HOOFDSTUK

6

Niet-metrische MDS: iteratieve algoritmen

6.1 De Shepard-Kruskalmethode

Tot aan het begin van de jaren zestig was de Young-Householdermethode de enige manier om een MDS-analyse uit te voeren. Enkele beperkingen van de Young-Householdermethode zijn al in Hoofdstuk 2 naar voren gebracht. De voornaamste beperking is dat de methode ervan uitgaat dat de te analyseren data evenredig zijn met 'echte' afstanden. De proximities moeten op rationiveau gemeten zijn. Wil men daarentegen intervaldata analyseren, dan moet men eerst een additieve constante schatten voordat men scalaire producten kan berekenen en een eigenwaarden-eigenvectorendecompositie kan toepassen. Heeft men echter nabijheden die op ordinaal niveau gemeten zijn, dan kan men strikt genomen 'Young-Householder' in het geheel niet toepassen.
Een belangrijke doorbraak in de ontwikkeling van de meerdimensionale schaaltechnieken is te danken aan Roger Shepard (1962a,b). Deze psycholoog bedacht een methode om een MDS-oplossing te krijgen van proximities die op ordinaal niveau gemeten zijn. Zijn methode is korte tijd later aangevuld en van een wiskundig fundament voorzien door Joseph Kruskal (1964a,b). Daarmee ontstond de eerste bruikbare methode voor MDS op ordinale gegevens: de zogenaamde *niet-metrische MDS*[1]. Deze methode, die we verder zullen aanduiden als de Shepard-Kruskalmethode, maakt gebruik van een iteratief algoritme. Het eerstvolgende deel van dit hoofdstuk geeft een beschrijving van dit Shepard-Kruskalalgoritme. Daarna bespreken we het computerprogramma ALSCAL (Young & Llewyckyj, 1979) en het algoritme dat daarin gebruikt wordt.

1 De methode is dermate algemeen dat ze ook toepasbaar is op gegevens die op interval- of rationiveau gemeten zijn. Immers, de toegestane transformaties van zulke observaties zijn speciale gevallen van monotone transformaties.

6.2 ITERATIEVE ALGORITMEN VOOR MDS

Om niet-metrische MDS-problemen op te lossen maakt men gebruik van *iteratieve algoritmen*. Een algoritme bestaat uit een opeenvolging van rekenkundige bewerkingen. In een iteratief algoritme worden dezelfde berekeningen een (groot) aantal keren herhaald. Door middel van een iteratief MDS-algoritme wordt uit de ingevoerde nabijheidsgegevens een oplossing (dat wil zeggen: coördinaten op een aantal dimensies) berekend. Daarbij fungeren de coördinaten die in de vorige stap (de vorige *iteratie*) berekend zijn als invoer voor de volgende iteratie. De uitkomsten daarvan vormen dan weer de invoer van de daaropvolgende herhaling, enzovoort.
Het iteratieve algoritme is zodanig geconstrueerd dat na iedere nieuwe iteratie een betere oplossing wordt verkregen. Dat wil zeggen dat de overeenstemming (*fit*) tussen de oorspronkelijke nabijheidsdata en de afstanden die over de coördinaten van de nieuwe oplossing berekend zijn, groter is dan de *fit* van de vorige oplossing. Het algoritme wordt net zo lang herhaald tot de *fit* van de laatste oplossing nauwelijks meer verschilt (bijvoorbeeld minder dan .00001) van de *fit* van de op één na laatste iteratie. Men zegt dan dat het algoritme *geconvergeerd* is. De laatste oplossing zal men dan willen interpreteren.

Stappen in een iteratief MDS-algoritme

In een iteratief MDS-algoritme worden de volgende stappen doorlopen:

1 Het iteratieproces begint met een zogenaamde *initiële configuratie*, dat wil zeggen, een verzameling coördinaten van de objecten op een door de onderzoeker vastgesteld aantal dimensies. Zo'n initiële configuratie kan op verschillende manieren verkregen worden (later in dit hoofdstuk zullen we daarop terugkomen). Is de startconfiguratie eenmaal vastgesteld, dan bestaat iedere iteratie uit de volgende stappen:

2 In de initiële en volgende configuraties worden de afstanden tussen de objecten berekend volgens een van tevoren gekozen afstandsfunctie (meestal de Euclidische). Laten we de op iteratie t berekende afstanden in het vervolg aanduiden met $d_{ij}^{(t)}$. Deze afstanden zijn dus berekend op basis van de coördinaten $x_{is}^{(t-1)}$; we noemen ze afstanden-in-het-model of afstanden-in-de-oplossing.

3 Aangezien we aannemen dat de geobserveerde nabijheidsdata een functie zijn van de zojuist berekende afstanden-in-het-model, proberen we in iedere iteratie transformaties van de observaties, $f^{(t)}(o_{ij})$, te vinden en wel zodanig dat deze $f^{(t)}(o_{ij})$ zoveel mogelijk lijken op $d_{ij}^{(t)}$. Deze stap van het algoritme wordt de *optimal scaling*-stap genoemd. De waarden $f^{(t)}(o_{ij})$ worden in het Engels *disparities* genoemd. Dit woord zullen we in de rest van dit boek vertalen door *pseudo-afstanden*, een term die door Kruskal (1977a) is voorgesteld. De vorm van de transformatiefunctie f hangt af van onze aannamen over meetniveau, meetproces en conditionaliteit van de data. Nemen we aan dat de nabijheidsdata op intervalniveau gemeten zijn (anders gezegd: we denken dat o_{ij} een

lineaire functie van d_{ij} is), dan zoeken we voor f een functie die o_{ij} op een lineaire manier transformeert: $g(o_{ij}) = b.o_{ij} + c$. Nemen we aan dat de nabijheidsdata op ordinaal niveau gemeten zijn, dan zoeken we voor $f(o_{ij})$ een monotoon dalende of stijgende transformatie van o_{ij}.

4 Als volgende stap in iedere iteratie wordt de *fit* berekend tussen de afstanden $d_{ij}^{(t)}$ en de pseudo-afstanden $f^{(t)}(o_{ij})$. Om de *fit* uit te drukken kan men verschillende maten berekenen. Meestal gebruikt men daarvoor een zogenaamde *stress*-maat (zie verderop in dit hoofdstuk), waarvan de waarde kleiner is naarmate de overeenstemming tussen afstanden en pseudo-afstanden groter is.

5 Na het berekenen van de stress wordt nagegaan of de laatstverkregen oplossing al goed genoeg is of nog verbeterd dient te worden. Daartoe wordt in de eerste plaats gekeken naar de hoogte van de stress. Als Stress = 0, dan hebben we een perfecte oplossing bereikt en kunnen we stoppen met itereren. Hetzelfde kunnen we natuurlijk beslissen als de stress (zeer) klein is, bijvoorbeeld kleiner dan .001. Een tweede reden om te stoppen is als de stress op iteratie t niet of nauwelijks kleiner is dan die op iteratie $t-1$ (bijvoorbeeld $\text{Stress}^{(t-1)} - \text{Stress}^{(t)} < .0001$); het algoritme is dan geconvergeerd. Een derde reden om te stoppen heeft te maken met het aantal iteraties: als de stress na een groot aantal iteraties (bijvoorbeeld 75) nog steeds tamelijk groot is, en zich niet lijkt te stabiliseren, kan men zich afvragen of het nog wel de moeite waard is om door te gaan met itereren. Meestal stopt men na zo'n 50 - 100 iteraties.

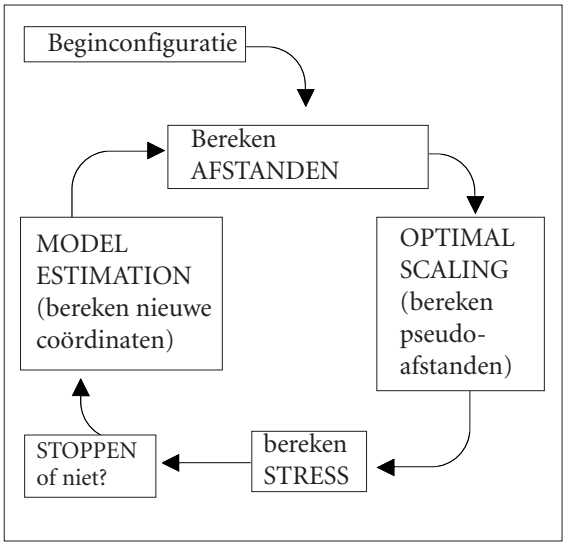
6 Wanneer er besloten is om door te gaan met itereren – er is dus nog geen bevredigende en/of gestabiliseerde oplossing verkregen – worden de 'oude' coördinaten (van iteratie $t-1$) veranderd door bij iedere coördinaat $x_{is}^{(t-1)}$ iets op te tellen:

$$x_{is}^{(t)} = x_{is}^{(t-1)} + u_{is}^{(t)}. \qquad [6.1]$$

De nieuwe coördinaat $x_{is}^{(t)}$ heet de *update* van $x_{is}^{(t-1)}$. Er zijn vele manieren om zulke updates te berekenen: daarover later meer. Wat belangrijk is, is dat de nieuwe coördinaten zodanig berekend worden dat er in de volgende iteratie in het algemeen een lagere stress gevonden zal worden. Deze stap van het algoritme heet de *model estimation*-stap. Hierna is iteratie t afgelopen en gaat het algoritme verder met Stap 2 van iteratie $t+1$. In deze stap worden opnieuw afstanden berekend, waarbij de coördinaten gebruikt worden die in de vorige *model-estimation-stap* bepaald zijn. Daarna worden verder weer alle stappen van het algoritme doorlopen, tot men op een bepaald moment besluit te stoppen.

In Figuur 6.1 is het iteratieve MDS-algoritme, dat hierboven beschreven is, schematisch weergegeven. De twee belangrijkste stappen in het algoritme zijn de *optimal scaling-stap* en de *model estimation-stap*. In de *optimal scaling-stap* worden de data zodanig getransformeerd dat de resulterende pseudo-afstanden zoveel mogelijk op de eerder berekende afstanden gaan lijken. In de *model*

estimation-stap worden de oude coördinaten bijgewerkt tot nieuwe coördinaten waarvoor geldt dat hun onderlinge afstanden beter overeenstemmen met de pseudo-afstanden.



Figuur 6.1 Schematische weergave van het iteratieve MDS-algoritme

Het zal duidelijk zijn, dat het aantal berekeningen in sommige stappen (met name de *model estimation-stap*) heel erg groot is. In de vele opeenvolgende iteraties is het aantal berekeningen zo groot dat het onmogelijk is deze zonder redelijk snelle computer uit te voeren. Er zijn dus computerprogramma's nodig die het bovengenoemde algoritme uitvoeren. Daarom zal hieronder in plaats van de term algoritme ook vaak de term programma gebruikt worden. Overigens voeren niet alle programma's alle stappen op dezelfde manier uit. In de volgende paragrafen zullen alle stappen van het algoritme nog eens gedetailleerd worden doorgenomen. Daarbij zal de nadruk liggen op de gemeenschappelijke elementen van de verschillende algoritmen en programma's.

De initiële configuratie

In principe zijn er drie typen initiële configuraties om de eerste iteratie mee te beginnen: (a) *random* configuraties, dat wil zeggen: verzamelingen coördinaten die *aselect* uit een bepaalde verdeling getrokken zijn, (b) configuraties die door de onderzoeker worden uitgekozen, en (c) configuraties die als onderdeel van het computerprogramma uit de nabijheidsdata worden berekend.
Over random beginconfiguraties kunnen we kort zijn: het nadeel ervan is dat zo'n configuratie in de meeste gevallen absoluut niet zal lijken op de 'werkelijke' configuratie van de objecten. Het computerprogramma zal dus

veel iteraties (veel computertijd) nodig hebben om van een slecht-fittende beginconfiguratie naar een goed-fittende eindoplossing te convergeren. De mogelijkheid bestaat zelfs dat het programma blijft steken in een oplossing die niet-optimaal is (daarover later meer).
In plaats van een random beginconfiguratie kan men beter een configuratie nemen die in eerder onderzoek voor dezelfde objecten gevonden is. Ook kan men als initiële configuratie coördinaten nemen die de objecten volgens een bepaalde theorie of hypothese zouden moeten hebben. In die gevallen mogen we verwachten dat de beginconfiguratie dichter bij de 'echte' configuratie zal liggen, zodat de computer sneller de eindoplossing zal bereiken en minder gauw in een sub-optimale oplossing terecht zal komen.
In de derde plaats kan men de initiële configuratie uit de te analyseren nabijheidsdata zelf berekenen. In de meeste computerprogramma's wordt daarvoor gebruikgemaakt van de Torgerson-Young-Householdermethode (zie Hoofdstuk 2). Uit de nabijheidsmaten worden scalaire producten berekend (eventueel na schatting van een additieve constante) en deze worden via eigenwaarden-eigenvectorendecompositie geanalyseerd. Zijn de data op ratio- of intervalniveau gemeten, dan is hiermee in principe de eindoplossing gevonden. Worden de data ordinaal opgevat, dan fungeren de coördinaten uit de metrische analyse als initiële configuratie om een niet-metrische oplossing te krijgen.
Bij alle typen van initiële configuraties is het mogelijk dat het programma in een sub-optimale oplossing blijft steken. De enige manier om dat na te gaan is om de data een aantal keren opnieuw te analyseren, uitgaande van verschillende (waaronder een aantal random) beginconfiguraties. Als alle beginconfiguraties uiteindelijk tot dezelfde oplossing leiden, mogen we aannemen dat we daarmee de 'echte' configuratie gevonden hebben.

Afstanden berekenen

In deze stap berekent men de afstanden tussen de objecten, uitgaande van de coördinaten uit de initiële configuratie of uit de configuratie die in de vorige iteratie berekend is. In principe kan men in deze stap elk van de afstandsfuncties gebruiken die in Hoofdstuk 4 besproken zijn. Welke men kiest, hangt dan af van inhoudelijke overwegingen over het veronderstelde verband tussen coördinaten en afstanden; anders gezegd, welk model ligt er volgens de onderzoeker aan de data ten grondslag? Slechts zelden blijken onderzoekers speciale overwegingen te hebben die tot de keuze van een specifieke niet-Euclidische afstandsfunctie aanleiding geven. Onder andere daarom wordt vrijwel altijd automatisch de Euclidische afstandsfunctie gebruikt. Bovendien kunnen er in de meeste computerprogramma's geen andere afstandsfuncties gekozen worden.

Optimal scaling

In een MDS-analyse komt het onder meer aan op het vinden van een geschikte transformatie van de nabijheidsmaten *O*. De betreffende transformatie moet voldoen aan de eisen die volgen uit het meetniveau, het meetproces en de

conditionaliteit van de data: het gaat om het vinden van een 'toegestane' transformatie. Bovendien moet het de beste transformatie zijn die we, gegeven de genoemde eisen, kunnen vinden: $f(o_{ij})$ moet zo gekozen worden dat er waarden ontstaan die overeenkomen met echte afstanden in een afbeelding. Dit hoeft niet altijd perfect te lukken! Als er meetfouten in de nabijheidsgegevens zitten, kan men de data in de praktijk nooit zo transformeren dat men perfecte afstanden krijgt in een laag-dimensionale ruimte. Ook kunnen er systematische afwijkingen in de nabijheidsdata voorkomen die ertoe leiden dat ze niet exact met (Euclidische) afstanden overeenkomen. In zulke gevallen geldt dus niet $d_{ij} = f(o_{ij})$ maar $d_{ij} \approx f(o_{ij})$.
Als de afstanden bekend zijn, is het niet erg moeilijk om de optimale transformatie $f(o_{ij})$ te vinden. Zijn de data op intervalniveau gemeten, dan is $f(o_{ij}) = k.o_{ij} + c$, waarbij k en c respectievelijk de regressiecoëfficiënt en het intercept zijn van de regressie van D op O. Is O op rationiveau gemeten, dan is $f(o_{ij}) = k.o_{ij}$ met k het regressiegewicht, onder voorwaarde dat $c = 0$. In het ordinale geval wordt veelal de door Kruskal voorgestelde monotone-regressiemethode gebruikt. Deze methode is reeds besproken in Blok 3.1, samen met Guttmans *rank image*-transformatie, een andere methode om pseudo-afstanden te vinden die in sommige computerprogramma's wordt toegepast.
Overigens wordt optimal scaling niet alleen in MDS-algoritmen, maar ook in andere analysemethoden toegepast, met name in de zogenaamde niet-lineaire multivariate analysetechnieken (zie Gifi, 1990; Young, 1981).

Fit

Bij het zoeken naar een optimale oplossing doet zich de vraag voor: hoe goed is de uiteindelijke oplossing? In MDS wordt die vraag beantwoord door te kijken naar de overeenstemming (de *goodness of fit*) tussen de afstanden in de uiteindelijke afbeelding en de pseudo-afstanden, de waarden van de getransformeerde nabijheidsmaten. Dat kan in eerste instantie door een grafiek te maken waarin de uiteindelijke afstanden uitgezet worden tegen de oorspronkelijke data. Deze grafiek wordt het *Shepard-diagram* genoemd. Als de data uit ordinaal gemeten *dis*similarities bestaan, dan krijgt men bij een perfecte MDS-oplossing een perfect monotoon stijgende (althans niet-dalende) curve in de grafiek te zien. Bij minder goede oplossingen ziet men dat de curve in het Shepard-diagram ook wel eens daalt. Hoe vaker dat gebeurt en hoe dieper de dalen zijn, hoe slechter de MDS-oplossing is. Bij metrische afstandsdata moet men idealiter een rechtlijnige grafiek krijgen die al dan niet door het nulpunt gaat (respectievelijk bij data op rationiveau en intervalniveau). Een variant van een Shepard-diagram zijn we al tegengekomen in Figuur 2.5. In die figuur zijn langs de verticale dimensie de geobserveerde afstanden tussen acht steden uitgezet tegen – op de horizontale dimensie – hun afstanden in de oplossing.
Voor een Shepard-diagram is het echter gebruikelijk dat de afstanden-in-de-oplossing langs de verticale as worden uitgezet tegen de geobserveerde data langs de horizontale as.

Naast het Shepard-diagram, dat op een grafische manier inzicht geeft in de goodness of *fit* van een oplossing, zijn er ook verschillende indices die de *fit* door middel van een getal uitdrukken. Een intuïtief voor de hand liggende goodness-of-*fit*-maat (eigenlijk: *badness of fit*!) is wat Kruskal de *ruwe stress* noemde. De ruwe stress, S^*, is gelijk aan

$$S^* = \sum_{i=1}^{m} \sum_{j=1}^{m} \{d_{ij} - f(o_{ij})\}^2. \qquad [6.2]$$

De ruwe stress is dus de som van de gekwadrateerde verschillen tussen de afstanden-in-de-oplossing d_{ij} en de pseudo-afstanden $f(o_{ij})$. Het bezwaar tegen deze grootheid is dat S^* afhankelijk is van de grootte van de afstanden en de pseudo-afstanden: als we alle afstanden en pseudo-afstanden een factor a keer zo groot zouden maken, dan wordt S^* a^2 keer zo groot. Zouden we alle afstanden en pseudo-afstanden zeer klein maken, dan wordt ook S^* zeer klein: S^* wordt zelfs nul als alle afstanden en alle pseudo-afstanden gelijk aan nul zijn. En hoewel nul de minimale waarde is die S^* aan kan nemen, is een oplossing met alle afstanden gelijk aan nul niet de oplossing die we willen hebben. Om deze reden heeft Kruskal (1964a) een gewijzigde stressformule voorgesteld waarin de ruwe stress gestandaardiseerd wordt door S^* te delen door de som van de gekwadrateerde afstanden. Deze gewijzigde formule, *Kruskal's Stress Formula 1*, ziet er in een onconditioneel, tweeweg/éénmodaal CMDS-probleem als volgt uit:

$$Stress_1 = \left[\frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \{d_{ij} - f(o_{ij})\}^2}{\sum_{i=1}^{m} \sum_{j=1}^{m} d^2_{ij}} \right]^{.5}. \qquad [6.3]$$

Kruskals formule kan op verschillende manieren aangepast worden. In het symmetrische geval hoeft men alleen maar de onder- of bovendriehoek van de afstandsmatrix te vergelijken met de onder- of bovendriehoek van de matrix met getransformeerde observaties, zodat

$$Stress_1 = \left[\frac{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \{d_{ij} - f(o_{ij})\}^2}{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d^2_{ij}} \right]^{.5}. \qquad [6.4]$$

(NB: de waarde van $Stress_1$ berekend met Formule [6.3] is natuurlijk gelijk aan die volgens Formule [6.4]).

In het rijconditionele geval kan men verschillende transformaties $f_i(o_{ij})$ toestaan (voor iedere rij *i* een aparte transformatie; zie Hoofdstuk 3). De formule wordt dan

$$Stress_1 = \left[\frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \{d_{ij} - f_i(o_{ij})\}^2}{\sum_{i=1}^{m} \sum_{j=1}^{m} d^2_{ij}} \right]^{.5} . \qquad [6.5]$$

Ook kan men deze formule aanpassen om de goodness of *fit* weer te geven van oplossingen voor drieweg/twee- en drieweg/driemodale MDS-problemen (zie Hoofdstuk 8).
In het MDS-algoritme probeert men de waarde van $Stress_1$ te minimaliseren, dat wil zeggen, men probeert een configuratie en een transformatie te vinden waarvoor geldt dat de som van de kwadraten van de verschillen tussen *D* en *f*(*O*) zo klein mogelijk is. Dat gebeurt door in de *model estimation-stap* de coördinaten van de oplossing te verbeteren en in de *optimal scaling-stap* de transformaties van de observaties te optimaliseren.

Stoppen of niet?
Nadat in iedere nieuwe iteratie de stress berekend is, moet er beslist worden of het programma verder moet gaan of moet stoppen. Deze beslissing vindt plaats aan de hand van drie criteria:

1 Is de stress laag genoeg? In de meeste computerprogramma's is een criteriumwaarde opgenomen (bijvoorbeeld .005): is de stress kleiner dan deze waarde, dan stopt het programma met de berekeningen. Is de stress groter, dan gaat het programma in principe door.
2 Is de oplossing geconvergeerd? De stress van de allerlaatste iteratie wordt vergeleken met de stress van de een-na-laatste iteratie. Is het verschil kleiner dan een bepaalde criteriumwaarde (bijvoorbeeld .0001) dan stopt het programma.
3 Is het aantal iteraties niet te groot? Wanneer het algoritme na veel iteraties niet geconvergeerd is én de stress nog onvoldoende klein is, dan is het niet aannemelijk dat de stress nog aanzienlijk zal dalen. Daarom stelt men meestal een maximum aan het aantal iteraties (bijvoorbeeld 50, 75 of 100). Voldoet de oplossing nog niet aan de eerste twee criteria, dan stopt het programma als het maximum aantal iteraties wordt overschreden.

Als een computerprogramma om een van deze redenen gestopt is, volgt er een bepaalde hoeveelheid uitvoer: in ieder geval de coördinaten van de laatstberekende oplossing en de bijbehorende stress. Meestal geeft de computer ook een overzicht van het stressverloop over de iteraties en één of meer grafieken, waaronder de configuratie van de objecten en het Shepard-diagram. Op deze uitvoer komen we later in dit hoofdstuk terug.

Model estimation

Als er besloten is om door te gaan met itereren, dan moet in de volgende stap van het algoritme de laatstberekende oplossing verbeterd worden. Dat wil zeggen dat sommige of alle coördinaten van de punten veranderd moeten worden om een betere overeenstemming van de afstanden tussen de punten en de getransformeerde observaties te verkrijgen. Dit is wiskundig de ingewikkeldste stap in elk algoritme. Het uitgangspunt bij deze stap is dat iedere coördinaat van elk object een onbekende is die een rol speelt in de verliesfunctie (de stressformule). In totaal zijn er $m \times r$ onbekende coördinaten die we willen schatten, en wel zodanig dat de waarde van de verliesfunctie (de stress) *geminimaliseerd* wordt. De algemene aanpak van zo'n probleem bestaat uit *differentiatie* van de stressfunctie naar elk van de onbekenden: we bepalen voor iedere onbekende de *partiële afgeleide* van de stress. Omdat we een minimumwaarde van de stress zoeken, worden de partiële afgeleiden gelijk aan nul gesteld. Aldus ontstaat er een stelsel van vergelijkingen waaruit we de onbekende coördinaten zouden willen oplossen.

Bij MDS kunnen we dit stelsel van vergelijkingen echter niet zonder meer oplossen (in dat geval zouden we maar één iteratie nodig hebben!). Het probleem is namelijk dat ook de partiële afgeleiden van de verliesfunctie een functie zijn van de (Euclidische) afstanden tussen de objecten, en dat die afstanden zelf weer tamelijk ingewikkelde functies zijn van de onbekende coördinaten (namelijk de wortel uit de som over de dimensies van de gekwadrateerde verschillen tussen de onbekende coördinaten op elke dimensie). Het oplossen van één onbekende coördinaat gaat dus niet zonder dat men de waarden van een aantal andere (eveneens onbekende) coördinaten kent. Daardoor bestaat er geen directe, *analytische oplossing* voor het minimaliseringsprobleem en moet er met een iteratief algoritme gewerkt worden.

Het principe van een iteratief algoritme is dat men de waarden van op één na alle onbekenden bekend veronderstelt (men substitueert dus waarden voor $(m \times r - 1)$ onbekende coördinaten) om vervolgens te proberen de ene overgebleven onbekende coördinaat uit het stelsel van vergelijkingen op te lossen. Bij MDS wordt er geen exacte oplossing van zo'n onbekende coördinaat verkregen, maar slechts een schatting daarvan. Deze schatting wordt daarna in de vergelijkingen gesubstitueerd, waarna een van de andere onbekende coördinaten wordt geschat. Dit proces herhaalt men net zolang, tot de schattingen van alle onbekende coördinaten niet meer noemenswaardig veranderen. Daarna gaat het algoritme verder met de volgende stappen: het berekenen van afstanden op basis van de zojuist geschatte coördinaten, het vinden van een nieuwe optimale transformatie en het berekenen van de stress, om tenslotte opnieuw in de *model estimation*-stap terecht te komen. Op deze wijze probeert het algoritme elke oplossing steeds een klein beetje te verbeteren en de stressfunctie stapsgewijs te minimaliseren. Dit onderdeel van het algoritme wordt nader uitgewerkt in Blok 6.1.

Aan het eind van de *model estimation*-stap beschikken we dus over een nieuwe verzameling coördinaten van de punten. Op grond van deze coördinaten

worden in de volgende iteratie weer afstanden berekend die opnieuw aanleiding geven tot nieuwe pseudo-afstanden en een nieuwe stresswaarde: enzovoort, enzovoort, net zolang tot de stresswaarde gestabiliseerd is of er om een andere reden met rekenen gestopt wordt.

Uitvoer

In Hoofdstuk 5 hebben we gezien dat een oplossing of configuratie niet uniek is. Configuraties kunnen op verschillende manieren getransformeerd worden, zonder dat de verhoudingen van de afstanden tussen de punten veranderen. De meeste computerprogramma's leveren daarom oplossingen die op een bepaalde manier gestandaardiseerd zijn. In de eerste plaats zijn de configuraties *gecentreerd*, dat wil zeggen dat op elke dimensie de som van de coördinaten gelijk aan nul is. In de tweede plaats is de stand van de assen meestal zodanig gekozen (de assen zijn zodanig geroteerd) dat ze *principale assen* vormen. Dat wil zeggen dat de coördinaten op de verschillende assen ongecorreleerd zijn (de assen zijn orthogonaal) en dat de kwadratensommen van de coördinaten op de opeenvolgende assen successievelijk afnemen. Tenslotte zijn de coördinaten meestal zodanig gekozen dat hun totale kwadratensom over alle dimensies gelijk is aan een van tevoren vastgestelde waarde, bijvoorbeeld *m*. Een alternatieve standaardisering is de coördinaten zodanig te schalen dat de som van de gekwadrateerde afstanden tussen de punten gelijk is aan een vaste waarde. De waarde waarop de coördinaten genormaliseerd worden, impliceert een bepaalde centrale dilatie van de configuratie en heeft dus geen invloed op de onderlinge verhoudingen van de afstanden.

Analyses met verschillende aantallen dimensies

Zoals gezegd, gaat het Shepard-Kruskalalgoritme bij de eerste berekeningen uit van een initiële configuratie, die ofwel door de onderzoekers is 'meegegeven' ofwel door het programma zelf berekend wordt. Een initiële configuratie bevat coördinaten van punten op één of meer dimensies. Het aantal dimensies (de *dimensionaliteit*) van de beginconfiguratie wordt aan het computerprogramma 'meegedeeld' en deze dimensionaliteit wordt vastgehouden in alle volgende programmastappen. Uiteindelijk levert het algoritme na convergentie een oplossing in het opgegeven aantal dimensies. Voor deze oplossing geldt dat de bij dit aantal dimensies behorende stress minimaal is.

Nu is het zelden zo dat een onderzoeker precies weet hoeveel dimensies nodig zijn om de nabijheidsgegevens zo goed mogelijk weer te geven. In het algemeen geldt dat hoe meer dimensies men kiest, hoe lager de stress is. Aan de andere kant wil men liever een oplossing met zo min mogelijk dimensies; zo'n oplossing is 'zuiniger' en over het algemeen gemakkelijker te interpreteren. De belangrijkste keuze bij een MDS-analyse is dus het aantal dimensies van de uiteindelijke oplossing. Daarom analyseert men de matrix met nabijheidsgegevens meestal een aantal keren, steeds met een verschillende dimensionaliteit. In de praktijk kiest men het aantal dimensies van de initiële configuratie nogal ruim (bijvoorbeeld zes) en laat men het ShepardKruskalalgoritme

achtereenvolgens oplossingen zoeken in zes, vijf, vier, drie en twee dimensies. Bij de meeste computerprogramma's kan men de boven- en ondergrens van de dimensionaliteit meegeven: het programma voert automatisch de opgegeven analyses uit, beginnend bij het grootste en eindigend bij het kleinste aantal dimensies. Als initiële configuratie van de volgende analyse neemt het programma steeds de oplossing van de voorafgaande analyse en laat daarvan de laatste dimensie weg. Na iedere analyse geeft het programma de gebruikelijke uitvoer, waaronder de stresswaarde. Het verloop van de stress als functie van het aantal dimensies geeft dan een aanwijzing over de optimale dimensionaliteit van de oplossing (daarover meer in Hoofdstuk 7).

BLOK 6.1 HET VERBETEREN VAN DE CONFIGURATIE

Het verbeteren van de configuratie vindt plaats in de zogenaamde *model estimation*-stap van het iteratieve algoritme. In deze stap wordt voor elke coördinaat van ieder punt een nieuwe waarde geschat die ervoor zorgt dat de bij die nieuwe coördinaten behorende stresswaarde lager is dan de stresswaarde die bij de vorige coördinaten hoorde. Met andere woorden: de punten in de configuratie veranderen van plaats om een betere overeenstemming tussen hun afstanden-in-de-configuratie en hun getransformeerde proximities (hun pseudo-afstanden) te verkrijgen. Sommige punten zullen naar elkaar toe bewegen, sommige van elkaar af, en sommige zullen ten opzichte van elkaar niet hoeven te veranderen.
Zoals gezegd proberen we de oplossing te verbeteren door de coördinaten van de punten op een bepaalde manier te veranderen. Van elke coördinaat die in iteratie t berekend was, wordt een verbeterde versie – de zogenaamde *update* – verkregen op iteratie $t + 1$. In het algemeen wordt de update berekend door iets op te tellen bij de waarde van de oude coördinaat:

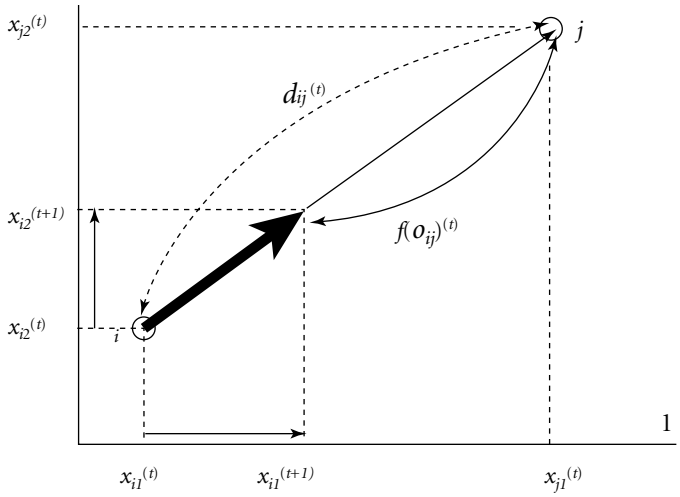
$$x_{is}^{(t+1)} = x_{is}^{(t)} + \delta_{is}^{(t)}. \qquad [6.6]$$

Hoe komen we nu aan de waarde die bij elke coördinaat opgeteld moet worden?
Deze vraag zullen we hieronder op twee manieren beantwoorden: een intuïtief-meetkundige manier en een algebraïsch-analytische manier.

De intuïtief-meetkundige aanpak

In Figuur 6.2 zijn twee punten, i en j, afgebeeld in twee dimensies. De coördinaten (op iteratie t) van deze punten zijn respectievelijk $x_{i1}^{(t)}$, $x_{i2}^{(t)}$, $x_{j1}^{(t)}$ en $x_{j2}^{(t)}$. De afstand tussen deze twee punten in de huidige configuratie is $d_{ij}^{(t)}$ en hun pseudo-afstand is $f(o_{ij})^{(t)}$. Als de afstanden tussen de punten niet gelijk zijn aan hun pseudo-afstanden dan moeten hun onder-

linge afstanden veranderd worden. Immers: de afstanden-uit-de-oplossing moeten in overeenstemming worden gebracht met de data, dat wil zeggen, met een toegestane transformatie daarvan. Stel nu dat, zoals in Figuur 6.2, $f(o_{ij})^{(t)}$ kleiner is dan $d_{ij}^{(t)}$. In dat geval moeten we de punten dus naar elkaar toe verplaatsen, en wel zodanig dat hun onderlinge afstand een factor $f(o_{ij})^{(t)}/d_{ij}^{(t)}$ maal zo groot wordt als hun huidige afstand $d_{ij}^{(t)}$. Is daarentegen $f(o_{ij})^{(t)}$ groter dan $d_{ij}^{(t)}$, dan moeten we i en j verder uit elkaar plaatsen.



Figuur 6.2 *Het berekenen van de update voor de coördinaten van punt i*

Ook in dat geval moet hun nieuwe afstand $f(o_{ij})^{(t)}/d_{ij}^{(t)}$ maal zo groot worden als hun huidige afstand. Als we de positie van punt j niet veranderen (we laten het liggen waar het ligt), dan moeten we de coördinaten van punt i dus zodanig veranderen dat

$$x_{j1}^{(t)} - x_{i1}^{(t+1)} = \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} [x_{j1}^{(t)} - x_{i1}^{(t)}] \qquad [6.7]$$

en

$$x_{j2}^{(t)} - x_{i2}^{(t+1)} = \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} [x_{j2}^{(t)} - x_{i2}^{(t)}] \qquad [6.8]$$

zodat

$$x_{i1}^{(t+1)} = x_{j1}^{(t)} - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} [x_{j1}^{(t)} - x_{i1}^{(t)}] \qquad [6.9]$$

$$= x_{j1}^{(t)} - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{j1}^{(t)} + \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{i1}^{(t)}$$

$$= x_{j1}^{(t)} - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{j1}^{(t)} + x_{i1}^{(t)} - x_{i1}^{(t)} + \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{i1}^{(t)}$$

$$= x_{i1}^{(t)} + \left(x_{j1}^{(t)} - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{j1}^{(t)} \right) - \left(x_{i1}^{(t)} - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} x_{i1}^{(t)} \right)$$

$$= x_{i1}^{(t)} + \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) x_{j1}^{(t)} - \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) x_{i1}^{(t)}$$

$$= x_{i1}^{(t)} + \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{j1}^{(t)} - x_{i1}^{(t)}].$$

Analoog is

$$x_{i2}^{(t+1)} = x_{i2}^{(t)} + \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{j2}^{(t)} - x_{i2}^{(t)}]. \qquad [6.10]$$

In het algemeen, dat wil zeggen, voor de verplaatsing van punt *i* op een willekeurige dimensie *s*, kunnen we dus schrijven

$$x_{is}^{(t+1)} = x_{is}^{(t)} + \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{js}^{(t)} - x_{is}^{(t)}]. \qquad [6.11]$$

Met behulp van deze formule kunnen we bepalen hoe we punt *i* ten opzichte van punt *j* moeten verplaatsen om een betere overeenstemming tussen de afstanden en de pseudo-afstanden te krijgen.
Het spreekt vanzelf dat we de gewenste verplaatsing van punt *i* niet alleen bepalen ten opzichte van één punt *j*, maar dat we dat doen ten opzichte van alle $m - 1$ overige punten. Daarvoor gebruiken we steeds de Formules [6.9], [6.10] en [6.11]. Voor iedere dimensie krijgen we dus $m - 1$ gewenste verschuivingen van punt *i*. Die gewenste verschuivingen zijn niet allemaal even groot. Bovendien zal de gewenste verschuiving de ene keer positief (dat wil zeggen $x_{is}^{(t+1)} > x_{is}^{(t)}$) en de andere keer negatief ($x_{is}^{(t+1)} < x_{is}^{(t)}$) of nul kunnen zijn. De vraag is nu hoe we die verschillende verschuivingen

met elkaar moeten combineren. Daarvoor zijn drie manieren mogelijk. De eerste manier om de $m-1$ gewenste bewegingen van punt i met elkaar te combineren, is het optellen van de verschuivingen over de verschillende punten. Dus

$$x_{is}^{(t+1)} = x_{is}^{(t)} + \sum_{j \neq i} \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{js}^{(t)} - x_{is}^{(t)}]. \qquad [6.12]$$

De tweede manier van combineren is de verschuivingen te middelen en daarbij te delen door $m-1$ (want punt i wordt vergeleken met $m-1$ andere punten):

$$x_{is}^{(t+1)} = x_{is}^{(t)} + \frac{1}{m-1} \sum_{j \neq i} \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{js}^{(t)} - x_{is}^{(t)}]. \qquad [6.13]$$

In de derde plaats kan men bij middeling door m delen in plaats van door $m-1$; immers: punt i heeft een verschuiving van 0 ten opzichte van zichzelf, omdat zowel $d_{ii} = 0$ en $f(o_{ii}) = 0$. Op deze manier wordt dus ook de gewenste verplaatsing van punt i ten opzichte van zichzelf meegewogen:

$$x_{is}^{(t+1)} = x_{is}^{(t)} + \frac{1}{m} \sum_{j \neq i} \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{js}^{(t)} - x_{is}^{(t)}]. \qquad [6.14]$$

De laatste combinatieregel zal in elke iteratie minder grote verplaatsingen van de punten tot gevolg hebben dan de allereerste. Wil men dus grote verplaatsingen van de punten bewerkstelligen, dan kan men beter de eerste combinatieregel gebruiken. Wil men fijnere bewegingen van de punten, dan kiest men beter de laatste. In het algemeen kan men schrijven

$$x_{is}^{(t+1)} = x_{is}^{(t)} + \alpha \sum_{j \neq i} \left(1 - \frac{f(o_{ij})^{(t)}}{d_{ij}^{(t)}} \right) [x_{js}^{(t)} - x_{is}^{(t)}] \qquad [6.15]$$

waarin de parameter α de *stapgrootte* van de gewenste veranderingen aangeeft. In de eerste iteraties kiest men vaak een grotere waarde voor de stapgrootte omdat men hoopt met grote veranderingen sneller in de buurt van de goede configuratie te komen. Na een aantal iteraties gaat men dan over op een kleinere stapgrootte, zodat de veranderingen van de coördinaten minder bruusk verlopen.

De algebraïsch-analytische aanpak

Zoals eerder gezegd is de stress een (ingewikkelde) functie van de getransformeerde observaties en de coördinaten van de punten. In principe zijn alle coördinaten onbekend, zodat de stress een functie is van een verzameling van $m \times r$ onbekende parameters. Uiteindelijk willen we een schatting van die ene verzameling parameters waarvoor geldt dat de stress minimaal is. De algemene aanpak van zo'n *optimaliseringsprobleem* is als volgt: bepaal de afgeleide van de stressfunctie, stel die op nul en los de onbekende parameters daaruit op. Omdat er veel onbekenden zijn die niet allemaal tegelijk kunnen worden opgelost, moeten er $m \times r$ partiële afgeleiden bepaald worden, voor iedere onbekende coördinaat één. Als men zo'n partiële afgeleide gelijk aan nul stelt en de onbekende oplost, dan heeft men één coördinaatswaarde gevonden die de stress minimaliseert *bij gelijkblijvende waarden van de overige coördinaten.* Het probleem bij niet-metrische MDS is dat ook de partiële afgeleiden geen eenvoudige functies van één onbekende parameter x_{is} zijn. Daardoor is het onmogelijk om de exacte waarde van de optimale x_{is} te berekenen. Wat we wel kunnen bepalen, is de *richting* waarin we x_{is} moeten veranderen als we uitgaan van een initiële verzameling coördinaten. Wat we niet weten is *hoeveel* we x_{is} moeten veranderen om de allerlaagste stress te krijgen. Dit probleem zijn we hierboven, in de meetkundige aanpak, al tegengekomen: we weten in principe in welke richting we i moeten verplaatsen, maar niet precies hoever.

Het zal duidelijk zijn dat de intuïtief-meetkundige en de analytische aanpak met elkaar samenhangen. Borg en Lingoes (1987) laten zien dat Formule [6.15] de functie weergeeft die we krijgen als we x_{is} willen oplossen uit de op nul gestelde partiële afgeleide van Kruskals ruwe stress. Formule [6.15] minimaliseert dus de verliesfunctie S^*. Het minimaliseren van Kruskals $Stress_1$ levert een ingewikkelder formule voor het verbeteren van x_{is} op, maar ook die formule is op te vatten als een functie van Formule [6.15]. Het is daarom niet zo verbazingwekkend dat het minimaliseren van S^* met behulp van Formule [6.15] uiteindelijk tot dezelfde oplossing leidt als het minimaliseren van Kruskals $Stress_1$ (Lingoes en Roskam, 1973). Men moet er alleen steeds voor zorgen dat $\sum\sum d_{ij}^2$ bij alle opeenvolgende iteraties steeds op een constante waarde wordt gesteld, bijvoorbeeld gelijk aan 1.0 gemaakt wordt.

Stel dat we bepaald hebben hoe we punt i moeten verplaatsen om een betere overeenstemming van de afstanden en de pseudo-afstanden van punt i ten opzichte van de andere punten te verkrijgen. Hierna moeten we bepalen hoe we een ander punt – laten we punt j nemen – moeten verplaatsen. Daarin telt ook de gewenste verplaatsing van j ten opzichte van i mee, maar: i hebben we net al verplaatst. Moeten we nu de nieuwe positie van i nemen om te bepalen hoe we j moeten verplaatsen, of moeten we de oude positie van i gebruiken? Stel, we kiezen de nieuwe positie van i om de

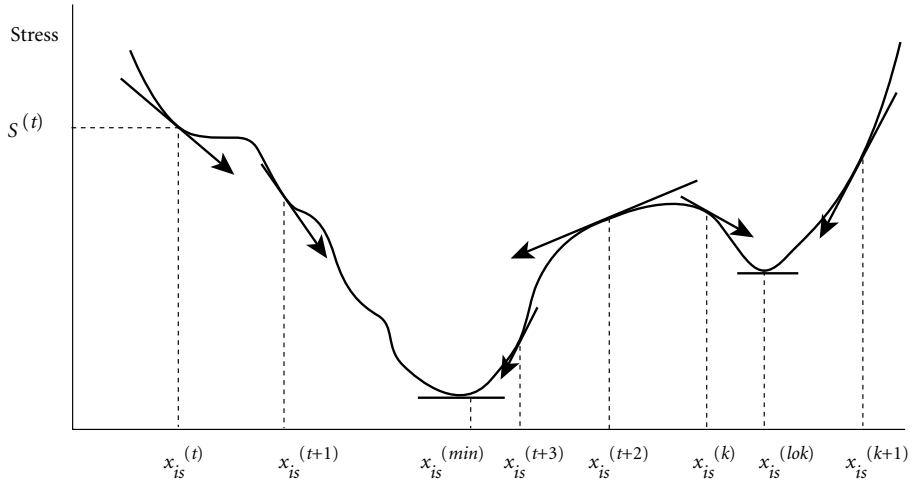
nieuwe positie van *j* te bepalen. Evengoed zouden we de nieuwe positie van *j* dan kunnen gebruiken om een nog betere nieuwe positie van *i* vast te stellen. Maar hebben we dat gedaan, dan kunnen we de nieuwe positie van *j* weer verbeteren door de laatstberekende positie van *i* te gebruiken, enzovoort, enzovoort. Veel iteratieve programma's zijn inderdaad zo opgezet. Eerst wordt begonnen met de nieuwe positie van een willekeurig punt *i* te berekenen. De nieuwe positie van *i* doet vervolgens weer mee om de nieuwe positie van *j* te berekenen, en de nieuwe posities van *i* en *j* spelen een rol in het verschuiven van punt *k*, enzovoort. Als alle punten aan de beurt zijn geweest kunnen we proberen de positie van *i* opnieuw te verbeteren, uitgaande van de nieuwe posities van *j* tot en met *m*. En dit kan voor de derde keer gedaan worden nadat alle punten voor een tweede keer verplaatst zijn. In het algemeen kan men dit iteratieve proces net zolang herhalen totdat er geen verandering meer optreedt: ieder punt ligt dan ten opzichte van de andere op de beste plaats, gegeven de verzameling pseudo-afstanden $f(o_{ij})$. Eigenlijk zijn er binnen een MDS-algoritme dus twee iteratieve processen te onderscheiden: het globale, *uitwendige* iteratieproces met *optimal scaling*, stressberekening en *model estimation*, en een *inwendig* iteratief proces in de *model estimation*-stap.

Stapgrootte en lokale minima

De hierboven beschreven methode om de coördinaten van een tussenoplossing te veranderen wordt de *gradiëntmethode* of de *method of steepest descent* genoemd. De gradiënt is namelijk de raaklijn in het punt $x_{is}^{(t)}$ aan de curve die het verband weergeeft tussen de stress en de coördinaten. De waarde van de partiële afgeleide van S^* (of $Stress_1$) naar x_{is} is de tangens van de hoek die deze raaklijn met de x_{is}-as maakt. De raaklijn geeft aan in welke richting we x_{is} moeten veranderen, namelijk in die richting waarin de raaklijn omlaag wijst. Het komt erop neer dat we in iedere iteratie de richting opzoeken met de steilste helling omlaag. Door de coördinaten in deze richting te veranderen, zouden we dus de *snelste daling* van de stress moeten krijgen.

In Figuur 6.3 is een voorbeeld van een stress-curve met raaklijnen getekend. De horizontale as in deze figuur correspondeert met alle mogelijke waarden die x_{is} kan aannemen; de verticale as geeft de stress weer die bij de verschillende waarden van x_{is} hoort. Links in de figuur is $x_{is}^{(t)}$ getekend met de bijbehorende $Stress_1^{(t)}$. Boven punt $x_{is}^{(t)}$ is de raaklijn aan de stress-curve getrokken; deze wijst van linksboven naar rechtsbeneden. Dat betekent dat we de stress omlaag kunnen krijgen door de waarde van x_{is} groter te maken, bijvoorbeeld tot hij zo groot is als $x_{is}^{(t+1)}$. Ook in het punt $x_{is}^{(t+1)}$ wijst de raaklijn van linksboven naar rechtsbeneden. Stel nu dat we x_{is} zo groot maken als $x_{is}^{(i+2)}$. De stress gaat dan weer wat omlaag, maar de

raaklijn bij $x_{is}^{(i+2)}$ loopt nu van rechtsboven naar linksbeneden: we moeten x_{is} dus wat verkleinen, maar natuurlijk niet zoveel dat we weer in het punt $x_{is}^{(t+1)}$ terechtkomen. We moeten een kleinere verschuiving uitvoeren, zodat we bijvoorbeeld uitkomen in $x_{is}^{(t+3)}$. Ook nu is de stress weer omlaaggegaan, maar nog steeds niet minimaal. Het idee is nu dat we dit proces net zolang herhalen tot we in $x_{is}^{(min)}$ zijn aangekomen.



Figuur 6.3 Grafische voorstelling van de method of steepest descent

Figuur 6.3 toont meteen een aantal problemen die zich bij deze methode kunnen voordoen. Hadden we de verandering van $x_{is}^{(t)}$ naar $x_{is}^{(t+1)}$ driemaal zo groot genomen, dan waren we in het punt $x_{is}^{(k)}$ terechtgekomen, zodat we bij verdere toepassing van de gradiëntmethode waarschijnlijk in $x_{is}^{(lok)}$ zouden eindigen. Dat is wel een minimum, maar een zogenaamd *lokaal minimum*: er is namelijk een punt op de *x*-as waarbij een lagere stress (het *globale minimum*) hoort. De *steepest descent*-methode geeft nooit een garantie dat dit globale minimum uiteindelijk ooit gevonden wordt; in principe kan het algoritme in een lokaal minimum (een van de vele) blijven steken. Het is duidelijk dat dit probleem samenhangt met de gekozen stapgrootte α: in dit voorbeeld moeten we α niet te groot kiezen. Maar stel dat we waren begonnen in $x_{is}^{(k)}$: met een kleine stapgrootte komen we dan waarschijnlijk in $x_{is}^{(lok)}$ terecht. Om in $x_{is}^{(min)}$ te eindigen zouden we in het begin juist een grote α hebben moeten gebruiken. In de praktijk wordt dit probleem opgelost door, zoals reeds gezegd, in het begin van de iteratieve procedure betrekkelijk grote stappen te kiezen en later kleinere stapjes te nemen. Het idee hierachter is: in het begin zitten we nog ver van de goede oplossing af, dus moeten we grote stappen nemen om

daarbij in de buurt te komen. Later zitten we waarschijnlijk al dichter bij de optimale oplossing, dus kunnen we beter maar kleine stapjes nemen. De precieze waarde van de stapgrootte kan voor een deel bepaald worden aan de hand van de stressverbetering uit de vorige iteratie[2]. Leverde de vorige iteratie een grote stressverbetering op, dan valt er kennelijk nog veel aan de configuratie te verbeteren, zodat we de punten beter maar een flink eind kunnen verplaatsen. Was de stressverbetering van de laatste iteratie klein, dan zitten we kennelijk al in de buurt van de optimale oplossing, zodat we nu beter kleine verplaatsingen kunnen uitvoeren, omdat we anders misschien de punten voorbij hun optimale positie laten schuiven.

Zoals gezegd, de *steepest descent*-methode geeft nooit een garantie dat dit globale minimum uiteindelijk ooit gevonden wordt. In principe kan het algoritme in een lokaal minimum blijven steken. De enige remedie hiertegen is de nabijheidsmatrix een aantal keren te analyseren met verschillende beginconfiguraties, waaronder enkele random configuraties. Komt het algoritme toch steeds in een en dezelfde oplossing terecht, dan geeft dat vertrouwen dat we met een globaal minimum te maken hebben. Komen er verschillende oplossingen naar voren, dan kiezen we uiteraard die met de laagste stress.
Naast het *steepest descent*-algoritme zijn er nog verschillende andere optimalisatiemethoden die in de MDS-technieken gebruikt worden om de stress te minimaliseren. Twee bekende algoritmen zijn het majorisatie-algoritme (De Leeuw & Heiser, 1980) dat wordt toegepast in het PROXSCAL-programma (Heiser, 1988; Busing, Commandeur & Heiser, 1997) en het *alternating least squares*-algoritme van Takane, Young en De Leeuw (1977) dat wordt toegepast in het computerprogramma ALSCAL, dat we verderop in dit boek gedetailleerd zullen bespreken.

2 Kruskal (1964b) onderscheidt vier componenten die de stapgrootte op iteratie $t + 1$ bepalen: (a) de stapgrootte op de vorige iteratie, (b) de hoekfactor, dat wil zeggen, de hoek tussen de gradiënt van iteratie t en de gradiënt van $t + 1$, (c) de relaxatiefactor, een functie van de stress op t en op $t - 5$, en (d) de *good luck factor*, een functie van de stress op t en op t-1.

6.3 ANDERE ALGORITMEN

Het Shepard-Kruskalalgoritme, dat we hierboven behandeld hebben, heeft drie kenmerkende eigenschappen: het gebruikt Kruskals Stress Formula 1 als verliesfunctie, het gebruikt Kruskals monotone regressie om de data te transformeren, en het gebruikt de gradiëntmethode om de verliesfunctie te minimaliseren. Andere algoritmen wijken vaak op één of meer van deze aspecten van het Shepard-Kruskalalgoritme af. Eén van zulke andersoortige algoritmen, ALSCAL, zal verderop in dit hoofdstuk uitgebreid behandeld worden. Twee andere algoritmen die van de Shepard-Kruskalmethode afwijken, zijn *smallest space analysis* en het *majorisatie*-algoritme. Deze worden hieronder slechts kort besproken.

Smallest space analysis

Smallest space analysis (SSA) is een programma uit de Guttman-Lingoes Series (Lingoes, 1973) dat evenals het Shepard-Kruskalalgoritme gebruikmaakt van de gradiëntmethode. Het wijkt daar echter op twee manieren van af. Ten eerste wordt, in plaats van Kruskals monotone regressie, Guttmans *rank image*-transformatie toegepast (zie Hoofdstuk 3, Blok 3.1). Ten tweede wordt niet Kruskals stress, maar de *coefficient of alienation* geminimaliseerd. Deze coëfficiënt ziet er als volgt uit:

$$K = \left(1 - \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} g(o_{ij})}{\sum_{i=1}^{m} \sum_{j=1}^{m} d^2_{ij}} \right)^{.5} \qquad [6.16]$$

waarin $g(o_{ij})$ de *rank image*-transformatie van o_{ij} is. Omdat bij een *rank image*-transformatie geldt dat $\sum_i \sum_j \{g(o_{ij})\}^2 = \sum_i \sum_j d^2_{ij}$ kan worden aangetoond dat

$$K = \sqrt{1 - \left(1 - \frac{1}{2} \; Stress_1^{\,2} \right)^2} \; .$$

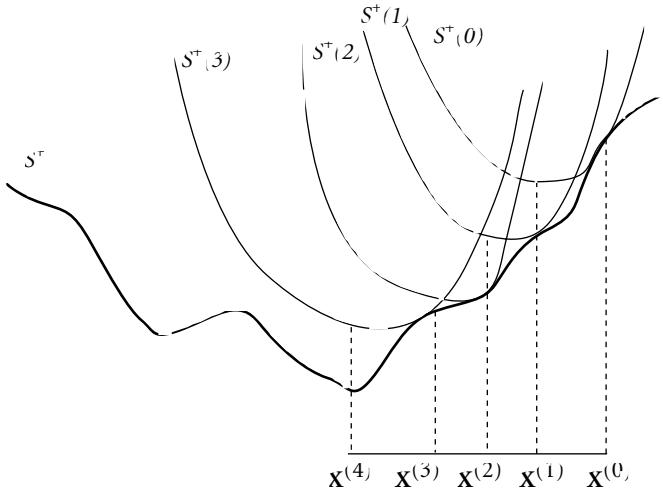
Met andere woorden, het minimaliseren van K komt op hetzelfde neer als het minimaliseren van $Stress_1$ als de observaties via de *rank image*-methode getransformeerd zijn.
In het bekende computerprogramma MINISSA (zie bijvoorbeeld Roskam & Lingoes, 1981; Davies & Coxon, 1983) worden de observaties gedurende de eerste iteraties op *rank image*-manier getransformeerd (K wordt dus geminimaliseerd), terwijl in de latere iteraties de observaties volgens Kruskals methode getransformeerd worden (zodat $Stress_1$ geminimaliseerd wordt). Op deze manier wordt getracht het convergentieproces te versnellen en een betere oplossing te krijgen.

Majorisatie

Het zogenaamde SMACOF-algoritme van De Leeuw en Heiser (1980) gebruikt zowel Kruskals *Stress*$_1$ als Kruskals monotone regressie, maar gebruikt *majorisatie* in plaats van de gradiëntmethode om de verliesfunctie te minimaliseren. De term SMACOF staat voor *Scaling by MAjorizing a COmplicated Function*. Dit algoritme berust op het idee de ruwe stressfunctie $S^* = f(\mathbf{X})$ (de stress is immers een ingewikkelde functie van de coördinaten $\mathbf{X}$) te vervangen door een hulpfunctie $S^+ = g(\mathbf{X})$ die gemakkelijker te minimaliseren is dan S^* en waarvan bekend is dat $S^+ \geq S^*$ voor alle waarden van $\mathbf{X}$. In het SMACOF-algoritme worden dan achtereenvolgens de volgende stappen doorlopen:

1 Kies een beginconfiguratie (dat wil zeggen: kies startwaarden voor de matrix met coördinaten), deze noemen we $\mathbf{X}^{(0)}$.
2 Bereken afstanden-in-de-(begin)configuratie en zoek optimale monotone transformaties van de observaties.
3 Vul deze waarden in S^+ in en minimaliseer S^+ zodat er nieuwe, optimale waarden $\mathbf{X}^{(t+1)}$ worden verkregen.
4 Bereken de stress en beslis of er gestopt moet worden.
5 Zo niet, vervang de vorige configuratie $\mathbf{X}^{(t)}$ door de nieuwe $\mathbf{X}^{(t+1)}$ en ga naar Stap 2.

Het verschil met het Shepard-Kruskalalgoritme zit in Stap 3, waar de nieuwe waarden voor $\mathbf{X}$ gevonden worden door S^+ in plaats van S te minimaliseren. Figuur 6.4 toont grafisch het verloop van het majorisatie-algoritme. Dit algoritme wordt toegepast in het MDS-programma PROXSCAL (Busing e.a., 1997).



Figuur 6.4 Weergave van het majorisatie-algoritme

6.4 ALSCAL

ALSCAL is een acroniem voor *alternating least squares scaling*. ALSCAL wijkt op twee manieren van de Shepard-Kruskalmethode af. In de eerste plaats gebruikt het een andere verliesfunctie (*S-stress*), en in de tweede plaats wordt er een andere optimaliseringsmethode (*alternating least squares*) toegepast om die verliesfunctie te minimaliseren. In ALSCAL wordt wél Kruskals monotone regressie gebruikt om de data te transformeren. Het ALSCAL-algoritme is gebaseerd op theoretisch werk van Young (1972), De Leeuw, Young en Takane (1976) en Takane, Young en De Leeuw (1977). ALSCAL is niet alleen de naam van een algoritme, maar ook van een computerprogramma dat ontwikkeld is door Young en Lewyckyj (1979).

ALSCAL is een MDS-computerprogramma voor de metrische en niet-metrische analyse van symmetrische en asymmetrische nabijheidsgegevens die tweeweg/éénmodaal, tweeweg/tweemodaal, drieweg/tweemodaal en drieweg/drie-modaal kunnen zijn. In de drieweg-gevallen kunnen de elementen van de derde weg als replicaties worden opgevat of door speciale parameters in het afstandsmodel worden weergegeven. De observaties kunnen discreet en continu getransformeerd worden, en kunnen als matrix-, rij- of onconditionele data worden opgevat. ALSCAL is dus in principe geschikt voor alle in hoofdstuk 3 genoemde MDS-problemen: CMDS, ASYMSCAL, RMDS, RAMDS, WMDS, ASINDSCAL, CMDU, RMDU en WMDU.[3]

Er zijn twee redenen waarom ALSCAL in dit boek zo uitgebreid aan de orde komt als methode om schaalproblemen op te lossen. Het is op dit moment waarschijnlijk het meest veelzijdige, voor MDS ontworpen computerprogramma. Bovendien is ALSCAL op relatief grote schaal beschikbaar, met name omdat het is opgenomen in verschillende versies van de statistische pakketten SPSS[4] en SAS. In onze bespreking van ALSCAL en zijn toepassingen zullen we steeds de *syntax*-aansturing van SPSS-versie 6.1 en 7.0 gebruiken.

In de rest van dit hoofdstuk gaan we eerst wat nader in op enkele eigenschappen van het ALSCAL-algoritme. Daarna geven we kort aan hoe ALSCAL moet worden aangestuurd en behandelen we in Blok 6.2 de niet-metrische MDS-analyse van de afstanden tussen de acht Nederlandse steden.

3 Hoewel ALSCAL *in principe* wel geschikt is voor de analyse van CMDU-, RMDU- en WMDU-problemen, is ALSCAL, net als andere computerprogramma's, in de praktijk vaak niet in staat voor deze problemen goede oplossingen te vinden. Dat is niet een speciale tekortkoming van ALSCAL, maar is inherent aan het type probleem. In Hoofdstuk 10 en 11 zullen we op deze kwestie terugkomen en andere analysemethoden aanbevelen.

4 ALSCAL is opgenomen in SPSS-X voor mainframes en in de *Professional Statistics* modules van SPSS 6.1 voor Windows (Norusˆis, 1994) en SPSS 7 voor Windows 95. Een op zichzelf staande ALSCAL-versie (ALSCALPC) voor DOS-gestuurde personal computers is te verkrijgen via INTERNET-adres *http://forrest.psych.unc.edu/research/ALSCAL.html*.

S-stress

De verliesfunctie die door ALSCAL geminimaliseerd wordt is

$$S\text{–}stress = \left(\frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \{d_{ij}^2 - (f(o_{ij}))^2\}^2}{\sum_{i=1}^{m} \sum_{j=1}^{m} \{f(o_{ij})\}^4} \right)^{.5}. \qquad [6.17]$$

Er zijn twee opvallende verschillen tussen *S-stress* en Kruskals $Stress_1$. In de eerste plaats is de teller van *S-stress* gelijk aan de wortel uit de som van de gekwadrateerde verschillen tussen de *gekwadrateerde* afstanden en de *gekwadrateerde* pseudo-afstanden. In de tweede plaats staat in de noemer de wortel uit de som van de vierde machten van de pseudo-afstanden (in plaats van de kwadraten van de afstanden). Het laatste verschil is niet essentieel, het eerste wel, omdat het een andere aanpak in de *optimal scaling*-stap mogelijk maakt. Daarop gaan we nog in.

In het algemeen is *S-stress* groter dan $Stress_1$. Het vergelijken van een ALSCAL-oplossing met een Shepard-Kruskaloplossing is daarom niet zonder meer mogelijk. In het computerprogramma is daarmee rekening gehouden, doordat voor de uiteindelijke configuratie ook Kruskals $Stress_1$ wordt uitgerekend en afgedrukt. Dit geeft een betere vergelijkingsmogelijkheid, maar men moet zich realiseren dat $Stress_1$ van een Shepard-Kruskaloplossing waarschijnlijk lager zal zijn dan van een ALSCAL-oplossing, omdat ALSCAL niet expliciet naar het minimum zoekt voor de waarde van $Stress_1$. Anders gezegd: de oplossing met minimum *S-stress* hoeft niet identiek te zijn aan de oplossing met minimum $Stress_1$.

De *alternating least squares* methode

Zoals de naam al zegt, gaat het bij de *alternating least squares* methode om het om beurten minimaliseren van een kleinste-kwadratenverliesfunctie. Deze verliesfunctie is *S-stress*, die een functie is van de verzameling afstanden $\{d_{ij}\}$ en de verzameling pseudo-afstanden $\{f(o_{ij})\}$. Het alternerend minimaliseren van *S-stress* gaat als volgt. In de optimal scaling-stap worden, uitgaande van $\{d_{ij}\}$ en $\{o_{ij}\}$, nieuwe kleinste-kwadratenschattingen van $\{f(o_{ij})\}$ gezocht die *S-stress* in de volgende iteratie kleiner maken. Dit gebeurt op precies dezelfde manier als in het Shepard-Kruskalalgoritme. In de *model estimation*-stap worden vervolgens, op grond van de zojuist gevonden $\{f(o_{ij})\}$, via een kleinste-kwadraten schattingsmethode nieuwe coördinaten berekend die nieuwe $\{d_{ij}\}$ opleveren met een lagere waarde van *S-stress*. Het verschil met de Shepard-Kruskalmethode is dus dat ook de updates van de coördinaten via een kleinste-kwadratenmethode verkregen worden. Hoe dat gaat, wordt hieronder beschreven.

Eerst constateren we dat het minimaliseren van *S-stress* op hetzelfde neerkomt als het minimaliseren van $S\text{-}stress^2$ (immers: *S-stress* is altijd positief). Bovendien is het minimaliseren van $S\text{-}stress^2$ voor een gegeven verzameling $\{f(o_{ij}\}$ hetzelfde als het minimaliseren van de teller van $S\text{-}stress^2$, dus van

$$SS^* = \sum_{i=1}^{m} \sum_{j=1}^{m} \left[{d_{ij}}^2 - \left(f(o_{ij}) \right)^2 \right]^2. \qquad [6.18]$$

Deze SS^* is opgebouwd uit m componenten, die we kunnen aanduiden als

$$SS_i^* = \sum_{j=1}^{m} \left[{d_{ij}}^2 - \left(f(o_{ij}) \right)^2 \right]^2 \qquad [6.19]$$

en die we allemaal willen minimaliseren. Immers: als iedere SS_i^* minimaal is, dan is ook SS^* minimaal. Stel nu dat we een tweedimensionale oplossing proberen te verkrijgen. In dat geval is

$$\begin{aligned} {d_{ij}}^2 &= (x_{i1} - x_{j1})^2 + (x_{12} - x_{j2})^2 \qquad [6.20] \\ &= {x^2}_{i1} + {x^2}_{j1} + {x^2}_{i2} + {x^2}_{j2} - 2x_{i1}x_{j1} - 2x_{i2}x_{j2}. \end{aligned}$$

Substitutie van [6.20] in Formule [6.19] geeft

$$SS_i^* = \sum_{j=1}^{m} \left[x_{i1}^2 + x_{j1}^2 + x_{i2}^2 + x_{j2}^2 - 2x_{i1}x_{j1} - 2x_{i2}x_{j2} - \left(f(o_{ij}) \right)^2 \right]^2 \qquad [6.21]$$

wat herschreven kan worden als

$$SS_i^* = \sum_{j=1}^{m} \left[x_{i1}(x_{i1} - 2x_{j1}) + x_{i2}(x_{i2} - 2x_{j2}) - \left(f(o_{ij}) \right)^2 + {x^2}_{j1} + {x^2}_{j2} \right]^2. \qquad [6.22]$$

Aan het begin van de model estimation-stap van iteratie t zijn de coördinaten ${x_{is}}^{(t)}$ van alle punten op alle dimensies bekend; ook zijn de pseudo-afstanden $f(o_{ij})$ bekend. $SS_i^{*(t)}$ heeft op dat moment dus een bepaalde, bekende waarde. Om op de volgende iteratie $SS_i^{*(t+1)}$ kleiner te maken proberen we voor elk afzonderlijk punt i nieuwe coördinaten te vinden. In het tweedimensionale geval gaat het dus om twee coördinaten, x_{i1} en x_{i2}, die we opnieuw willen schatten. Noemen we deze onbekenden ${x_{i1}}^{(t+1)}$ en ${x_{i2}}^{(t+1)}$ dan kunnen we formule [6.22] herschrijven als

$$\begin{aligned} SS_i^{*(t+1)} = \sum_{j=1}^{m} \Big[& {x_{i1}}^{(t+1)} \left({x_{i1}}^{(t)} - 2{x_{j1}}^{(t)} \right) + {x_{i2}}^{(t+1)} \left({x_{i2}}^{(t)} - 2{x_{j2}}^{(t)} \right) - \\ & \left(f^{(t)}(o_{ij}) \right)^2 + \left({x_{j1}}^{(t)} \right)^2 + \left({x_{j2}}^{(t)} \right)^2 \Big]^2. \qquad [6.23] \end{aligned}$$

$$\begin{aligned} = \sum_{j=1}^{m} \Big[& \left(f^{(t)}(o_{ij}) \right)^2 - \left({x_{j1}}^{(t)} \right)^2 - \left({x_{j2}}^{(t)} \right)^2 \Big] - {x_{i1}}^{(t+1)} \left({x_{i1}}^{(t)} - \right. \\ & \left. 2{x_{j1}}^{(t)} \right) - {x_{i2}}^{(t+1)} \left({x_{i2}}^{(t)} - 2{x_{j2}}^{(t)} \right) \Big]^2. \qquad [6.24] \end{aligned}$$

Deze formule heeft de vorm van de verliesfunctie $\mathcal{L} = \sum (Y - b_1 Z_1 - b_2 Z_2)^2$ die geminimaliseerd wordt bij multipele-regressieproblemen waarin men Y uit Z_1

en Z_2 wil voorspellen met regressiecoëfficiënten b_1 en b_2. De criteriumvariabele Y uit deze formule komt overeen met de verzameling van m getallen die bestaan uit $\{(f(o_{ij}))^2 - (x_{j1}^{(t)})^2 - (x_{j2}^{(t)})^2\}$ waarbij j loopt van 1 tot m. De predictoren Z_1 en Z_2 komen respectievelijk overeen met de verzamelingen $\{x_{i1}^{(t)} - 2x_{j1}^{(t)}\}$ en $\{x_{i2}^{(t)} - 2x_{j2}^{(t)}\}$. De regressiecoëfficiënten b_1 en b_2 zijn dan de beste schattingen voor $x_{i1}^{(t+1)}$ en $x_{i2}^{(t+1)}$ die $SS_i^{*(t+1)}$ minimaliseren. Het intercept α is in dit geval gelijk aan nul.
Door voor alle punten i ($i = 1,...,m$) achtereenvolgens een (multipele) regressieanalyse (met intercept $a = 0$) toe te passen, krijgt men een nieuwe verzameling optimale coördinaten. Deze coördinaten worden dan weer gebruikt om afstanden te berekenen en op basis van die afstanden worden in de optimal scaling-stap nieuwe pseudo-afstanden bepaald.[5]

De aansturing van ALSCAL

Een ALSCAL-toepassing in SPSS bestaat uit drie onderdelen: programmaregels die de invoer van de data verzorgen, programmaregels die de invoer van eventuele initiële configuraties verzorgen, en programmaregels die de eigenlijke ALSCAL-analyse besturen.
Data-invoer. ALSCAL verwacht een matrix met similarities of dissimilarities die de nabijheidsrelaties tussen een aantal rij-objecten en een aantal kolomobjecten aangeven. De kolomobjecten zijn in SPSS-terminologie de *variabelen*. Deze kunnen met specifieke namen worden aangeduid (bijvoorbeeld Groningen of Amsterdam) of door middel van symbolen als V1 tot en met V17. Indien er geen verdere aanwijzingen gegeven worden, verwacht ALSCAL een vierkante matrix met evenveel rijen als variabelen (kolommen). De afstanden tussen de acht Nederlandse steden uit Tabel 2.1 zijn als volgt in te lezen:

```
Title 'Alscal op Nederlandse steden'
subtitle 'het inlezen van de data (fixed format)'
Data list table / adm 1-3 RDM DNH UTR END ARN
                  ZWO GRO 4-31.

Begin data.
 73      0
 57     21      0
 37     57     61      0
120    111    132     87      0
 98    117    116     62     81      0
112    151    155     94    150     67      0
202    242    253    195    251    168    102      0
End data.
```

5 Het ALSCAL-algoritme wordt gedetailleerd beschreven door Young (1981) en door Young, Takane en Lewyckyj (1978). De hierboven gepresenteerde uitleg is enigszins een simplificatie.

SPSS leest nu eigenlijk een vierkante matrix die ook boven de diagonaal gevuld is met nullen. Voor de ALSCAL-analyse is dit geen probleem, omdat ALSCAL in principe een symmetrische matrix verwacht en daarom alleen de onderdriehoek van de afstandsmatrix gebruikt. We kunnen natuurlijk ook de volledige matrix inlezen (dat is nodig bij asymmetrische of rijconditionele data). Het inlezen van data hoeft niet *per se* zoals boven met *fixed format*, maar kan ook met *free format*. Dat gaat als volgt:

```
Title 'Alscal op Nederlandse steden'
subtitle 'het inlezen van de data (free format volledig)'
Data list free / ADM RDM DNH UTR END ARN ZWO GRO.

Begin data.
0  73  57  37  120  98  112  202  73  0  21  57  111  117  151
242  57  21  0  61  132  116  155  253  37  57  61  0  87  62  94
195  120  111  132  87  0  81  150  251  98  117  116  62  81  0
67  168  112  151  155  94  150  67  0  102  202  242  253  195
251  168  102  0
End data.
```

In bepaalde gevallen kan het voorkomen dat men een nabijheidsmatrix heeft waarin sommige cellen leeg zijn; die elementen noemt men *missing values*. Deze moet men aangeven door een getal in de matrix in te vullen dat kleiner is dan alle andere, niet-ontbrekende waarden in de matrix. Omdat afstandsdata meestal positief zijn, gaat ALSCAL ervan uit dat *negatieve waarden* missing values vertegenwoordigen. Wil men dat niet, dan moet men bij de besturingscommando's van ALSCAL een zogenaamde *cutoff*-waarde opgeven (zie verderop).

Inlezen van een initiële configuratie. Ieder iteratief MDS-programma heeft een initiële configuratie nodig om de analyse mee te beginnen. Als we verder niets opgeven, berekent ALSCAL zelf zo'n beginconfiguratie die gebaseerd is op een Young-Householderanalyse van de nabijheidsdata *alsof* het metrische data zijn. Stel dat we willen kijken of en in hoeveel iteraties het ALSCAL-algoritme een oplossing kan vinden voor het stedenprobleem, als er gestart wordt met een *random* beginconfiguratie. Zo'n beginconfiguratie moet voorafgaand aan de matrix met nabijheidsdata, apart worden ingelezen en moet als SPSS *system-file* worden bewaard. Alleen dan kan deze file door ALSCAL worden aangeroepen.

Bijvoorbeeld:

```
Title 'Alscal op Nederlandse steden'
subtitle 'het inlezen van een initïele configuratie'.
Data list table / DIM1 1-4 DIM2 6-9 Type_ 11-16 (A).

Begin data.
-.09 -.54 config
-.42  .01 config
-.80  .06 config
-.06  .26 config
  .57  .79 config
  .52 -.80 config
-.45  .68 config
  .59 -.48 config
End data.

Save / outfile = 'randcon.sys'.
* Het inlezen van de data (free format volledig).
Data list free / ADM RDM DNH UTR END ARN ZWO GRO.

Begin data.
0  73  57  37  120  98  112  202  73  0  21  57  111  117  151
242  57  21  0  61  132  116  155  253  37  57  61  0  87  62  94
195  120  111  132  87  0  81  150  251  98  117  116  62  81  0
67  168  112  151  155  94  150  67  0  102  202  242  253  195
251  168  102  0
End data.
```

Later kan bij de eigenlijke ALSCAL-analyse het bestand RANDCON.SYS als initiële configuratie worden aangeroepen (dat gebeurt door in de ALSCAL-besturing de regel /FILE='RANDCON.SYS' CONFIG INITIAL op te nemen).
Besturing van de analyse. Het ALSCAL-programma wordt aangeroepen met het commando ALSCAL, gevolgd door de namen van de variabelen die geanalyseerd moeten worden. Dit moeten vanzelfsprekend de variabelen zijn die zojuist zijn ingelezen, of – wat alleen in sommige gevallen mogelijk is – een deelverzameling daarvan. Vervolgens moet een aantal kenmerken van de data worden ingevoerd. In de eerste plaats moet worden gespecificeerd wat de vorm (SHAPE) van de datamatrix is. Als we de vorm niet nader beschrijven, dan wordt per *default* aangenomen dat de matrix vierkant en symmetrisch (SYMMETRIC) is. In de tweede plaats moet gespecificeerd worden of het om SIMILARities of om DISSIMILARities gaat, wat het meetniveau (LEVEL) is, of het meetproces continu is (UNTIE) en wat de CONDITIONaliteit van de data is. Als deze

kenmerken niet gespecificeerd worden, volgt per default een niet-metrische analyse waarin de data worden opgevat als ordinale, matrix-conditionele dissimilarities waarvan eventuele *ties* gelijke transformatiewaarden moeten krijgen. Als de ingevoerde tweeweg/éénmodale data inderdaad vierkant en symmetrisch zijn, is het enige afstandsmodel dat ALSCAL kan toepassen het Euclidische afstandsmodel. In dit geval hoeft het afstandsmodel niet nader gespecificeerd te worden, ook al kan men dit doen door middel van /MODEL = EUCLID. Bij andersoortige data kunnen er andere modellen gekozen worden, bijvoorbeeld /MODEL = ASCAL bij asymmetrische data.
Ten slotte kan men de dimensionaliteit van de oplossing(en) bepalen, alsmede het maximum aantal iteraties, het convergentiecriterium, de minimum S-stresswaarde waarbij het programma moet stoppen en de eerdergenoemde CUTOFF-waarde om missing values te definiëren. Een niet-metrische analyse van de stedendata kunnen we op verschillende manieren opgeven. De eenvoudigste manier is

```
ALSCAL VARIABLES = ADM TO GRO.
```

Deze opdracht heeft per default hetzelfde resultaat als

```
Alscal Variables = ADM, RDM, DNH UTR, END, ARN, ZWO,GRO
  /Shape = symmetric
  /Level = ordinal
  /Condition = matrix
  /Model = euclid
  /Criteria    dimension (2)
                 Cutoff (0)
                 Iter (30)
                 Convergence (.001)
                 Stressmin (.05).
```

De hierboven vermelde CRITERIA zijn de standaardwaarden die de ALSCAL-programmeurs hebben aangebracht. In het algemeen is het aan te raden om het maximum aantal iteraties flink te vergroten, dus liever honderd dan dertig. Ook is het raadzaam het criterium voor CONVERGENCE aanzienlijk kleiner te maken, bijvoorbeeld .00005, omdat bij .001 de oplossing vaak nog niet gestabiliseerd is.
Behalve een aantal commando's om de kenmerken van de data en de analyse te definiëren, kent ALSCAL verschillende opdrachten die de uitvoer van het programma besturen. De eerste hiervan is /PRINT = DATA HEADER. Deze zorgt ervoor dat zowel de ingevoerde data als de uiteindelijke transformaties (de pseudo-afstanden of disparities) worden afgedrukt. Bovendien wordt er,

voorafgaand aan de analyse, een compleet overzicht gegeven van de eigenschappen van de data en de opties van de analyse. Met het commando /PLOT DEFAULT wordt een aantal grafieken opgevraagd die ALSCAL ook zonder dit commando geeft. In het symmetrische tweeweg/éénmodale geval worden er één of meer grafieken van de stimulusconfiguratie afgedrukt (één grafiek bij twee dimensies, $r(r-1)/2$ grafieken bij r dimensies). Daarna volgen drie grafieken die inzicht in de goodness-of-*fit* geven: een grafiek van de afstanden uitgezet tegen de observaties (het Shepard-diagram), een grafiek van de transformaties, de pseudo-afstanden, uitgezet tegen de observaties, en een grafiek van de afstanden uitgezet tegen de transformaties. In de ALSCAL-uitvoer heten deze grafieken respectievelijk *plot of nonlinear fit, plot of transformations* en *plot of linear fit*. Tenslotte kan men door middel van het commando /OUTFILE = 'file-naam' de configuratie voor latere bewerkingen bewaren in een SPSS system file.
Na het aanbrengen van de eerdergenoemde wijzigingen in de /CRITERIA is het ALSCAL-programma toegepast op onze stedendata.

BLOK 6.2 NIET-METRISCHE ANALYSE VAN DE AFSTANDEN TUSSEN ACHT STEDEN

De resultaten van deze analyse worden besproken in Blok 6.2.

Om met ALSCAL een niet-metrische analyse van de afstanden van Tabel 2.1 tussen de acht Nederlandse steden te verrichten is onderstaand SPSS-programma uitgevoerd.

```
Title 'Alscal op Nederlandse steden'
subtitle 'niet-metrische analyse'
Data list table / adm 1-3 RDM DNH UTR END ARN ZWO GRO 4-31.

Begin data.
  0
 73      0
 57     21      0
 37     57     61      0
120    111    132     87      0
 98    117    116     62     81      0
112    151    155     94    150     67      0
202    242    253    195    251    168    102    0
End data.
```

```
Alscal      variables=ADM, RDM, DNH, UTR, END, ARN, ZWO, GRO
            /Shape = symmetric
            /Level = ordinal
            /Condition = matrix
            /Model = euclid
            /Criteria       dimension (2)
                            cutoff (0)
                            iter (100)
                            convergence (.00005)
                            stressmin (.005)
            /Print data header
            /Plot default.
Finish.
```

Hieronder volgt een afdruk van (delen van) de uitvoer van dit programma, op sommige plaatsen voorzien van commentaar. Het eerste stuk van de uitvoer is de zogenaamde *header*, waarin de opgegeven kenmerken van de data, de analyse en de uitvoer worden opgesomd. (Afhankelijk van de gebruikte versie van SPSS en ALSCAL kan de uitvoer er verschillend uitzien).

```
Alscal procedure options

Data options-

Number of rows (observations/matrix) .......   8
Number of columns (variables) ..............   8
Number of matrices .........................   1
Measurement level ..........................   ordinal
Data matrix shape ..........................   symmetric
Type .......................................   dissimilarity
Approach to ties............................   leave tied
Conditionality .............................   matrix
Data cutoff at .............................   0.000.000
```

```
Model options-

Model ..............................   Euclid
Maximum dimensionality .............   2
Minimum dimensionality .............   2
Negative weights ...................   not permitted

Output options-

Job option header...................   printed
Data matrices ......................   printed
Configurations
and transformations ................   plotted
Output dataset .....................   not created
Initial stimulus coordinates .......   computed

Algorithmic options-

Maximum iterations .................   100
Convergence criterion...............   0.00005
Minimum S-stress ...................   0.00500
Missing Data Estimated by ..........   Ulbounds
Tiestore ...........................   28
```

Nu volgt een afdruk van de ingelezen datamatrix. Omdat de afzonderlijke matrices van een driewegprobleem vaak corresponderen met verschillende proefpersonen, noemt ALSCAL dit de matrix van Subject 1. In dit tweeweg/ éénmodale geval is er slechts één zo'n matrix.

```
Raw (unscaled) data for subject 1
          1       2       3       4       5       6       7       8
1       0.000
2      73.000   0.000
3      57.000  21.000   0.000
4      37.000  57.000  61.000   0.000
5     120.000 111.000 132.000  87.000   0.000
6      98.000 117.000 116.000  62.000  81.000   0.000
7     112.000 151.000 155.000  94.000 150.000  67.000   0.000
8     202.000 242.000 253.000 195.000 251.000 168.000 102.000  0.000
```

```
>Warning # 14654
>The total number of parameters being estimated (the number of stimulus coordinates plus the
number of weights, if any) is large relative to the number of data values in your data matrix. The
results may not be reliable since there may not be enough data to precisely estimate the value of
the parameters. You should reduce the number of parameters (e.g. request fewer dimensions) or
increase the number of observations.
Number of parameters is 16. Number of data values is 28.
```

ALSCAL waarschuwt voor het feit dat er $8 \times 2 = 16$ coördinaten berekend moeten worden terwijl er maar 28 gegevens zijn. In Hoofdstuk 7 komen we op dit probleem terug.

```
Iteration history for the 2 dimensional solution (in squared distances) Young S-stress
formula 1 is used.
```

Iteration	S-stress	Improvement
1	0.02883	
2	0.02142	0.00741
3	0.01759	0.00383
4	0.01608	0.00151
5	0.01515	0.00093
6	0.01437	0.00078
7	0.01369	0.00068
8	0.01310	0.00059
9	0.01259	0.00052
10	0.01213	0.00045
11	0.01182	0.00032
12	0.01172	0.00010
13	0.01165	0.00007
14	0.01158	0.00007
15	0.01152	0.00006
16	0.01146	0.00006
17	0.01140	0.00006
18	0.01134	0.00006
19	0.01129	0.00005
20	0.01125	0.00004

$$\text{RSQ} = \frac{\left[\sum_{i=2}^{m}\sum_{j=1}^{i-1}\left(f(o_{ij}) - \frac{\sum_{i=2}^{m}\sum_{j=1}^{i-1} f(o_{ij})}{m(m-1)/2}\right)\left(d_{ij} - \frac{\sum_{i=2}^{m}\sum_{j=1}^{i-1} d_{ij}}{m(m-1)/2}\right)\right]^2}{\left[\sum_{i=2}^{m}\sum_{j=1}^{i-1}\left(f(o_{ij}) - \frac{\sum_{i=2}^{m}\sum_{j=1}^{i-1} f(o_{ij})}{m(m-1)/2}\right)^2\right]\left[\sum_{i=2}^{m}\sum_{j=1}^{i-1}\left(d_{ij} - \frac{\sum_{i=2}^{m}\sum_{j=1}^{i-1} d_{ij}}{m(m-1)/2}\right)^2\right]}.$$

6 De in dit boek afgebeelde ALSCAL-grafieken zijn gemaakt met een SPSS-versie voor mainframe computers. Met SPSS voor Windows worden fraaiere afbeeldingen verkregen.

Iterations stopped because S-stress improvement less than 0.000050 Stress and squared correlation (RSQ) in distances RSQ values are the proportion of variance of the scaled data (disparities) in the partition (row, matrix, or entire data) which is accounted for by their corresponding distances.

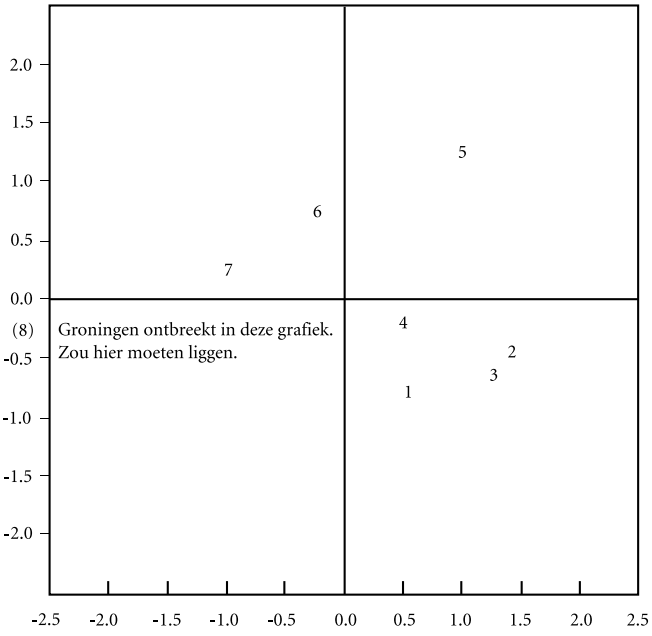
Stress values are Kruskal's stress formula 1.
For matrix stress = 0.014
RSQ = 0.999

CONFIGURATION DERIVED IN 2 DIMENSIONS

STIMULUS COORDINATES

STIMULUS NUMBER	STIMULUS NAME	PLOT SYMBOL	DIMENSION 1	DIMENSION 2
1	ADM	1	0.3953	-0.6253
2	RDM	2	1.2353	-0.4644
3	DNH	3	1.2165	-0.6041
4	UTR	4	0.3665	-0.0966
5	END	5	0.8893	1.2515
6	ARN	6	-0.2696	0.6624
7	ZWO	7	-1.1008	0.2281
8	GRO	8	-2.7325	-0.3515

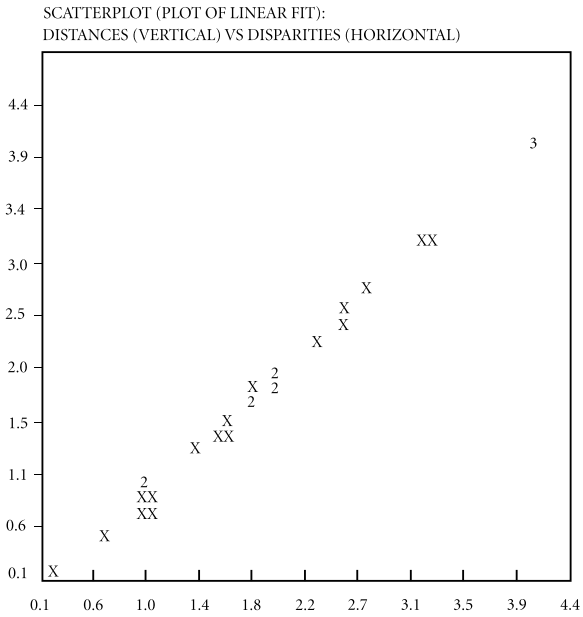
DERIVED STIMULUS CONFIGURATION:
DIMENSION 1 (HORIZONTAL) VS DIMENSION 2 (VERTICAL)



Uit bovenstaand overzicht blijkt dat ALSCAL er 20 iteraties over gedaan heeft om een *S-stress* van .01125 te bereiken. De *S-stress* op iteratie 19 bedroeg .01129, zodat het verschil tussen de laatste twee *S-stress*waarden gelijk is aan .00004 en dus kleiner is dan .00005, het opgegeven convergentiecriterium. De gevonden oplossing heeft een Kruskal $Stress_1$ van .014; deze oplossing kunnen we dus *goed* noemen.
De RSQ (*R-square*) is een andere *goodness-of-fit*-maat. Het is de gekwadrateerde correlatie tussen de pseudo-afstanden en de afstanden-in-de-oplossing. In het CMDS-geval, met vierkante en symmetrische data, wordt RSQ berekend over de onderste helften van de matrices met afstanden en pseudo-afstanden, zonder de nullen van de diagonalen. De formule is

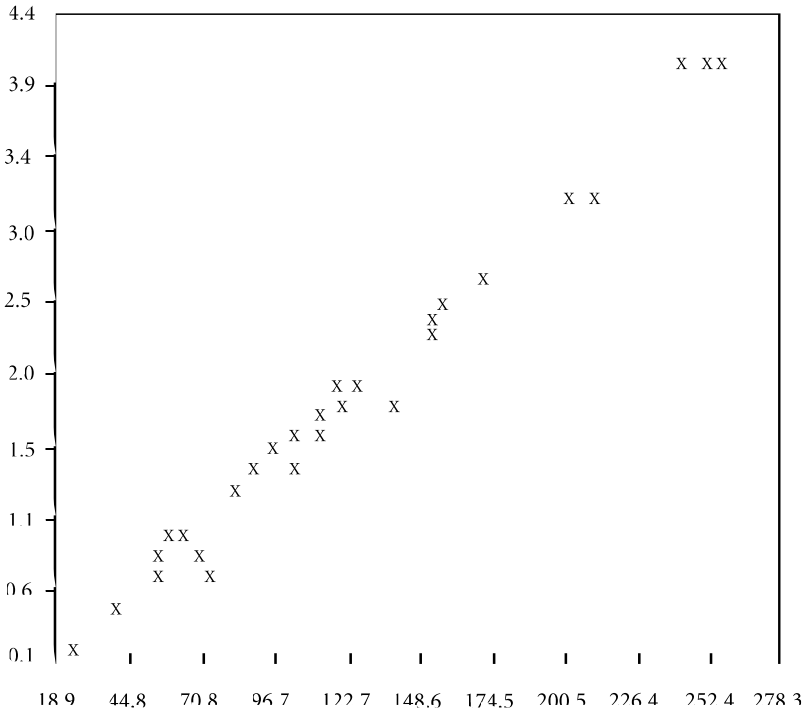
[6.25]

In andere gevallen dan CMDS worden alle $m \times m$ elementen in de berekening betrokken, ook de nullen op de diagonaal! De sommaties over i en j lopen dan steeds van 1 tot m. RSQ is een gekwadrateerde correlatie en is dus gelijk aan de proportie variantie van de pseudo-afstanden die verklaard wordt door de afstanden-in-de-oplossing. RSQ neemt gauw hoge waarden aan. Hoge RSQ-waarden zijn daarom niet erg informatief; lage



RSQ-waarden zijn daarentegen een duidelijke aanwijzing dat een oplossing een slechte *fit* heeft.
Na de *fit*-maten wordt de eigenlijke oplossing afgedrukt: een tabel met de coördinaten van de 'stimuli', hier dus de acht steden. De tabel wordt gevolgd door één of meer grafieken waarin alle dimensies paarsgewijs tegen elkaar zijn uitgezet. In dit geval is er slechts één grafiek. Bij een vierdimensionale oplossing zijn er zes grafieken (Dimensie 1 versus 2, 1 versus 3, 1 versus 4, 2 versus 3, 2 versus 4 en 3 versus 4).[6]

PLOT OF NONLINEAR FIT:
DISTANCES (VERTICAL) VS OBSERVATIONS (HORIZONTAL)



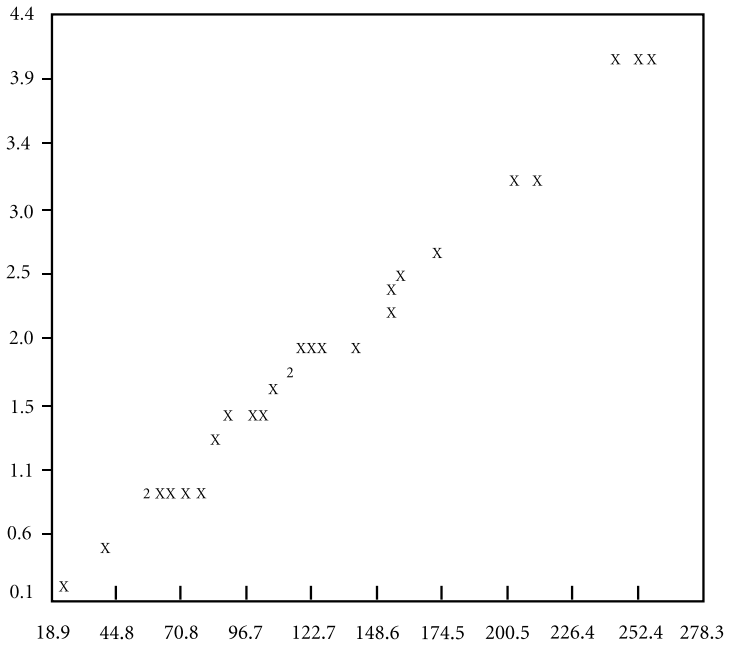
Na de grafieken van de configuraties volgt de matrix met de pseudo-afstanden, Kruskals monotone transformaties van de data.

Optimally scaled data (disparities) for subject 1

	1	2	3	4	5	6	7	8
1	0.000							
2	0.945	0.000						
3	0.885	0.141	0.000					
4	0.529	0.885	0.945	0.000				
5	1.915	1.736	1.915	1.446	0.000			
6	1.476	1.915	1.915	0.945	1.300	0.000		
7	1.736	2.437	2.462	1.476	2.238	0.945	0.000	
8	3.140	3.962	3.962	3.110	3.962	2.663	1.732	0.000

Hierna komen drie grafieken die inzicht geven in de goodness of *fit*. In de eerste grafiek zijn de afstanden-in-de-configuratie uitgezet tegen de trans-

PLOT OF TRANSFORMATION:
DISPARITIES (VERTICAL) VS OBSERVATIONS (HORIZONTAL)

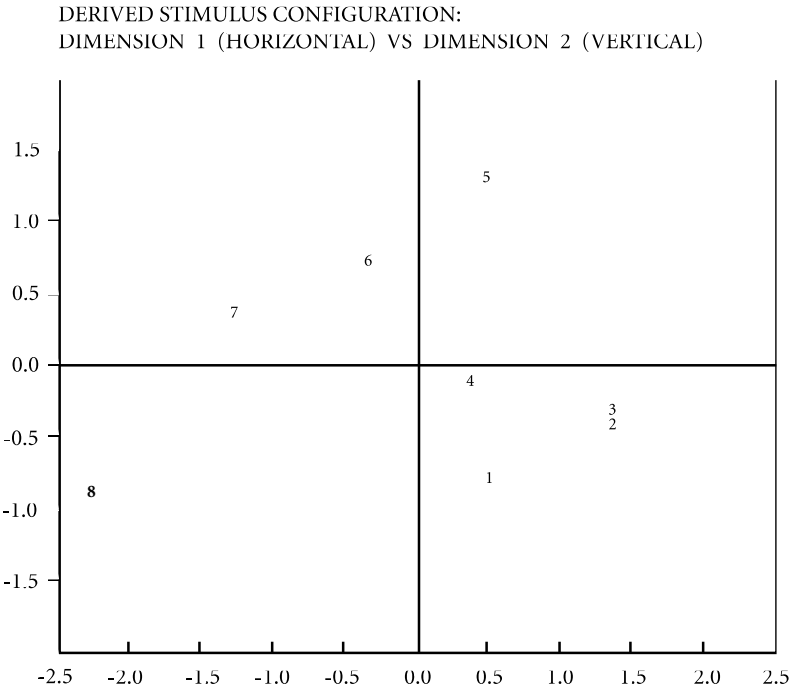


formaties. In het ideale geval liggen de punten (NB: in deze grafieken corresponderen de punten met stimulusparen) op een rechte lijn. Hoe meer afwijkingen van een rechte lijn, hoe hoger de (s)stress en hoe lager de RSQ.

In bovenstaande grafiek liggen de punten inderdaad dicht in de buurt van een rechte lijn en zijn daar gelijkmatig over verdeeld. In andere gevallen is dat niet altijd zo.Vaak ziet men een puntenwolk die bestaat uit één of enkele punten rechts bovenaan (stimulusparen met grote afstanden en grote pseudo-afstanden) en de overige punten in een kluitje links onderaan in de grafiek. Door het grote verschil tussen de punten rechtsboven en de punten linksonder is er dan toch vaak sprake van een hoge RSQ.

De tweede grafiek uit deze reeks is het zogenaamde Shepard-diagram, waarin de afstanden-in-de-configuratie zijn uitgezet tegen de observaties. Deze curve zegt enerzijds iets over de kwaliteit van de oplossing (de *fit*): in het ideale geval liggen de punten op een monotoon stijgende curve. Anderzijds zegt de vorm van de curve iets over de specifieke relatie tussen de geobserveerde nabijheden en de afstanden van de oplossing. In dit voorbeeld vormen afstanden en observaties nagenoeg een rechte lijn.

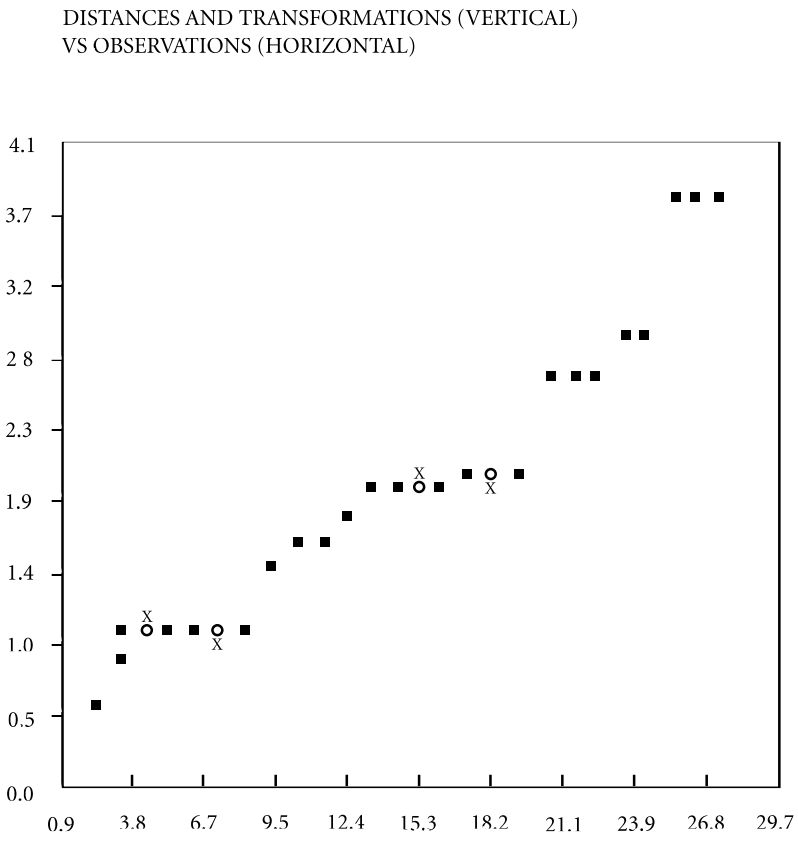
Dat moet ook eigenlijk wel, omdat onze nabijheidsdata uit echte, in kilo-



meters uitgedrukte afstanden bestonden. Als de ALSCAL-oplossing inderdaad op de kaart van Nederland lijkt, dan moet er wel een rechtlijnige relatie tussen observaties en afstanden in de configuratie bestaan. De vorm van deze curve laat dus zien dat we eigenlijk net zo goed een metrische analyse hadden kunnen doen, waarbij we ervan uit hadden kunnen gaan dat onze nabijheidsgegevens op rationiveau gemeten afstanden zijn. In ALSCAL kan men zo'n analyse krijgen door het commando /LEVEL = RATIO te geven in plaats van /LEVEL = ORDINAL. Aan het einde van dit blok zullen we enkele resultaten laten zien van een ordinale ALSCAL-analyse, waarbij niet de afstanden in kilometers, maar de rangnummers van die afstanden als nabijheidsdata zijn ingevoerd.
De derde grafiek laat zien hoe de observaties getransformeerd worden. Dit is dus de weergave van de optimale relatie tussen de afstanden-in-de-oplossing en de observaties. Ook hier zien we weer een nagenoeg rechte lijn. In het ordinale geval zien we doorgaans een uitgesproken trapvormige curve. Idealiter heeft zo'n curve slechts weinig brede plateaus en weinig extreem hoge treden.

Niet-metrische analyse op de rangnummers van de afstanden

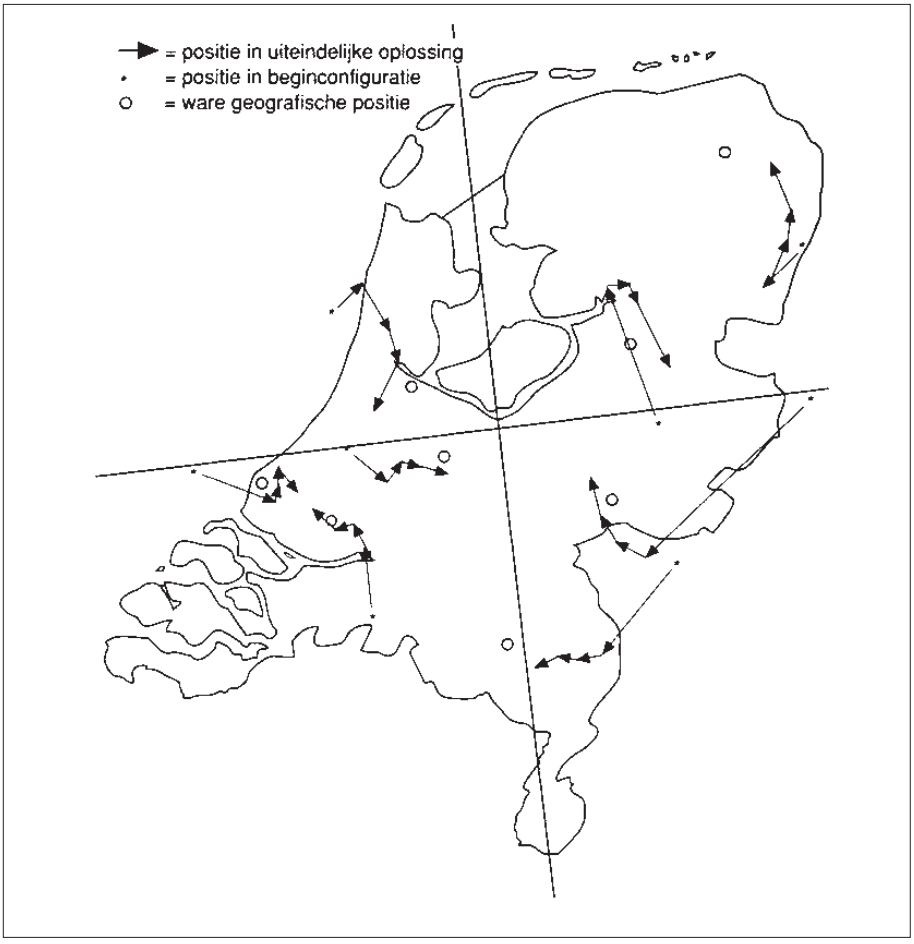
Hoewel bovenstaande, niet-metrische ALSCAL-analyse een goed gelijkende



kaart van Nederland oplevert, kan men zich afvragen of het goede resultaat toch niet vooral te danken is aan het feit dat de ingevoerde data echte afstanden waren. Om deze vraag te beantwoorden is dezelfde ALSCAL-analyse opnieuw uitgevoerd, maar nu met de *rangorde* van de afstanden als ingevoerde data. Daartoe zijn de echte afstanden-in-kilometers omgezet in rangnummers: de kleinste afstand (21 kilometer; tussen Rotterdam en Den Haag) krijgt rangnummer 1, de grootste afstand (253 kilometer; tussen Den Haag en Groningen) krijgt rangnummer 28. Hieronder zijn de ingevoerde rangnummers en de goodness-of-*fit*-maten afgedrukt, gevolgd door een afbeelding van de configuratie. Het blijkt dat ALSCAL na 38 iteraties een nagenoeg perfecte oplossing met *S-stress* = .009, RSQ = .999 en Kruskals $Stress_1$ = .012 oplevert.

Raw (unscaled) data for subject 1

	1	2	3	4	5	6	7	8
1	0.000							
2	7.000	0.000						
3	3.000	1.000	0.000					
4	2.000	3.000	4.000	0.000				
5	17.000	13.000	18.000	9.000	0.000			
6	11.000	16.000	15.000	5.000	8.000	0.000		



7	14.000	20.000	21.000	10.000	19.000	6.000	0.000	
8	24.000	25.000	27.000	23.000	26.000	22.000	12.000	0.000

ITERATION HISTORY FOR THE 2 DIMENSIONAL SOLUTION(IN SQUARED DISTANCES)

```
YOUNGS S-STRESS FORMULA 1 IS USED.

  ITERATION       S-STRESS      IMPROVEMENT

       1          0.08258
       2          0.04922        0.03336
       3          0.03694        0.01228
       .            .....          .....
       .            .....          .....
      37          0.00906        0.00005
      38          0.00902        0.00005

ITERATIONS STOPPED BECAUSE
S-STRESS IMPROVEMENT LESS THAN 0.000050
STRESS = 0.012 RSQ = 0.999
```

Het Shepard-diagram, dat hieronder is afgebeeld, laat zien dat alle punten, op vier na, op een monotoon stijgende curve liggen, dat wil zeggen, op de curve van de transformaties uitgezet tegen de observaties. Deze curve ziet eruit als een regelmatig stijgende trap met zeven, niet al te brede treden. Ook dit is een indicatie dat we een goede oplossing gevonden hebben.

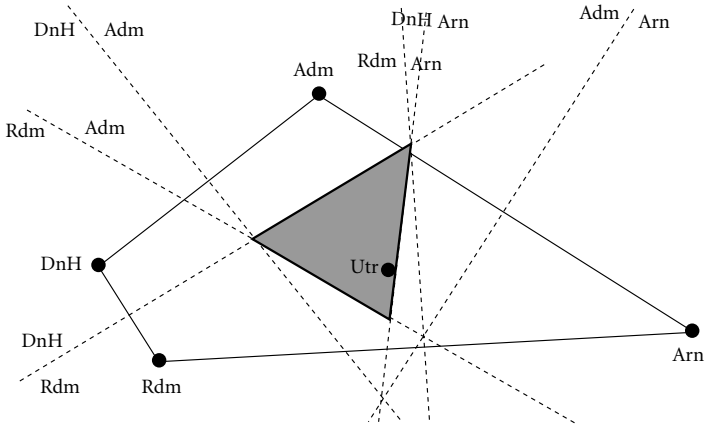
Een afbeelding van het iteratieve proces

Om een indruk te geven van de manier waarop het algoritme al itererend naar een oplossing toegaat, is in Figuur 6.5 de kaart van Nederland weergegeven met daarin de werkelijke geografische posities van de acht steden uit ons voorbeeld. In deze figuur is ook de initiële configuratie afgebeeld waarmee wij een ALSCAL-analyse hebben laten beginnen. Daarnaast toont Figuur 6.5 hoe de punten voor de steden zich in drie iteraties verplaatst hebben. We zien hier dat de oplossing na drie iteraties al een behoorlijk goede kaart van Nederland oplevert; de Kruskals $Stress_1$ van deze oplossing is .073. Het blijkt dat er na 34 iteraties een vrijwel perfecte oplossing verkregen wordt met $Stress_1 = .015$. In deze oplossing vallen de steden in grote trekken samen met hun echte geografische posities; ze doen dat
echter niet volledig. Dit komt doordat onze rangordegegevens iets meer vrijheden toestaan om de steden af te beelden; in plaats van exact gedefinieerde punten waar de steden moeten liggen is er voor iedere stad een (meestal klein) gebiedje waarin de stad afgebeeld mag worden, zonder dat de stress daardoor verandert. Hier komen we in de volgende paragraaf op terug.

Figuur 6.5 Beginconfiguratie, posities na 1, 2, 3 en 34 iteraties en de ware geografische posities van acht Nederlandse steden

Metrische analyse met ALSCAL

Door de commandoregel /LEVEL=ORDINAL voert ALSCAL een niet-metrische analyse op de ingelezen (dis)similarities uit. Als de data,
zoals hier, uit echte afstanden bestaan, ligt het voor de hand om (ook)



een metrische MDS-analyse uit te voeren. Daarbij kunnen we, zoals in Hoofdstuk 3 is uitgelegd, verschillende situaties onderscheiden:

(a) de observaties zijn recht evenredig met echte afstanden, het zijn afstanden op rationiveau, (b) de observaties zijn een lineaire functie van de echte afstanden, het zijn afstanden op intervalniveau waardoor het nodig is een additieve constante te schatten, (c) de observaties zijn evenredig met een curvilineaire functie van echte afstanden, en (d) de observaties zijn, op een additieve constante na, evenredig met een curvilineaire functie van de echte afstanden. De eerste twee situaties kunnen geanalyseerd worden door respectievelijk de SPSS-opdrachten /LEVEL=RATIO en /LEVEL=
INTERVAL te geven of – wat hetzelfde is – /LEVEL=RATIO(1) en /LEVEL=INTERVAL(1). Het cijfer 1 tussen haakjes betekent dat er in deze gevallen *polynomische functies van de graad 1* (dus rechte lijnen!) gezocht worden om de observaties te transformeren. In het derde en het vierde geval kan men kromlijnige transformatiecurves specificeren door tussen de haakjes een getal groter dan 1 in te vullen. Bij /LEVEL=
RATIO(2) zoekt ALSCAL een parabool die door het nulpunt gaat. Bij /LEVEL=INTERVAL(2) zoekt ALSCAL een parabool die, ten gevolge van een additieve constante ongelijk nul, niet noodzakelijk door het nulpunt hoeft te gaan. De maximumwaarde die men tussen de haakjes kan invullen, is vier. Hoewel een metrische oplossing in het algemeen hogere (s)stresswaarden oplevert, kan het soms zinvol zijn om ordinale data met een polynomische transformatiefunctie te analyseren. Dit is het geval als een niet-metrische oplossing niet of moeilijk te interpreteren is, met name als er sprake is van een zogenaamde gedegenereerde oplossing. Dit probleem wordt behandeld in Hoofdstuk 7.

De behandeling van ties en similarities in plaats van dissimilarities

Bij ALSCAL krijgen *ties* in de observaties automatisch gelijke transformatiewaarden, tenzij men opgeeft /LEVEL=ORDINAL (UNTIE). In dat geval mogen gelijke observaties verschillende pseudo-afstanden krijgen.
ALSCAL neemt aan dat de observaties uit dissimilarities bestaan. Heeft men *ordinale similarities*, dan moet men opgeven /LEVEL=ORDINAL (SIMILAR). Heeft men metrische similarities, dan moet men de data eerst zelf hercoderen tot dissimilarities voordat men een ALSCAL-analyse kan doen.

Metrische informatie in niet-metrische data

Aan het slot van dit hoofdstuk is het interessant om even stil te staan bij de vraag hoe het komt dat niet-metrische MDS-algoritmen inderdaad in staat zijn de onderliggende, metrische structuur van een verzameling punten te ontdekken, zelfs al bestaan de gegevens alleen maar uit de rangnummers van afstanden en laat men het algoritme starten met een foute beginconfiguratie. Dat dit kan, lijkt nogal wonderbaarlijk. Het antwoord op deze vraag is te vinden in het door Shepard (1962b) en Coombs (1964) naar voren gebrachte feit dat ook rangordegegevens impliciet al veel metrische informatie bevatten. Bijvoorbeeld: als we weten dat Utrecht dichter bij Amsterdam dan bij Rotterdam ligt, dan weten we dat Utrecht aan de Amsterdamse kant ligt van de middelloodlijn die de rechte tussen Amsterdam en Rotterdam in tweeën deelt. Weten we bovendien dat Utrecht dichter bij Rotterdam dan bij Den Haag ligt, dan weten we dat Utrecht aan de Rotterdamse kant ligt van de middelloodlijn van Rotterdam-Den Haag. Uit de rangnummers van de afstanden die in Tabel 2.1 staan, weten we nu ook dat Utrecht aan de Amsterdamse kant ligt van de middelloodlijn van Amsterdam-Arnhem, aan de Rotterdamse kant van de middelloodlijn van Rotterdam-Arnhem en aan de Haagse kant van de middelloodlijn van Den Haag-Arnhem. In Figuur 6.6 zijn de werkelijke geografische posities van Amsterdam, Rotterdam, Den Haag en Arnhem afgebeeld. Tussen deze steden zijn verbindingslijnen getrokken en zijn de bijbehorende middelloodlijnen getekend. We zien nu dat er een betrekkelijk klein gebiedje overblijft waarin Utrecht kan liggen. De plaats van Utrecht is dus binnen zekere grenzen vrij nauwkeurig te bepalen, echter *niet exact*: alle punten in het betreffende gebiedje voldoen namelijk aan de uit de rangorden afgeleide relaties. In een MDS-programma worden dus coördinaten voor Utrecht gezocht die in dit gebiedje liggen. Waar precies is niet belangrijk, omdat alle punten in dit gebiedje een even grote stresswaarde opleveren. Het hangt dus onder andere van de beginconfiguratie af waar Utrecht uiteindelijk terechtkomt.

Figuur 6.6 Gebied voor Utrecht, afgebakend door de middelloodlijnen van de verbindingen tussen Amsterdam, Rotterdam, Den Haag en Arnhem

Het zal duidelijk zijn dat het gebiedje voor Utrecht in principe steeds kleiner wordt naarmate men meer andere steden in de afbakening betrekt. Op basis van de voorlopige posities van de andere steden worden er coördinaten binnen een gebiedje voor Utrecht gevonden. Die coördinaten worden in een volgend onderdeel van het iteratieve proces weer gebruikt om gebiedjes voor de andere steden af te bakenen. Uiteindelijk levert dit een stabiele constellatie van gebiedjes op, waarin punten liggen die slechts weinig bewegingsvrijheid hebben.

'Recovery'

De kwestie of en onder welke omstandigheden niet-metrische MDS in staat is de metrische structuur van een aantal stimuli te ontdekken – Young (1970) noemt dit de *recovery of metric information* – is onderzocht door middel van