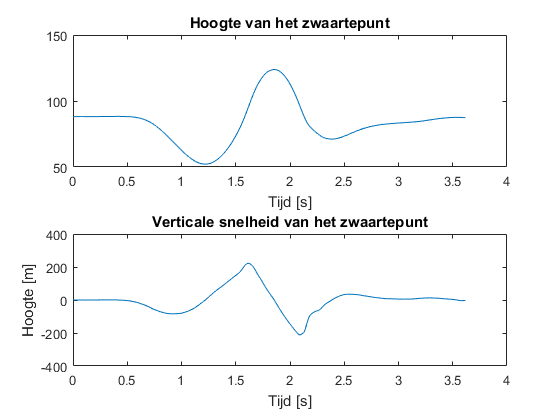
snelheid = gradient(afstand)/dt;

1. Je geeft het aan bij het aanroepen van de functie gradient

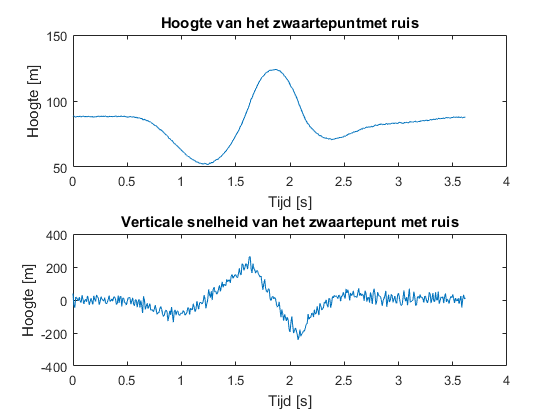
snelheid = gradient(afstand,dt);

## Effecten van ruis en meetfouten op de numerieke afgeleide

Hieronder is de hoogte van het zwaartepunt met zijn numerieke afgeleide gegeven zonder ruis



Nu voegen we een heel klein beetje ruis toe (je kunt het bijna niet zien). Echter, als je de rij getallen nu numeriek differentieert, dan wordt de ruis in het originele signaal versterkt.



Ruis ontstaat onder andere door onnauwkeurige metingen. Als er in het originele signaal veel ruis aanwezig is, dan kunnen we dit wegfilteren om zo de versterking van de ruis bij differentiëren te voorkomen. Bedenk wel dat bij filteren altijd informatie weggooit.

1. Mark Schrauwen [↑](#footnote-ref-1)