Janne Cools

Verslag

Cyclische matrices

Stelling 1

Het is logisch dat de som van 2 cyclische matrices opnieuw een cyclische matrix is. Je telt telkens elementen van dezelfde rij en kolom met elkaar op, waardoor alle elementen van de ene matrix, die dezelfde c_i-waarde hebben, met dezelfde waarde van de andere matrix wordt opgeteld, waardoor ze gelijk blijven. Hieronder staat het getoond.

$$A = \begin{pmatrix} a0 & a1 & a2 & \cdots & an-1 \\ an-1 & a0 & a1 & \cdots & an-2 \\ an-2 & an-1 & a0 & \cdots & an-3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a1 & a2 & a3 & \cdots & a0 \end{pmatrix} B = \begin{pmatrix} b0 & b1 & b2 & \cdots & bn-1 \\ bn-1 & b0 & b1 & \cdots & bn-2 \\ bn-2 & bn-1 & b0 & \cdots & bn-3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b1 & b2 & b3 & \cdots & b0 \end{pmatrix}$$

$$A + B = \begin{pmatrix} a0+b0 & a1+b1 & a2+b2 & \cdots & an-1+bn-1 \\ an-1+bn-1 & a0+b0 & a1+b1 & \cdots & an-2+bn-2 \\ an-2+bn-2 & an-1+bn-1 & a0+b0 & \cdots & an-3+bn-3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a1+b1 & a2+b2 & a3+b3 & \cdots & a0+b0 \end{pmatrix}$$

Bij het product van twee matrices A en B (voorgesteld zoals hierboven) wordt het element van de nieuwe matrix C op rij i en kolom j als volgt berekend: $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

Als je c_{11} berekent, is dit gelijk aan $a_0*b_0+a_1*b_{n-1}+a_2*b_{n-2}+\ldots+a_{n-1}*b_1$.

Bij het berekenen van c_{22} bekom je $a_{n-1}*b_1+a_0*b_0+a_{n-1}*b_1+\ldots+a_2*b_{n-2}$.

Deze beide waarden (die in matrix A overeenkomen met a_0) zijn gelijk, wat ook geldt voor alle andere elementen in de matrix C. Dat betekent dat de elementen in C die op dezelfde plaats staan zoals bijvoorbeeld het element a_0, ook allemaal gelijk zijn. Hiermee is dus bewezen dat C, het product van twee cyclische matrices, opnieuw een cyclische matrix is.

Aangezien een cyclische matrix inverteerbaar is, geldt dat $CC^{-1} = \mathrm{I}$ waarbij de eenheidsmatrix ook een cyclische matrix is. Het product van een cyclische en een niet-cyclische matrix zou als resultaat een niet-cyclische matrix geven. Daarom moet de inverse van een cyclische matrix zelf ook cyclisch zijn.

Implementatie Vector-object

Voor de implementatie van het Vector-object heb ik een klasse gemaakt die de lijst met de elementen van de vector bijhoudt. Vervolgens heb ik verschillende methodes toegevoegd die het later gebruik van de klasse eenvoudiger maken (zoals een element toevoegen of wijzigen). Vervolgens heb ik methodes toegevoegd voor het opvragen van de lijst zelf, de lengte ervan, een element op een specifieke index en de som van alle elementen (wat gebruikt wordt bij de

vermenigvuldiging van twee vectoren). Daarnaast kan je een vector schalen, i.e. alle elementen ervan vermenigvuldigen met hetzelfde complex getal, en 2 vectoren met elkaar vermenigvuldigen, bij elkaar optellen en aftrekken.
Ten slotte heb ik nog een functie geïmplementeerd om de vector om te zetten naar een string om op de commandolijn een vector duidelijk te kunnen voorstellen.
Om te controleren of mijn klasse volledig correct werkt, heb ik testen toegevoegd in een aparte klasse VectorTesten. Bij elke test laat ik de klasse een resultaat berekenen en vergelijk ik die met mijn eigen resultaat. Aangezien het makkelijke voorbeelden zijn, ben ik zeker dat mijn met de hand berekende resultaten juist zijn. Al mijn testen slagen dus weet ik zeker dat elke methode het juiste resultaat geeft.

Implementatie CirculantMatrix-object
De circulaire matrix houdt een lijst van Vector-objecten bij. Elk Vector-object stelt een rij van de matrix voor. Bij het aanmaken van de matrix wordt ofwel een vector meegegeven, die de eerste rij van de matrix wordt, ofwel meerdere elementen die dan in een vector geplaatst worden en de eerste rij vormen. Op basis van de verkregen vector worden de volgende rijen gemaakt aan de hand van permutaties. Ik heb methodes toegevoegd om een vector (rij) op te vragen of een element op een specifieke rij en kolom. Daarnaast kan je matrices bij elkaar optellen, van elkaar aftrekken en vermenigvuldigen met een andere matrix of een vector. Bovendien zijn er nog een aantal extra methodes in de klasse, maar die komen later aan bod bij de delen waar ze gebruikt worden. Voorbeelden hiervan zijn de Fourier matrix, de eigenwaarden en de macht van de matrix.

Deze methodes heb ik getest in de klasse CirculantMatrixTesten. Om het optellen, aftrekken en vermenigvuldigen van matrices te testen, heb ik de eigenschap van cyclische matrices gebruikt. Ik heb namelijk 2 matrices gemaakt en het verschil ervan laten berekenen. Vervolgens heb ik zelf een matrix laten construeren door enkel de eerste rij door te geven, aangezien de som, het verschil en het product van 2 circulaire matrices opnieuw een circulaire matrix is. De rij dat ik doorgeef is de eerste rij van de matrix die de som (resp. verschil of product) zou moeten zijn van de matrices die ik heb doorgegeven aan de functies.
Om de methode voor het product van een circulaire matrix met een vector te testen, heb ik de vermenigvuldiging zelf eerst op papier uitgewerkt en daarna op basis van mijn eigen resultaat de circulaire matrix gemaakt en deze laten vergelijken met het resultaat dat de methode in de klasse bekwam. Alle testen zijn geslaagd dus alle methodes kloppen. Voor de overige genoemde methodes zal ik mijn testen en de correctheid ervan later bespreken.

1.1 Eigenwaarden en eigenvectoren

Stelling 2 narekenen

Om stelling 2 na te rekenen, gebruik ik de definitie van eigenwaarden en eigenvectoren, namelijk: $Cv_j = \lambda_j v_j$ met C de cyclische matrix, v_j eigenvector horend bij eigenwaarde λ_j. Als ik dit verder uitwerk, bekom ik

$C\ \frac{1}{\sqrt{n}}(1 \quad w^j \quad w^{2j} \quad \dots \quad w^{(n-1)j})^T = \sum_{k=0}^{n} c_k w^{kj}\ \frac{1}{\sqrt{n}}(1 \quad w^j \quad w^{2j} \quad \dots \quad w^{(n-1)j})^T$.

Hierbij mag die $1/\sqrt{n}$ weggelaten worden in beide leden.

Stel dat je het element op rij r wilt berekenen. Dan krijg je in het linkerlid de vergelijking $(c_{n-r} \quad c_{n-r+1} \quad \cdots \quad c_{n-1} \quad c_0 \ c_1 \quad \cdots \quad c_{n-r-1})(1 \quad w^j \quad w^{2j} \quad \dots \quad w^{(n-1)j})^T$ en in het rechterlid $\left(\sum_{k=0}^{n} c_k w^{kj}\right) * w^{rj}$.

Het rechterlid kan herschreven worden als $\sum_{k=0}^{n} c_k w^{kj}\, w^{rj} = \sum_{k=0}^{n} c_k w^{(k+r)j} =$
$c_0 w^{rj} + c_1 w^{(r+1)j} + \dots + c_{n-1} c^{(n-1+r)j} =$
$(c_0 \quad c_1 \quad \cdots \quad c_{n-1})\,(w^{rj} \quad w^{(r+1)j} \quad \dots \quad w^{(n-1+r)j})^T$

Aangezien een optelling commutatief is, kan je de elementen in deze verkregen vector (na de vermenigvuldiging) van plaats wisselen. Hierdoor zal je dezelfde vectoren krijgen als in het linkerlid.

De elementen van de vector die je bekomt in het linkerlid, zijn dus gelijk aan de elementen van de vector die je bekomt in het rechterlid.

Algoritme voor macht

Om de macht van een matrix te berekenen, heb ik gebruik gemaakt van de Fourier matrix.

Voor het algoritme maak ik gebruik van het feit dat een circulaire matrix C geschreven kan worden als $Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1}$ met F de Fourier matrix. De Fourier matrix is een nxn-matrix waarbij het element op rij i en kolom j gelijk is aan $w^{(i-1)(j-1)}/\sqrt{n}$ met $w = e^{\frac{2\pi i}{n}}$.

Om dan de macht van de matrix te berekenen, kan je C^m herschrijven als $C^m = CCC..C = Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1}Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1} \dots Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1}$
$= Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)^m F^{-1} = Fdiag({\lambda_1}^m, {\lambda_2}^m, \dots, {\lambda_n}^m)F^{-1}$

In mijn CirculantMatrix-object heb ik een methode "power" toegevoegd en deze getest in de klasse CirculantMatrixTesten. Bij de methode "power" wordt een andere methode gebruikt, namelijk "multiply" waarin ik twee matrices met elkaar vermenigvuldig. Bij de test van het macht-algoritme heb ik van een matrix de tweede en derde macht berekend en vergeleken met de juiste oplossing. Hierdoor weet ik dat mijn implementatie van dit algoritme en de methode "multiply" correct zijn.

Twee matrices kunnen met elkaar vermenigvuldigd worden met een complexiteit van $O(n^2)$. Om de macht te berekenen, voer ik deze bewerking 2 keer uit, wat resulteert in een complexiteit van $O(2n^2)$. Het berekenen van de macht van de diagonale matrix kan uitgevoerd worden in een constante tijd en heeft dus geen invloed op de totale complexiteit. Hierdoor is de totale complexiteit $O(2n^2)$.

Algoritme voor Cx=b

Ook voor dit algoritme is de Fourier matrix een handig hulpmiddel.

De vergelijking Cx=b kan herschreven worden als: $Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1}x = b$, wat herleid kan worden naar $x = (Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)F^{-1})^{-1}b = Fdiag(\lambda_1, \lambda_2, \dots, \lambda_n)^{-1}F^{-1}b$.

Om de inverse van een diagonaalmatrix te berekenen, is het enkel nodig om de inverse waarden te bepalen. Bovendien is de inverse van de Fourier matrix dezelfde als zijn geconjugeerde getransponeerde matrix waardoor de vergelijking aangepast kan worden naar $x = Fdiag(1/\lambda_1, 1/\lambda_2, \dots, 1/\lambda_n)F^*b$ met F^* deze genoemde matrix. In mijn algoritme heb ik enkel de geconjugeerde matrix ervan berekend aangezien de Fourier matrix en zijn getransponeerde gelijk zijn.

Het rechterlid heb ik in mijn code associatief van rechts naar links uitgerekend.

Dit algoritme heb ik geïmplementeerd in mijn CirculantMatrix-klasse als methode "solve" waaraan je één argument meegeeft, namelijk de vector b. In deze methode gebruik ik andere geïmplementeerde methodes namelijk "getFourier", die de (eventueel inverse) Fourier matrix geeft en "getEigenvalues" die de eigenwaarden teruggeeft.

Om te controleren dat dit algoritme klopt, heb ik een test geschreven. In deze test maak ik een circulaire matrix c van 5x5 en twee kolomvectoren van elk 5 elementen (vectoren x_1 en x_2). Dan bereken ik b_1 en b_2 door de matrix te vermenigvuldigen met de kolomvectoren, $c*x_1=b_1$ en $c*x_2=b_2$ (deze berekening heb ik apart op papier gemaakt en is dus niet te zien in de test). Uiteindelijk voer ik de methode "solve" van de matrix c uit met als argument de vector b_1 (resp. b_2) en controleer ik dat de bekomen vector gelijk is aan x_1 (resp. x_2). Deze test staat in de klasse CirculantMatrixTesten en slaagt, waardoor ik weet dat zowel de methode solve correct is als de methodes getFourier en getEigenvalues.

Middens verbinden

Bewerking op cyclische matrix

Om dit probleem om te zetten naar een bewerking op een cyclische matrix, moet er eerst bepaald worden hoe de cyclische matrix opgebouwd zal zijn. Het is de bedoeling om van een reeks punten het midden te berekenen van elke 2 opeenvolgende punten. De vermenigvuldiging van de vector van punten met de cyclische matrix moet er dus voor zorgen dat een nieuwe vector ontstaat met het midden van alle opeenvolgende punten.

Wanneer je een vector vermenigvuldigt met een matrix, vermenigvuldig je de vector met elke kolom van die matrix. Dit betekent dat elke kolom van de cyclische matrix telkens een ander paar punten moet krijgen om daarvan het midden te berekenen.

Om dit te verwezenlijken, wordt mijn cyclische matrix gedefinieerd aan de hand van de n-dimensionale vector $(1/2 \quad 0 \quad \cdots \quad 0 \quad 1/2)$. Hierdoor ziet mijn cyclische matrix er zo uit: $\begin{pmatrix} 1/2 & 0 & \cdots & 0 & 1/2 \\ 1/2 & 1/2 & \cdots & \vdots & 0 \\ 0 & 1/2 & \cdots & \vdots & \vdots \\ \vdots & 0 & \cdots & 0 & \vdots \\ \vdots & \vdots & \cdots & 1/2 & 0 \\ 0 & 0 & \cdots & 1/2 & 1/2 \end{pmatrix}$.

Als inputwaarden moeten twee vectoren meegegeven worden, één met de x- en één met de y-coördinaten. Beide vectoren zullen dan apart met deze matrix vermenigvuldigd worden waaruit de nieuwe coördinaten ontstaan.

Deze implementatie heb ik getest in het bestand MiddensTesten.cpp. Hierin heb ik twee keer een groep punten bepaald en de nieuwe groep van punten berekend na één stap en na twee stappen. De resultaten van de geïmplementeerde klasse Middens kwamen overeen met mijn met de hand gevonden resultaten, waardoor ik zeker ben dat mijn berekeningen en mijn cyclische matrix correct zijn.

Complexiteit

Een matrix-vector vermenigvuldiging kan normaal gezien uitgevoerd worden met een complexiteit van O(m*n). Aangezien de lengte van een vector echter even groot is als de grootte van de matrix zal de complexiteit hier O(n²) zijn. Daarnaast wordt er tweemaal een vermenigvuldiging uitgevoerd, voor de vector met de x-coördinaten en voor de vector met de y-coördinaten. Ten slotte worden er m stappen berekend, wat betekent dat de dubbele matrix-vector vermenigvuldiging m keer wordt uitgevoerd. De totale complexiteit voor n punten en m stappen is dus O(m*2n²).

Grootste eigenwaarden

Om te eigenvectoren horend bij de twee grootste eigenwaarden te kunnen interpreteren, heb ik gebruik gemaakt van de formule $Q * D * Q^T$.

Stel dat je een 2x2-diagonaalmatrix D maakt met de 2 grootste eigenwaarden en een matrix Q met als kolommen de eigenvectoren die bij deze eigenwaarden horen. Als je dan deze matrices met elkaar vermenigvuldigt en dit resultaat vermenigvuldigt met de getransponeerde matrix van Q, dan krijg je een nieuwe

matrix M. Als je dan deze matrix M vermenigvuldigt met de inputvectoren, de x-coördinaten en y-coördinaten van de punten, dan krijg je als resultaat je nieuwe punten. Deze punten zijn ongeveer gelijk aan de punten die je krijgt wanneer je het op de normale manier berekent, dus met de bewerking op de cyclische matrix.
Dit betekent voor het limietgedrag dat je bij voldoende iteraties genoeg hebt aan de grootste eigenwaarden en hun eigenvectoren.

Wildwaterbaan

Om de vector $h^{(k)}$ te berekenen, kan de gegeven formule omgevormd worden als volgt: $h^{(k-1)} - 2h^{(k)} = \frac{\Delta t^2}{\Delta x^2} A h^{(k+1)} - h^{(k+1)} <=> h^{(k-1)} - 2h^{(k)} = (\frac{\Delta t^2}{\Delta x^2} A - I) h^{(k+1)} <=> (\frac{\Delta x^2}{\Delta t^2} A - I)^{-1}(h^{(k-1)} - 2h^{(k)}) = h^{(k+1)}$

In mij algoritme bereken ik eerst $\frac{\Delta x^2}{\Delta t^2} A$ en trek ik er daarna de eenheidsmatrix van af, waardoor de diagonaalelementen met 1 verminderd worden. Om de inverse van deze bekomen matrix a te berekenen, gebruik ik de fourier matrix. Aangezien $a = F diag(\lambda_1, \lambda_2, \dots, \lambda_n) F^{-1}$ is de inverse van a gelijk aan $a^{-1} = (F diag(\lambda_1, \lambda_2, \dots, \lambda_n) F^{-1})^{-1} = F diag(1/\lambda_1, 1/\lambda_2, \dots, 1/\lambda_n) F^{-1}$.
Deze inverse bereken ik dan ook direct in de constructor van mijn klasse en houdt ik bij in de variabele *a*.

De golven heb ik benaderd met verschillende waarden voor n, de beginwaarde en Δt. De waarde Δt heb ik laten variëren van 0.5 tot en met 2.5 met stappen van 0.5. Voor elk van deze waarden werden golven benaderd voor 6 en 8 punten (deze stellen de waarde n voor). Uiteindelijk heb ik ook wat geëxperimenteerd met de beginwaarde $h^{(-1)} = h^{(0)}$.
Voor al deze mogelijkheden heb ik telkens $h^{(1)}$ tot en met $h^{(3)}$ berekend waarbij alle waarden afgerond zijn op 2 cijfers na de komma.

Als eerste beginwaarde $h^{(0)}$ heb ik een vector gekozen waarvan de elementen alternerend 0 en 1 zijn. Bij n=4 bijvoorbeeld is $h^{(0)} = (1 \quad 0 \quad 1 \quad 0)$.
De verschillende hoogtes die dan op basis van de eerdergenoemde waarden n en Δt zijn berekend, zijn in de onderstaande tabel weergegeven.

n	**Δt**	$\mathbf{h^{(1)}}$	$\mathbf{h^{(2)}}$	$\mathbf{h^{(3)}}$
6	1	(0.52, 0.48, 0.52, 0.48, 0.52, 0.48)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
6	1.5	(0.53, 0.47, 0.53, 0.47, 0.53, 0.47)	(0.49, 0.51, 0.49, 0.51, 0.49, 0.51)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
6	2	(0.56, 0.44, 0.56, 0.44, 0.56, 0.44)	(0.46, 0.54, 0.46, 0.54, 0.46, 0.54)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
6	2.5	(0.58, 0.42, 0.58, 0.42, 0.58, 0.42)	(0.44, 0.56, 0.44, 0.56, 0.44, 0.56)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
6	3	(0.61, 0.39, 0.61, 0.39, 0.61, 0.39)	(0.44, 0.56, 0.44, 0.56, 0.44, 0.56)	(0.45, 0.55, 0.45, 0.55, 0.45, 0.55)
8	1	(0.52, 0.48, 0.52, 0.48, 0.52, 0.48, 0.52, 0.48)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
8	1.5	(0.53, 0.47, 0.53, 0.47, 0.53, 0.47, 0.53, 0.47)	(0.49, 0.51, 0.49, 0.51, 0.49, 0.51, 0.49, 0.51)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50)

8	2	(0.56, 0.44, 0.56, 0.44, 0.56, 0.44, 0.56, 0.44)	(0.46, 0.54, 0.46, 0.54, 0.46, 0.54, 0.46, 0.54)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
8	2.5	(0.58, 0.42, 0.58, 0.42, 0.58, 0.42, 0.58, 0.42)	(0.44, 0.56, 0.44, 0.56, 0.44, 0.56, 0.44, 0.56)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50)
8	3	(0.61, 0.39, 0.61, 0.39, 0.61, 0.39, 0.61, 0.39)	(0.44, 0.56, 0.44, 0.56, 0.44, 0.56, 0.44, 0.56)	(0.45, 0.55, 0.45, 0.55, 0.45, 0.55, 0.45, 0.55)

Op basis van deze tabel kan je een conclusie maken over het effect van de gebruikte beginwaarde $h^{(0)}$. Aangezien de elementen hiervan alternerend 0 en 1 zijn, zullen de vectoren van de volgende hoogtes ($h^{(1)}$ tot en met $h^{(3)}$) ook telkens alternerend 2 waarden bevatten. Die waarden naderen uiteindelijk ½, het gemiddelde van de 2 beginelementen. Daarnaast wisselen de elementen van een vector $h^{(i)}$ telkens tussen een waarde hoger en een waarde lager dan dat gemiddelde.

Vervolgens heb ik als beginwaarde $h^{(0)}$ een vector gekozen waarvan de elementen alterneren tussen 1 en stijgende getallen. Dit resulteer in de vector $(1 \quad 2 \quad 1 \quad 3 \quad 1 \quad 4 \quad 1 \quad 5 \quad \cdots)$.
De verschillende hoogtes die dan op basis van de eerdergenoemde waarden n en Δt zijn berekend, zijn in de onderstaande tabel weergegeven.

n	**Δt**	$\mathbf{h^{(1)}}$	$\mathbf{h^{(2)}}$	$\mathbf{h^{(3)}}$
6	1	(1.97, 1.94, 1.92, 2.03, 2.02, 2.12)	(2.00, 2.02, 2.02, 2.00, 1.98, 1.97)	(2.00, 2.00, 2.00, 2.00, 2.00, 2.00)
6	1.5	(1.93, 1.89, 1.85, 2.07, 2.02, 2.25)	(2.01, 2.07, 2.07, 1.99, 1.96, 1.91)	(2.00, 2.00, 2.00, 2.00, 2.00, 2.00)
6	2	(1.89, 1.84, 1.77, 2.11, 2.00, 2.38)	(2.09, 2.01, 2.08, 1.91, 2.10, 1.82)	(2.00, 2.07, 2.06, 2.00, 1.93, 1.93)
6	2.5	(1.83, 1.81, 1.70, 2.17, 1.97, 2.53)	(2.11, 1.95, 2.05, 1.89, 2.17, 1.83)	(2.00, 2.13, 2.12, 2.00, 1.89, 1.86)
6	3	(1.78, 1.78, 1.63, 2.22, 1.92, 2.66)	(2.12, 1.89, 2.01, 1.88, 2.24, 1.87)	(2.11, 2.08, 2.14, 1.89, 2.07, 1.70)
8	1	(2.21, 2.13, 2.08, 2.20, 2.21, 2.38, 2.34, 2.45)	(2.25, 2.31, 2.32, 2.29, 2.25, 2.20, 2.18, 2.19)	(2.25, 2.24, 2.24, 2.24, 2.25, 2.26, 2.26, 2.26)
8	1.5	(2.17, 2.03, 1.95, 2.17,	(2.27, 2.38, 2.41, 2.33,	(2.25, 2.26, 2.26, 2.26,

		2.17, 2.50, 2.38, 2.64)	2.27, 2.14, 2.12, 2.09)	2.25, 2.24, 2.23, 2.25)
8	2	(2.11, 1.95, 1.83, 2.17, 2.11, 2.61, 2.38, 2.84)	(2.36, 2.24, 2.27, 2.11, 2.36, 2.17, 2.45, 2.04)	(2.25, 2.40, 2.44, 2.38, 2.25, 2.13, 2.05, 2.11)
8	2.5	(2.04, 1.89, 1.73, 2.20, 2.04, 2.72, 2.35, 3.03)	(2.39, 2.14, 2.18, 2.03, 2.39, 2.20, 2.59, 2.08)	(2.25, 2.46, 2.49, 2.39, 2.25, 2.10, 2.02, 2.03)
8	3	(1.97, 1.84, 1.65, 2.24, 1.97, 2.82, 2.29, 3.22)	(2.40, 2.04, 2.10, 1.96, 2.40, 2.23, 2.71, 2.15)	(2.38, 2.33, 2.34, 2.11, 2.38, 2.13, 2.43, 1.90)

Net zoals bij het vorig voorbeeld, kan je uit deze tabel afleiden dat het verschil tussen de hoogtes telkens verkleint. Daarnaast kan je zien dat, zoals bij echte golven, de hoogte op een positie telkens wisselt tussen een waarde hoger en lager dan het gemiddelde van alle hoogtes.