Signaalafhankelijke Kernel-functies voor de berekening van tijd-frequentie distributies

Timmerman, P.H.

Award date:
1994

7176

TECHNISCHE UNIVERSITEIT EINDHOVEN

FACULTEIT ELEKTROTECHNIEK

VAKGROEP Signaalverwerking

esp SIGNAL PROCESSING

Signaalafhankelijke kernel-functies voor de berekening van tijd-frequentie distributies

door

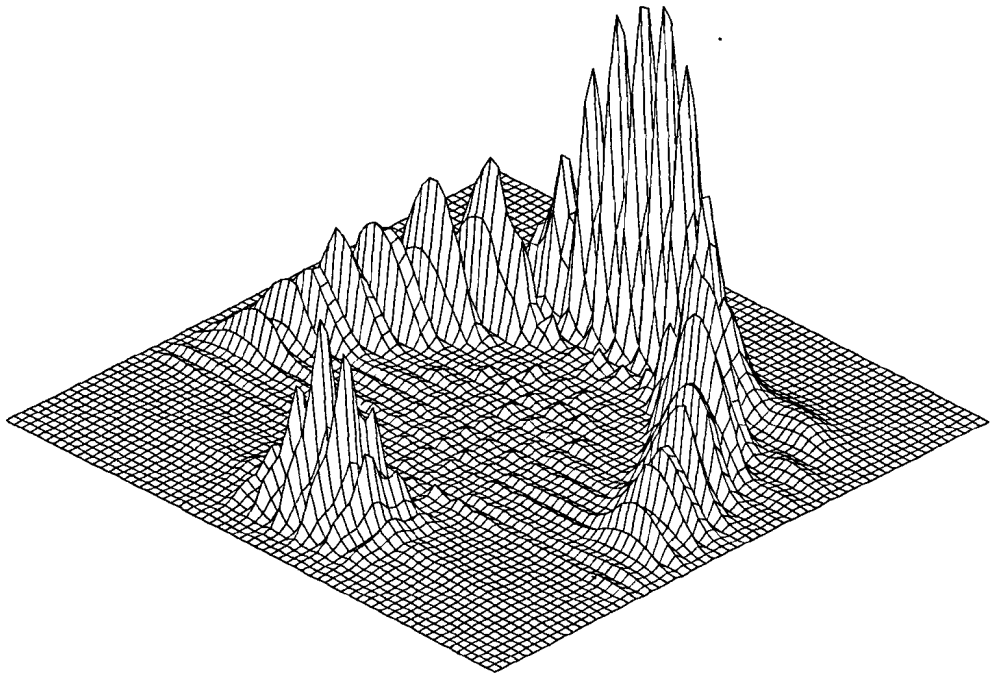
P.H. Timmerman

ESP-10-94

Verslag van een afstudeeronderzoek, verricht in de vakgroep ESP, onder leiding van dr.ir. M.J. Bastiaans, in de periode januari 1994 - augustus 1994.

Eindhoven, 23 augustus 1994.

Signaalafhankelijke kernel-functies voor de berekening van Tijd-Frequentie Distributies



Auteur: P.H.Timmerman
Begeleider: dr.ir.M.J.Bastiaans
Vakgroep: ESP

Datum: augustus 1994

Inhoudsopgave

Voorwoord

Dit verslag is tot stand gekomen op de vakgroep ESP van de Technische Universiteit Eindhoven. Het is het resultaat van zes maanden afstudeerwerk. Bij de totstandkoming ervan heb ik gebruik gemaakt van de kennis die ik tijdens mijn stageperiode aan dezelfde vakgroep heb opgedaan. Ik ben dank verschuldigd aan Martin Bastiaans voor zijn begeleiding van zowel de stage als het afstuderen en aan Professor H.J.Butterweck voor zijn belangstelling en kritische vragen.

Samenvatting

Dit afstudeerverslag presenteert een methode voor de berekening van Tijd-Frequentie Distributies (TFD's) met zoveel mogelijk gunstige eigenschappen. Dit met het doel om te komen tot een TFD met een fysisch zinvolle interpretatie. De belangrijkste eigenschappen die in het oog gehouden worden zijn: positiviteit, het voldoen aan de tijd-marginaal, het voldoen aan de frequentie-marginaal en minimale interferentie. Aan een deel van de eisen wordt voldaan door gebruik te maken van een signaalafhankelijke kernel-functie. Deze kernel-functie wordt vervolgens gebruikt voor de constructie van een TFD. De constructie verloopt analoog aan de manier waarop TFD's uit de Cohen-klasse geconstrueerd worden. Door het signaalafhankelijk zijn van de kernel-functie verkrijgt men een distributie die voor een brede klasse van signalen goede resultaten levert. Het resultaat wordt een Optimale-Kernel TFD (OKTFD) genoemd. De andere eisen waar we de TFD aan willen laten voldoen worden geforceerd met behulp van een Minimum-Cross-Entropy (MCE) methode. De OKTFD en de tijd- en frequentie-marginalen dienen hierbij als uitgangspunten.

1 Inleiding

Dit afstudeerverslag geeft een beschrijving van een methode om Tijd-Frequentie-Distributies (TFD's) te berekenen. Hierbij wordt getracht een TFD te verkrijgen met zoveel mogelijk gunstige eigenschappen, met als doel een representatie te krijgen die aangeeft hoe de energie van een signaal is verdeeld over het tijd-frequentie vlak en die een fysisch betekenisvolle interpretatie kent. Om hiertoe te komen wordt gebruik gemaakt van een bilineaire klasse van TFD's, ook wel de Cohen-klasse genoemd. Iedere TFD uit deze klasse wordt gekarakteriseerd door een kernel-functie die bij conventionele methoden altijd signaalonafhankelijk geweest is. Er is echter gebleken dat er geen vaste kernel-functie bestaat die voldoet voor meerdere soorten signalen. Voor een verschillende klasse van signalen dienen verschillende kernel-functies ontworpen te worden. Als we een methode willen hebben die voor een brede klasse van signalen goed functioneert, zullen we de kernel-functie signaalafhankelijk moeten maken. Dit verslag reikt hier een methode voor aan.

Ten grondslag aan de problemen van de TFD's uit de Cohen-klasse ligt het feit dat iedere bilineaire distributie interferentie-termen heeft die interpretatie moeilijk of onmogelijk maken. Het dilemma is nu dat enerzijds deze interferentie-termen de interpretatie verstoren, maar anderzijds zorgen voor gunstige mathematische eigenschappen. In dit verslag wordt de volgende oplossing voor dit probleem gepresenteerd:

Stap A: Verwijder eerst zoveel mogelijk interferentie-termen door gebruik te maken van een signaalafhankelijke kernel-functie.

Stap B: Bewerk dit resultaat zodanig dat de nuttige informatie uit de verwijderde interferentie-termen weer in de TFD voorkomt, maar nu op een manier die niet storend is voor de interpretatie.

Deze laatste stap wordt uitgevoerd met behulp van een Minimum-Cross-Entropy (MCE) methode.

De indeling van het verslag is nu als volgt:

Gestart wordt met een bespreking over tijd-frequentie analyse en TFD's. Vervolgens wordt iets meer verteld over de Cohen-klasse van TFD's en hoe deze klasse beschreven kan worden door uit te gaan van kernel-functies in verschillende domeinen. De beperkingen van de TFD's uit de Cohen-klasse met vaste kernel-functie komen aan de orde, wat zal leiden tot de conclusie dat signaalafhankelijke kernel-functies noodzakelijk zijn (hoofdstuk 2). Hoe een dergelijke kernel-functie, of kortweg kernel, geconstrueerd kan worden is het onderwerp van hoofdstuk 3. Als eenmaal een TFD via een signaalafhankelijke kernel verkregen is, is stap A voltooid en moet stap B nog uitgevoerd worden. Dit wordt aan de hand van de MCE-

methode toegelicht in hoofdstuk 4. In beide hoofdstukken 3 en 4 zullen van een aantal testsignalen de resultaten van de methoden getoond worden. Deze resultaten worden afgezet tegen de resultaten van TFD's uit de Cohen-klasse die gebruik maken van een vaste kernel. In de daarop volgende hoofdstukken 5 en 6 zal aangegeven worden hoe de stappen A en B in algoritmen uitgevoerd en geïmplementeerd worden. Afgesloten wordt met een beschrijving van de programmatuur en met conclusies en aanbevelingen.

2 Tijd-Frequentie Distributies

2.2 Tijd-Frequentie analyse

Een signaal kan op verschillende manieren aan een analyse onderworpen worden. De meest voorkomende manier is een analyse in het tijddomein. Willen we van een signaal s(t) de momentele energie weten, dan nemen we $|s(t)|^2$. $|s(t)|^2$ kan opgevat worden als een energiedichtheidsfunctie in het tijddomein en geeft aan hoe de energie van het signaal is verdeeld over de tijd. Of ook:

$|s(t)|^2$ = intensiteit per eenheid tijd op tijdstip t,

$|s(t)|^2 \Delta t$ = fractie energie in tijdinterval Δt op tijdstip t.

Naast een analyse in het tijddomein bestaat er de analyse in het frequentie domein. Een signaal wordt nu gerepresenteerd door S(f) met hierin f de frequentievariabele. $|S(f)|^2$ is nu de spectrale energiedichtheidsfunctie of energiespectrum van s(t). Of ook:

$|S(f)|^2$ = intensiteit per eenheid frequentie bij frequentie f,

$|S(f)|^2 \Delta f$ = fractie energie in frequentie-interval Δf bij frequentie f.

s(t) en S(f) zijn aan elkaar gerelateerd door middel van de Fourier-transform:

$$s(t) = \int_{-\infty}^{\infty} S(f)\, e^{j2\pi ft} df$$

$$S(f) = \int_{-\infty}^{\infty} s(t)\, e^{-j2\pi ft} dt$$

En volgens het theorema van Parseval geldt:

$$\int_{-\infty}^{\infty} |s(t)|^2 dt = \int_{-\infty}^{\infty} |S(f)|^2 df = E_s$$

waarbij E_s de energie van signaal s voorstelt.

Het energiespectrum geeft aan hoe de energie van het signaal over de frequenties is verdeeld. Wat het niet aangeeft is wanneer deze frequenties optraden. Wanneer de frequentie-inhoud over langere tijd constant blijft is dit geen bezwaar. Als echter deze frequentie-inhoud sterk varieert, kan het energiespectrum geen informatie leveren over het moment van optreden van bepaalde frequenties. Om deze informatie toch te krijgen moeten we naar een energiedichtheids-functie simultaan beschreven in een tijd én frequentie variabele. Dergelijke functies noemen we Tijd-Frequentie Distributies (TFD's). Voor een TFD P(t,f) hebben we:

P(t,f) = intensiteit op tijdstip t en frequentie f

en

P(t,f)$\triangle$t$\triangle$f = fractie energie in tijd-frequentie cel $\triangle$t$\triangle$f op tijdstip t en bij frequentie f.

Een (ideale) TFD kunnen we (formeel) vergelijken met een simultane kansdichtheids-functie in 2 variabelen. Net als deze functie behoort een ideale TFD positief te zijn en moet een TFD $P_s(t,f)$ voldoen aan:

$$\int_{-\infty}^{\infty} P_s(t,f)\, dt = |S(f)|^2 \qquad (1)$$

$$\int_{-\infty}^{\infty} P_s(t,f)\, df = |s(t)|^2 \qquad (2)$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P_s(t,f)\, dtdf = E_s \qquad (3)$$

(1) en (2) zijn de zogenaamde 'marginal constraints' of marginaal-voorwaarden van de TFD, vergelijkbaar met de marginale kansdichtheids-functies zoals die horen bij een simultane kansdichtheids-functie.

In het vervolg zullen we uitgaan van genormaliseerde signalen, zodanig dat $E_s = 1$. Ook in dit opzicht gaat de vergelijking met de kansdichtheden op.

Nu is het de vraag of er een dergelijke ideale TFD bestaat en of die TFD daadwerkelijk de correlaties tussen tijd- en frequentie-inhouden weergeeft. Het antwoord hierop is nog onbekend. Belangrijke stappen zijn gemaakt door het herkennen van een bepaalde klasse van TFD's die gemeenschappelijke eigenschappen hebben namelijk: de Cohen-klasse. Omdat de Cohen-klasse TFD's bevat die bilineair zijn zullen we daar eerst iets over zeggen.

2.2 Bilineaire Tijd-Frequentie Representaties

Iedere bilineaire Tijd-Frequentie Representatie (TFR) voldoet aan het kwadratische superpositie beginsel (zie [HlaBou92] en de referenties daarin):

$$x(t) = c_1 x_1(t) + c_2 x_2(t) \Rightarrow$$
$$P_x(t,f) = |c_1|^2 P_{x_1}(t,f) + |c_2|^2 P_{x_2}(t,f) + c_1 c_2^* P_{x_1,x_2}(t,f) + c_2 c_1^* P_{x_2,x_1}(t,f)$$

waarbij $P_{x1,x2}(t,f)$ de kruis TFR is van de 2 signalen $x_1(t)$ en $x_2(t)$. Generalisering van dit principe voor eèn signaal met N componenten:

$$x(t) = \sum_{k=1}^{N} c_k x_k(t)$$

levert de volgende regel op:

* Iedere signaalcomponent $c_k x_k(t)$ correspondeert met een autocomponent $|c_k|^2 P_{xk}(t,f)$ (signaal-term).
* Voor ieder paar van signaalcomponenten $c_k x_k(t)$ en $c_l x_l(t)$ (k<>l) bestaat er een kruiscomponent (interferentie-term):

$$c_k c_l^* P_{x_k,x_l}(t,f) + c_l c_k^* P_{x_l,x_k}(t,f) .$$

Een signaal met N componenten levert dus een TFR op met N signaal-termen en N(N-1)/2 interferentie-termen. Het zijn deze interferentie-termen die de interpretatie van bilineaire TFR's bemoeilijkt, mede omdat deze termen zich doorgaans bevinden op plaatsen in het (t,f)-vlak waar men geen energiebijdrage zou verwachten.

2.2.1 De Wigner Distributie en de Ambiguity functie

De Wigner Distributie (WD) is een bilineaire functie met een energetische interpretatie:

$$W_x(t,f) \triangleq \int_{-\infty}^{\infty} x(t+\tau/2)\, x^*(t-\tau/2)\, e^{-j2\pi f\tau} d\tau = \int_{-\infty}^{\infty} X(f+\nu/2)\, X^*(f-\nu/2)\, e^{j2\pi t\nu} d\nu \qquad (4)$$

De WD voldoet aan een groot aantal gunstige mathematische eigenschappen waaronder het voldoen aan (1), (2) en (3). De WD heeft een goede tijdresolutie én een goede frequentieresolutie maar is echter voor vrijwel alle signalen lokaal nega-

tief en heeft interferentie-termen op plaatsen in het (t,f)-vlak waar men geen energiebijdrage zou verwachten.

De Ambiguity functie (AF) is een bilineaire functie die geïnterpreteerd kan worden als een tijd-frequentie correlatie functie:

$$A_x(\nu,\tau) \triangleq \int_{-\infty}^{\infty} x(t+\tau/2)\, x^*(t-\tau/2)\, e^{-j2\pi\nu t} dt = \int_{-\infty}^{\infty} X(f+\nu/2)\, X^*(f-\nu/2)\, e^{j2\pi\tau f} df \qquad (5)$$

Belangrijke eigenschappen van deze correlatiefunctie zijn:

$$A_x(0,\tau) = \int_{-\infty}^{\infty} x(t+\tau)\, x^*(t)\, dt$$

$$A_x(\nu,0) = \int_{-\infty}^{\infty} X(f+\nu)\, X^*(f)\, df$$

en

$$|A_x(\tau,\nu)| \leq A_x(0,0) = \int_{-\infty}^{\infty} |x(t)|^2 dt$$

Tussen de WD en de AF bestaat een Fourier relatie:

$$W_x(t,f) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A_x(\nu,\tau)\, e^{-j2\pi(\tau f-\nu t)}\, d\nu\, d\tau \qquad (6)$$

Net als de WD zal de AF interferentie-termen bevatten. Deze interferentie-termen bevinden zich bij de AF echter altijd op afstand van de oorsprong, terwijl de signaal-termen zich rond de oorsprong concentreren. Dit is een gevolg van het feit dat de interferentie-termen oscillatorische eigenschappen hebben, waardoor ze na een Fourier transformatie verder van het nulpunt af komen te liggen dan de signaal-termen, die deze eigenschap niet hebben. De positie van de interferentie-termen in de AF kan gebruikt worden om ze te onderdrukken. Door een masker te leggen over de signaal-termen rond de oorsprong en vervolgens weer terug te transformeren naar het (t,f)-domein, kunnen interferentie-termen benadeeld worden ten opzichte van de signaal-termen. De exacte positie van de signaal-termen is echter niet bekend en verandert van oriëntatie als men een ander signaal neemt.

2.3 De Cohen-klasse van Tijd-Frequentie Distributies

De Cohen-klasse bestaat uit alle bilineaire TFR's die verschuivingsinvariant zijn in tijd- en frequentie-richting. Formeel:

$$x(t) = y(t-t_0)\, e^{j2\pi f_0 t} \Rightarrow P_x(t,f) = P_y(t-t_0, f-f_0)$$

Dus een verschuiving t_0 van y(t) in de tijd, levert dezelfde verschuiving van de distributie op. Modulering van het signaal y(t) met frequentie f_0, levert een verschuiving van f_0 in de frequentie richting op van de distributie.

Elementen uit de Cohen-klasse kunnen als volgt gekarakteriseerd worden:

$$P_x(t,f,\phi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(\mu+\tau/2)\, x^*(\mu-\tau/2)\, \phi(\nu,\tau)\, e^{-j2\pi(\tau f+\nu t+\nu\mu)}\, d\mu\, d\tau\, d\nu$$

wat men kan herschrijven tot:

$$P(t,f,\phi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \phi(\nu,\tau)\, A_x(\nu,\tau)\, e^{j2\pi(t\nu-f\tau)}\, d\tau\, d\nu \qquad (7)$$

$\phi(\nu,\tau)$ is een kernel-functie waarmee men een element uit de Cohen-klasse selecteert. Met de kernel-functie leggen we alle eigenschappen van de distributie vast. Iedere eigenschap van de distributie correspondeert met een bepaalde eigenschap van de kernel-functie. Voor het voldoen aan de marginaal-voorwaarden moet gelden: $\phi(\nu,0) = 1$ voor alle ν (tijd-marginaal) en, $\phi(0,\tau) = 1$ voor alle τ (frequentie-marginaal). Iedere distributie uit de Cohen-klasse die hieraan voldoet heeft als gevolg hiervan een kernel in het (ν,τ)-domein met een kruisvormig karakter.

Hieronder volgt een tabel met distributie-eigenschappen met corresponderende eisen die aan de kernel gesteld worden.

P0: distributie is positief.
Q0: $\phi(v,\tau)$ is een Ambiguity functie van een window w(t).
P1: distributie is reëel.
Q1: $\phi(v,\tau) = \phi^*(-v,-\tau)$.
P2: tijdverschuivingsinvariant.
Q2: $\phi(v,\tau)$ is onafhankelijk van t.
P3: frequentieverschuivingsinvariant.
Q3: $\phi(v,\tau)$ is onafhankelijk van f.
P4: voldoet aan tijd-marginal.
Q4: $\phi(v,0) = 1$ voor alle τ.
P5: voldoet aan frequentie-marginal.
Q5: $\phi(0,\tau) = 1$ voor alle v.
P6: momentane frequentie = eerste-orde frequentie-moment van distributie.
Q6: Q4 en partiële afgeleide van $\phi(v,\tau)$ naar τ voor $\tau=0$ is gelijk aan nul voor alle v.
P7: groep-delay = eerste-orde tijd-moment van distributie.
Q7: Q5 en partiële afgeleide van $\phi(v,\tau)$ naar v voor $v=0$ is gelijk aan nul voor alle τ.
P8: tijd-ondersteuning: distributie is nul in tijddomein als signaal nul is in tijddomein.

Q8: $$\int \phi(v,\tau)\, e^{-j2\pi vt} dv = 0 \quad voor \ |\tau| < 2|t|.$$

P9: frequentie-ondersteuning: distributie is nul in frequentiedomein als signaal nul is in frequentiedomein.

Q9: $$\int \phi(v,\tau)\, e^{j2\pi f\tau} d\tau = 0 \quad voor \ |v| < 4\pi|f|.$$

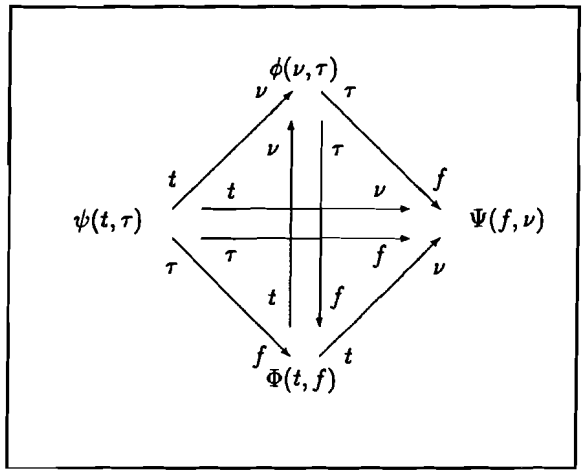
P10: verminderde interferentie.
Q10: $\phi(v,\tau)$ is een 2D-lowpass filter type.

Tabel 1: Eigenschappen van distributies en corresponderende eisen aan kernel.

De eigenschappen van de distributie worden in deze tabel aangegeven met een P. De eisen die aan de kernel gesteld worden zijn aangegeven met een Q.

Naast de beschrijving van een kernel-functie in het (v,τ)-vlak kunnen we een kernel-functie ook vastleggen in 3 andere domeinen. De overgangen tussen deze domeinen bestaan uit Fourier transformaties. Zo verkrijgen we kernels in respectie-

velijk het (t,f)-vlak, (v,τ)-vlak, (t,τ)-vlak en (v,f)-vlak. Het schema in figuur 1 geeft de onderlinge relaties weer.



Figuur 1: Relaties tussen kernels in verschillende domeinen. Een pijl geeft een Fourier transformatie weer.

Aan de hand van deze 4 soorten kernels kan een element uit de Cohen-klasse op 4 manieren uniek gekarakteriseerd worden:

$$P_x(t,f) = \int_{\tau}\int_{t'} \Psi(t-t',\tau)\, q_x(t',\tau)\, e^{-j2\pi f\tau} dt' d\tau \qquad (8a)$$

$$P_x(t,f) = \int_{\nu}\int_{f'} \Psi(f-f',\nu)\, Q_x(f',\nu)\, e^{j2\pi t\nu} df' d\nu \qquad (8b)$$

$$P_x(t,f) = \int_{t'}\int_{f'} \Phi(t-t',f-f')\, W_x(t',f')\, dt' df' \qquad (8c)$$

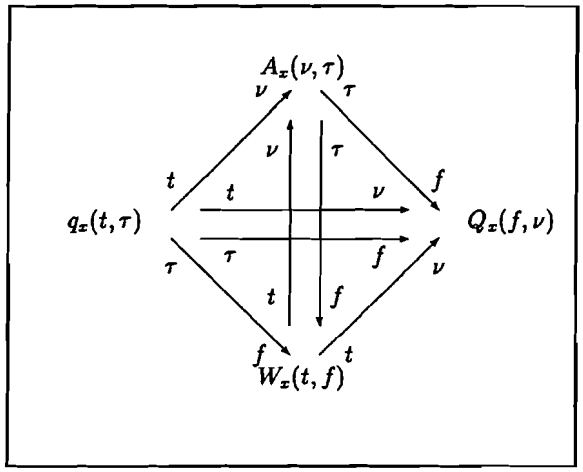
$$P_x(t,f) = \int_{\tau}\int_{\nu} \Phi(\nu,\tau)\, A_x(\nu,\tau)\, e^{j2\pi(t\nu-f\tau)} d\tau d\nu \qquad (8d)$$

waarin

$$q_x(t,\tau) = x(t+\tau/2)\,x^*(t-\tau/2) \quad en$$
$$Q_x(f,\nu) = X(f+\nu/2)\,X^*(f-\nu/2)$$

en de WD en AF zijn zoals gedefinieerd in (4) en (5).

De bilineaire signaal representaties q_x, Q_x, W_x en A_x zijn aan elkaar gerelateerd via Fourier transformaties (zie figuur 2).



Figuur 2: Relaties tussen de verschillende bilineaire signaal representaties.

In (8c) herkennen we een 2-dimensionale convolutie van de WD met een kernel-functie $\Phi(t,f)$. Deze convolutie gaat over in een vermenigvuldiging in het Fourier-domein (8d). We zien dat een element uit de Cohen-klasse verkregen kan worden door een filtering van de WD met een kernel-functie. Dit principe kan als basis dienen voor de berekening van een element uit de Cohen-klasse. In stappen:

- Bereken $A_x(v,\tau)$.
- Kies $\phi(v,\tau)$. Hiermee selecteren we een element uit de Cohen-klasse.
- Bepaal produkt: $A_x(v,\tau)\phi(v,\tau)$.
- Transformeer terug naar (t,f)-domein door middel van Fourier transformatie.

De berekening van $A_x(v,\tau)$ kan indirect gebeuren door eerst de WD te bepalen en deze vervolgens Fourier te transformeren. Het nu verkregen rekenschema kan vergeleken worden met de 'fast convolution' zoals die in de filtertechniek toegepast wordt. Het algoritme kan geïmplementeerd worden met FFT's [Tim94].

2.3.1 Belangrijke elementen uit de Cohen-klasse

Een belangrijk element uit de Cohen-klasse is natuurlijk de Wigner Distributie. Deze kan verkregen worden door als kernel te nemen: $\phi(v,\tau) = 1$. (7) gaat dan over in (6). De WD voldoet aan eisen P1-P9. Alleen aan P0 en P10 wordt niet voldaan, wat meteen het nadeel is van de WD.

Het spectrogram maakt ook deel uit van de Cohen-klasse. Deze distributie heeft de gunstige eigenschap dat ze altijd positief is. Een nadeel is dat tijd- en frequentie-

resolutie afhankelijk van elkaar zijn. Een goede tijdresolutie levert een matige frequentieresolutie op en omgekeerd. Dit komt doordat het spectrogram afgeleid is van de Short-Time-Fourier-Transform (STFT) of sliding-window-spectrum, waarin gebruik gemaakt wordt van een vensterfunctie. Door het gebruik hiervan is men gebonden aan het onzekerheidsprincipe: Men kan geen vensterfunctie maken met een willekeurig kleine duur én een willekeurig smalle bandbreedte. Dit resulteert in het genoemde tijd-frequentie resolutie compromis. Het spectrogram voldoet aan eisen P1-P3 en P10.

Andere belangrijke TFD's uit de Cohen-klasse zijn:

- Cone-Kernel Distribution (CKD): P1-P3, P8 en P10.
- Choi-Wiliams Distribution (CWD): P1-P7 en P10.
- Reduced-Interference Distributions (RID): P1-P10.

RID's vormen een groep binnen de Cohen-klasse. Alle RID's hebben produkt-kernels, dat wil zeggen dat de kernel geschreven kan worden als: $\phi(v,\tau) = \phi(v\tau)$. Voor het verkrijgen van RID's bestaat er een vaste ontwerpmethode [JeoWil92]. Wanneer men deze methode volgt, verkrijgt men een distributie die automatisch voldoet aan P1-P10. RID's hebben het nadeel dat ze niet voldoen aan P0 en dat aan eis P10 slechts gedeeltelijk tegemoet gekomen kan worden. Dit komt doordat een aantal eisen, met name de marginaal-eisen P4 en P5, in tegenspraak zijn met eis P10. Wel kunnen we stellen dat de RID's het beste is wat de Cohen-klasse met behulp van vaste kernels te bieden heeft. Wanneer we eisen P0 (positiviteit) en P4, P5 (marginalen) willen combineren in 1 distributie zullen we de Cohen-klasse moeten verlaten.

2.4 Onderdrukking van interferentie-termen

In de vorige paragraaf kwam naar voren dat ieder element uit de Cohen-klasse verkregen kan worden door een filtering van de WD. Tevens is bekend dat de interferentie-termen in het (v,τ)-vlak zich op afstand van de oorsprong bevinden (§2.2). Dit leidt tot de conclusie dat interferentie-termen onderdrukt kunnen worden door een element uit de Cohen-klasse te kiezen, die een kernel heeft met een laagdoorlaat karakter in het (v,τ)-domein. De interferentie-termen worden nu uitgefilterd. We zullen nu aantonen, aan de hand van voorbeelden, dat bij de keuze van een element uit de Cohen-klasse zich een aantal problemen voordoen. De voorbeelden zijn zodanig gekozen dat ze aantonen dat gebruik van vaste kernels leidt tot distributies met een overmaat aan interferentie-termen en een verlies aan resolutie in de signaaltermen.

Om het één en ander te illustreren maken we gebruik van de volgende TFD's uit de Cohen-klasse met een vaste kernel:

- Cone-Kernel Distributie (CKD)

$$\phi(v,\tau) = g(\tau)\,|\,\tau\,|\,\frac{\sin(\pi\tau v)}{\pi\tau v}$$

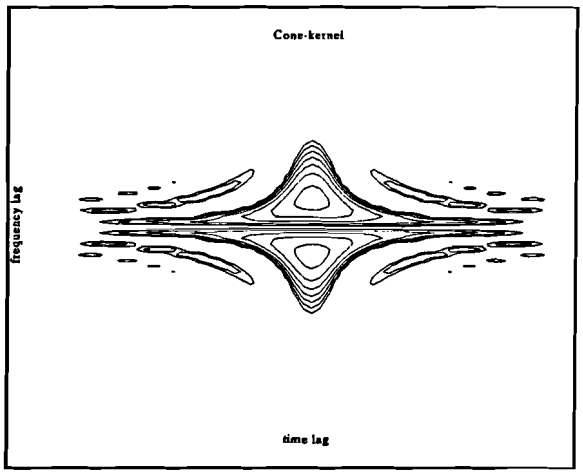
met $g(\tau)$ een Gaussische vensterfunctie.

- Choi-Williams Distributie (CWD)

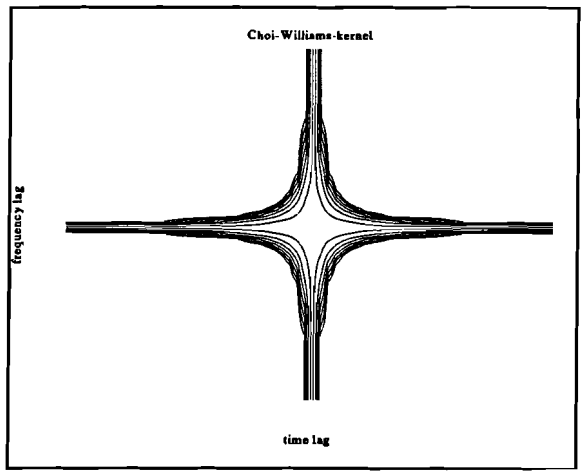
$$\phi(v,\tau) = \exp[(2\pi\tau v)^2/\sigma]$$

met σ de spreiding van de kernel.

De contourplots van beide kernels zien er als volgt uit:



Figuur 3: Contourplot van Cone-kernel.



Figuur 4: Contourplot van Choi-Williams kernel.

Beide kernels hebben een relevante bijdrage bij één van beide assen (Cone) of bij beide assen (Choi-Wiliams) van het (v,τ)-vlak.

De volgende testsignalen worden gebruikt; Signaal s1(t) is de som van 2 Gaussische pulsen:

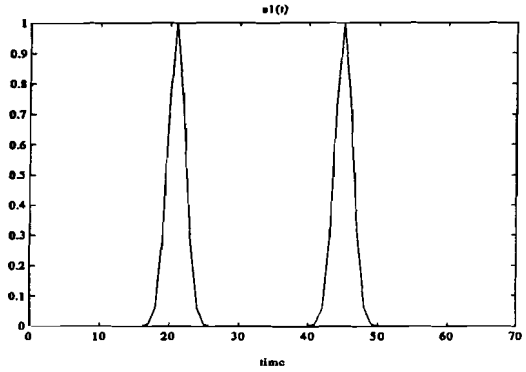
$$s_1(t) = e^{-a(t-T_1)^2} + e^{-a(t-T_2)^2}$$

$a = 0.31$, $T_1 = 21$ en $T_2 = 45$.

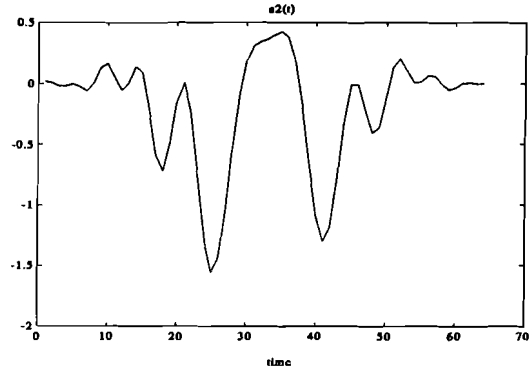
Signaal s2(t) is de som van twee lineair gemoduleerde signalen:

$$s_2(t) = e^{-a(t-T_1)^2+jc(t-T_1)^2-j\omega_1(t-T_1)} + e^{-a(t-T_2)^2+jc(t-T_2)^2-j\omega_2(t-T_2)}$$

met a = 0.005, c = 0.025, T_1 = 33, T_2 = 33, w_1 = 0.34 en w_2 = -0.34. Beide signalen worden eens per seconde bemonsterd. Signalen s1(t) en s2(t) zijn afgebeeld in Figuur 5a en 5b.

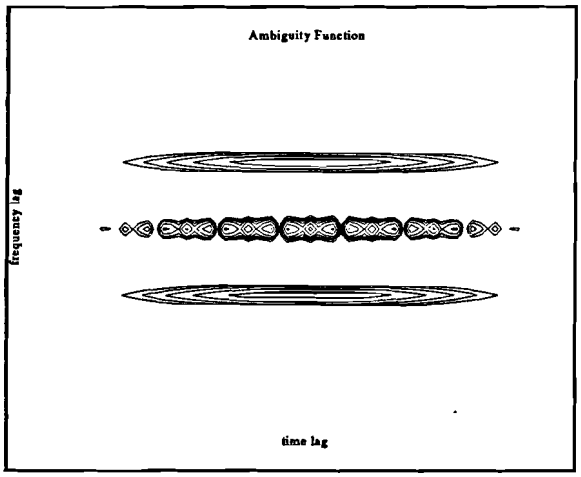


Figuur 5a: Signaal s1(t).

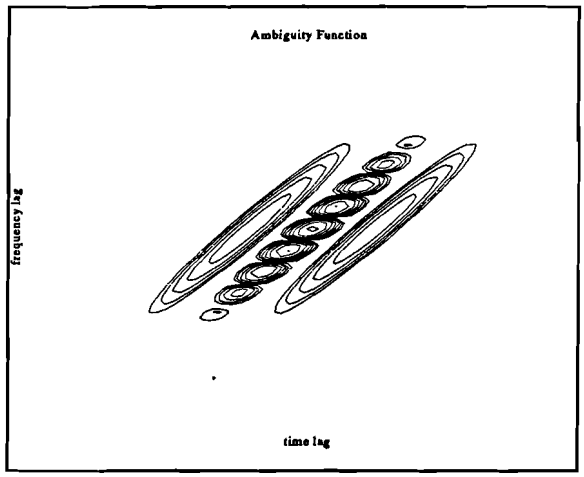


Figuur 5b: Signaal s2(t).

Deze twee signalen hebben de volgende twee corresponderende Ambiguity functies:

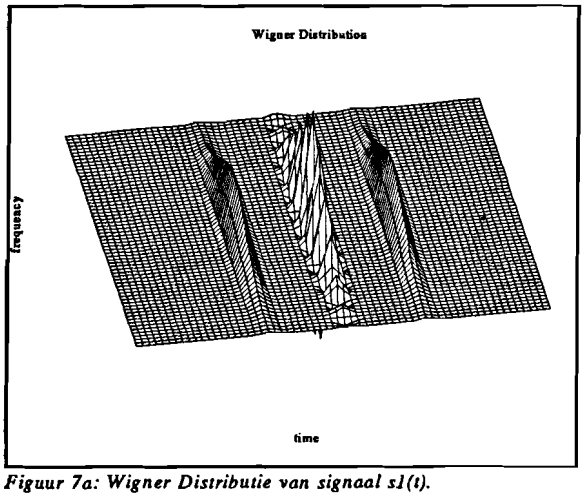


Figuur 6a: Ambiguity functie van signaal s1(t).

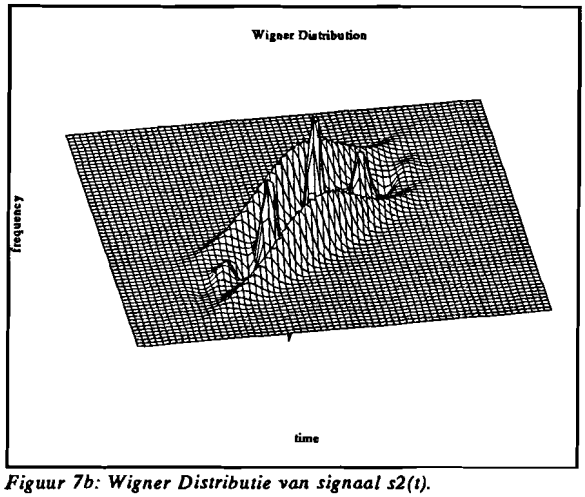


Figuur 6b: Ambiguity functie van signaal s2(t).

We zien dat de signalen alleen van elkaar verschillen in oriëntatierichting in het (v,τ)-vlak. De signaal-termen bevinden zich in het centrum van de AF. De interferentie-termen bevinden zich aan de buitenkant. Hoe deze verschillende termen zich uiten in de WD zien op de volgende pagina.



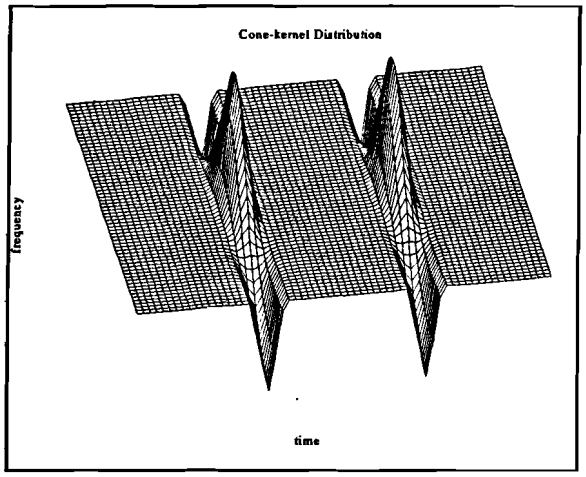
Figuur 7a: Wigner Distributie van signaal s1(t).



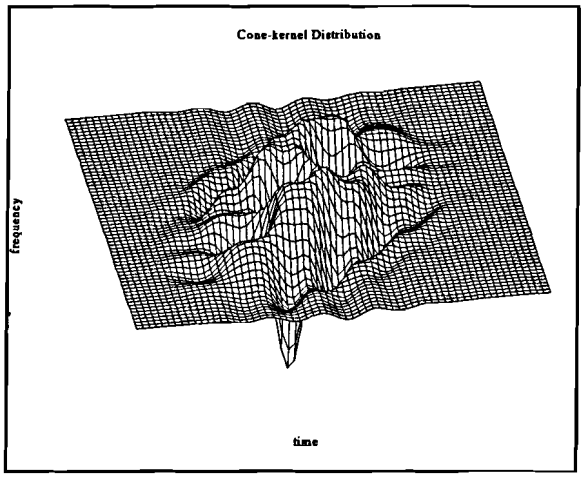
Figuur 7b: Wigner Distributie van signaal s2(t).

Duidelijk te zien zijn de interferentie-termen tussen de twee signaal-termen. Ze komen voor op plaatsen in het (t,f)-vlak waar geen energiebijdrage behoort te zijn. Ook te zien is het oscillatorische karakter van de interferentie-termen.

De CKD levert de volgende resultaten op:



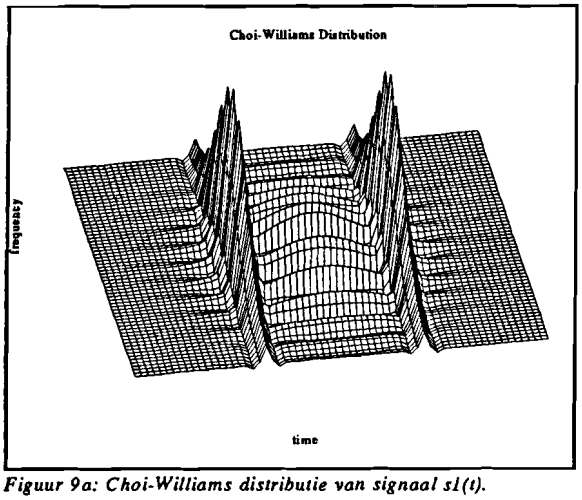
Figuur 8a: Cone-Kernel Distributie van signaal s1(t).



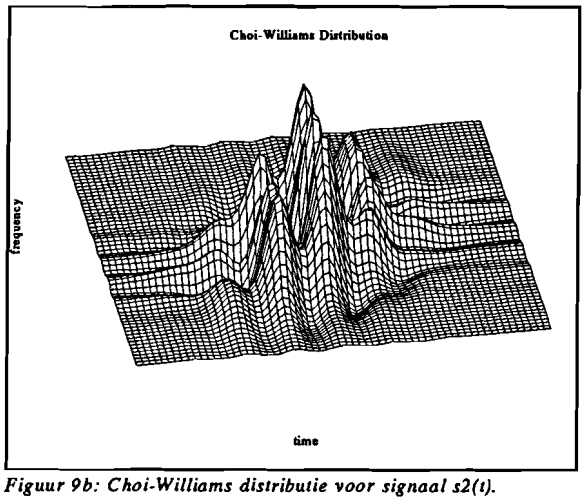
Figuur 8b: Cone-Kernel Distributie van signaal s2(t).

De CKD levert voor pulsachtige signalen zoals signaal s1(t) in het algemeen goede resultaten. Signaal s2(t) bevat signaal-termen in het (v,τ)-vlak op een afstand van de v- en τ-as. De Cone-Kernel heeft op deze punten geen bijdrage (zie figuur 3), waardoor een groot deel van de signaal-termen worden afgekapt. Het gevolg is een verlies in resolutie van de signaal-termen.

De CWD's voor signalen s1(t) en s2(t) zien er als volgt uit:



Figuur 9a: Choi-Williams distributie van signaal s1(t).



Figuur 9b: Choi-Williams distributie voor signaal s2(t).

De CWD voldoet aan de marginaal-eisen, maar een gevolg daarvan is dat de interferentie niet voldoende onderdrukt kan worden. Dit zien we bij de CWD voor signaal s1(t). Bij de CWD van signaal s2(t) treden de zelfde problemen op als bij de CKD: De Choi-Williams kernel is nul op plaatsen waar signaal-termen dit niet zijn, met een verlies aan resolutie van de signaal-termen tot gevolg.

Concluderend: Een TFD uit de Cohen-klasse met een vaste kernel kan alleen goede resultaten behalen voor een bepaalde configuratie van de AF signaal-termen en interferentie-termen en dus alleen voor een bepaalde klasse van signalen.

Omdat de locaties van signaal- en interferentie-termen afhankelijk zijn van het signaal dat we willen analyseren is het te verwachten dat de resultaten beter zullen zijn wanneer we de kernel-functie af laten hangen van het signaal. Op welke manier we dit gaan doen, komt aan de orde in het volgende hoofdstuk.

Voor we hiertoe overgaan eerst nog iets over de eigenschappen waaraan de TFD moet voldoen. Welke minimum eisen stellen we aan de TFD? Welnu dat zijn de volgende:

a) $P_x(t,f)$ moet voldoen aan de tijd-marginaal (eis P4),
b) $P_x(t,f)$ moet voldoen aan de frequentie-marginaal (eis P5),
c) $P_x(t,f) \geq 0$,
d) $P_x(t,f)$ geeft correlaties tussen tijd- en frequentie-inhouden van signaal x weer.

Eis (c) in combinatie met eis (d) houdt in dat de interferentie-termen van de WD zo goed mogelijk onderdrukt moeten worden en dat de signaal-termen van de WD zoveel mogelijk doorgelaten moeten worden. Door het verwijderen van de interferentie-termen wordt de Wigner distributie positief. Door het kiezen van de WD als uitgangspunt wordt er gezorgd voor de aanwezigheid van correlaties tussen tijd- en frequentie-inhouden.

Het tegelijkertijd voldoen aan eisen a, b en c is niet mogelijk binnen de Cohen-klasse. Als we toch aan deze minimum eisen willen voldoen moeten we de Cohen-klasse verlaten. De manier die we hier volgen is:

- Verwijder eerst zo goed mogelijk de interferentie-termen met behulp van een signaalafhankelijke kernel-functie, zodat aan eis c en d is voldaan (hoofdstuk 3).
- Verlaat de Cohen-klasse van TFD's door een bewerking op de TFD uit de vorige stap uit te voeren, zodat voldaan wordt aan eis a en b (hoofdstuk 4)

Met de eerste stap zullen we nu beginnen.

3 Signaalafhankelijke kernel-functies

Om een bilineaire TFD te vinden die de 'beste' tijd-frequentie representatie is van een gegeven signaal, gaan we gebruik maken van een optimalisatie techniek. Uitgangspunt hierbij is dat, gegeven een signaal, de methode automatisch een kernel-functie genereert die optimaal is ten op zichte van een aantal gekozen criteria. De kernel wordt vervolgens gebruikt om de WD te filteren zoals beschreven in het vorige hoofdstuk. De belangrijkste eis waaraan de resulterende distributie moet voldoen is dat de interferentie-termen zoveel mogelijk onderdrukt worden. Aan de andere eisen zoals de marginaal-eisen wordt in eerste instantie geen aandacht besteed.

3.1 Continue optimalisatie formulering

Omdat er bij de AF een scheiding optreedt tussen de signaal-termen en de interferentie-termen, is deze functie bij uitstek geschikt om als uitgangspunt te dienen voor de optimalisatie techniek. Gegeven een signaal en de bijbehorende AF: $A(\nu,\tau)$ dan definiëren we de optimale kernel als de reële, niet negatieve functie $\phi_{opt}(\nu,\tau)$ die het volgende optimalisatie probleem oplost [BarJon93]:

$$\max(\phi) \quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |A(\nu,\tau)\phi(\nu,\tau)|^2 d\nu d\tau, \tag{9a}$$

onder de voorwaarde dat:

$$\phi(0,0) = 1, \tag{9b}$$

$$\phi(\nu,\tau) \textit{ is radiaal niet stijgend}, \tag{9c}$$

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |\phi(\nu,\tau)|^2 d\nu d\tau \le \alpha. \tag{9d}$$

(9a) geeft hierbij aan wat de kwaliteitsindex van de kernel is. Hoe hoger deze waarde des te beter is de kernel. Deze kwaliteitsindex en de voorwaarden (9b), (9c) en (9d) zorgen er samen voor dat signaal-termen worden bevoordeeld ten opzichte van de interferentie-termen. De 3 voorwaarden zorgen er namelijk voor dat de kernel van een laagdoorlaat type is met vast volume α. De voorwaarden dicteren niet een exacte vorm van de kernel. De vorm wordt bepaald door de maximalisatie van de kwaliteitsindex.

Voorwaarde (9b) zorgt ervoor dat de distributie voldoet aan:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P_s(t,f)\,dfdt = E_s$$

Zonder voorwaarde (9c) zou de optimale kernel grote waarden aannemen op plaatsen waar $|A(v,\tau)|^2$ groot is, onafhankelijk of het nu signaal-termen of interferentie-termen betreft. Als men uitgaat van het feit dat signaal-termen en interferentie-termen zich gescheiden van elkaar in het (v,τ)-vlak bevinden, dan zorgt de monoticiteit (voorwaarde (9c)) van de kernel ervoor dat er geen kernel-volume verspild wordt aan interferentie-termen. De ruimte tussen de signaal-termen en interferentie-termen levert namelijk nauwelijks een bijdrage aan de kwaliteitsindex omdat daar de $|A(v,\tau)|^2$ en dus ook
$|A(v,\tau)\phi(v,\tau)|^2$ klein is. Op deze manier zorgt de gekozen optimalisatie formulering ervoor dat kernels gevonden worden die signaal-termen doorlaten en interferentie-termen sperren onafhankelijk van oriëntatie van deze termen in het (v,τ)-vlak.

De gekozen optimalisatie formulering levert een lineair optimalisatie probleem op (lineair in $|\phi(v,\tau)|^2$). Hoe dit probleem opgelost wordt leest u in hoofdstuk 5.

3.2 Keuze van kernel-volume α

Met de keuze van α controleren we in hoeverre we de interferentie-termen onderdrukken ten koste van eventueel resolutieverlies in de signaal-termen. Als α te klein is zullen ook een groot deel van de signaal-termen verloren gaan wat resulteert in verlies van resolutie. Als α te groot gekozen wordt zullen te veel interferentie-termen doorgelaten worden. We zullen nu enige richtlijnen geven voor de keuze van α.

Tot een beneden en bovengrens voor α kunnen we komen door de energie overdracht van de kernel te meten voor een maximaal in (v,τ)-vlak geconcentreerd monocomponent signaal, dat wil zeggen een Gaussisch signaal. Als we de fractie energie die door de kernel overgedragen wordt, aanduiden met δ, dan blijkt uit de analyse [BarJon93] dat geldt:

$\alpha = -\ln(1-\delta)$.

Een redelijke keuze voor de ondergrens van α kan gevonden worden door uit te gaan van de hoeveelheid energie die een spectrogram kernel overdraagt. Er is namelijk geen reden om aan te nemen dat het zin heeft meer afvlakking van de signaal-termen toe te laten als bij het spectrogram het geval is. De maximale energie overdracht van een spectrogram kernel bedraagt: $\delta = \frac{1}{2}$.

Als we dit invullen komen we tot:

$\alpha \geq 0.69$.

Om de vervorming van signaal-termen te minimaliseren moeten we de energie-overdracht maximaliseren. Dat wil zeggen δ->1. Dit betekent echter dat α op een gegeven moment exponentieel moet groeien wil men nog een relevante verbetering in de energie overdracht bewerkstelligen. Dit betekent dat het waarschijnlijk is dat er interferentie-termen meegenomen zullen worden. Een redelijke bovengrens wordt gevonden door de α te nemen die het kniepunt vormt in de δ versus α kromme:

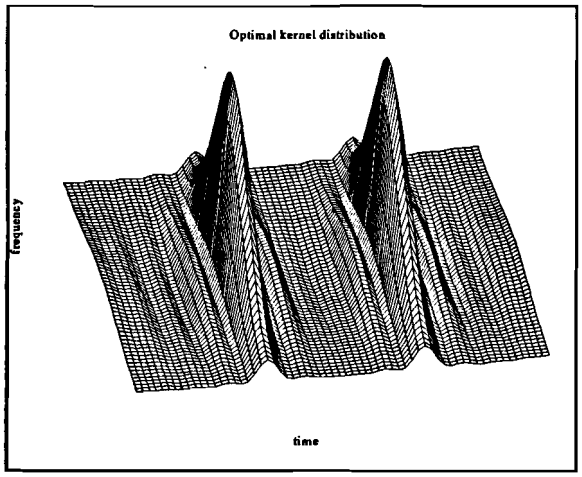
$\alpha \leq 3$.

Gegeven onder- en boven-grens zijn slechts een richtlijn om een overmaat aan afvlakking of een te veel aan interferentie-termen in de TFD te voorkomen.

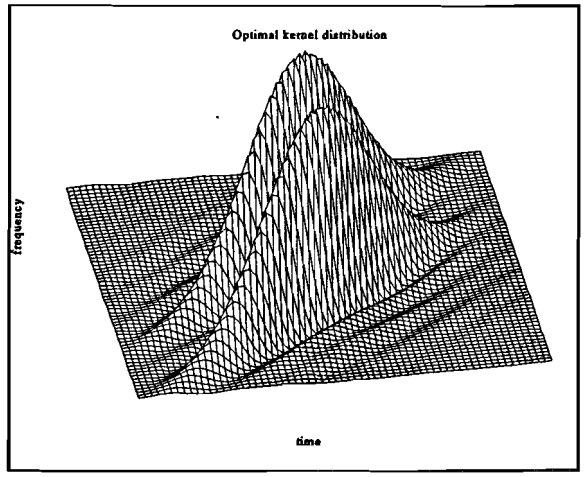
3.3 Optimale-kernel distributies van de testsignalen

Om de prestaties van TFD's met signaalafhankelijke kernels te vergelijken met de prestaties van TFD's met een vaste kernel zullen we de zelfde testsignalen gebruiken als in hoofdstuk 2. Voordat de Optimale-Kernel TFD (OKTFD) berekend wordt, zal de kernel, die in eerste instantie bestaat uit een masker van énen en nullen, afgevlakt worden om oscillatorische verschijnselen in de TFD te voorkomen. De gebruikte Gaussische afvlakfunctie heeft hierbij een spreiding σ van 1.5. Over het nut van afvlakking meer in hoofdstuk 5.

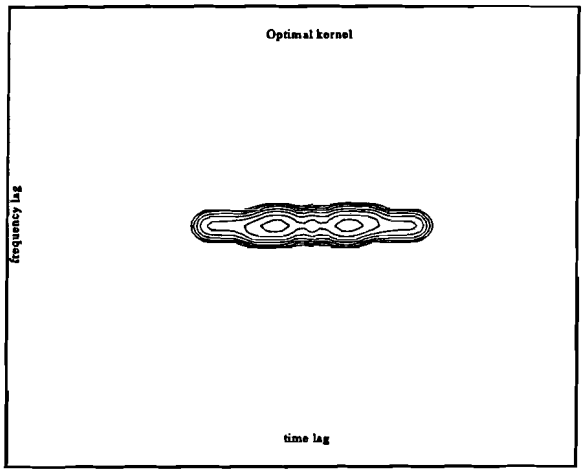
Van signalen s1(t) en s2(t) zijn in figuren 10a en 10b de OKTFD's weergegeven. Figuren 11a en 11b tonen de contourplots van de bijbehorende optimale kernels.



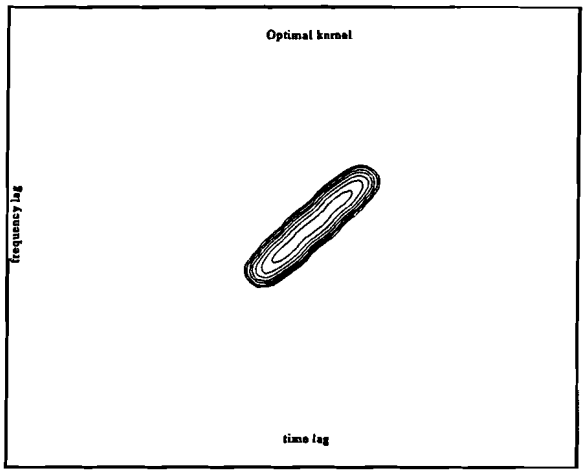
Figuur 10a: OKTFD van signaal s1(t).



Figuur 10b: OKTFD van signaal s2(t).



Figuur 11a: Contourplot van de optimale kernel voor s1(t).

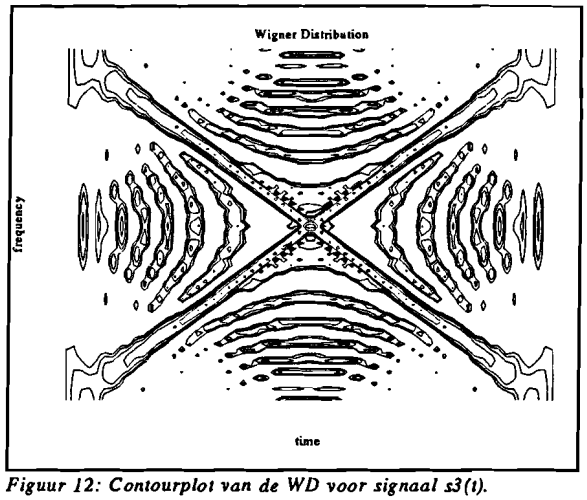


Figuur 11b: Contourplot van de optimale kernel voor s2(t).

De interferentie-termen die te zien zijn in TFD's met vaste kernels zijn grotendeels verwijderd, terwijl de signaal-term resolutie nog goed is. Te zien is ook dat de vorm van de kernel zich aanpast aan het signaal en dat de oriëntatierichting automatisch met het signaal mee verandert. Bij monocomponent signalen levert de OKTFD resultaten op die sterk lijken op de resultaten van de WD.

We illustreren de prestaties van de OKTFD nog aan de hand van één voorbeeld signaal. Hiertoe nemen we een reëel signaal s3(t) dat bestaat uit de som van twee lineair gemoduleerde signalen die elkaar kruisen in het (t,f)-vlak.

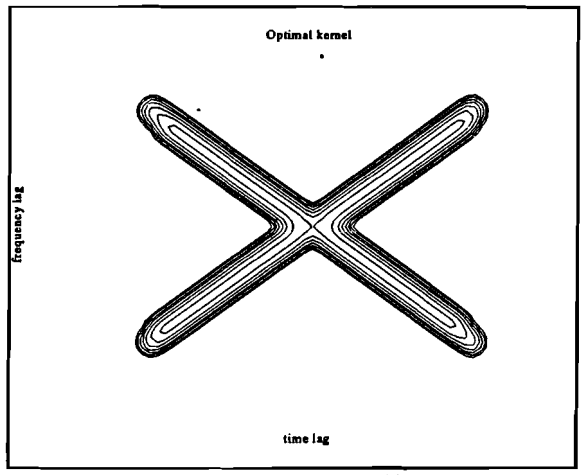
Een contourplot van de WD voor dit signaal is weergegeven in figuur 12.



Figuur 12: Contourplot van de WD voor signaal s3(t).

We zien in deze contourplot de signaal-termen als twee rechte lijnen en daaromheen de interferentie-termen.

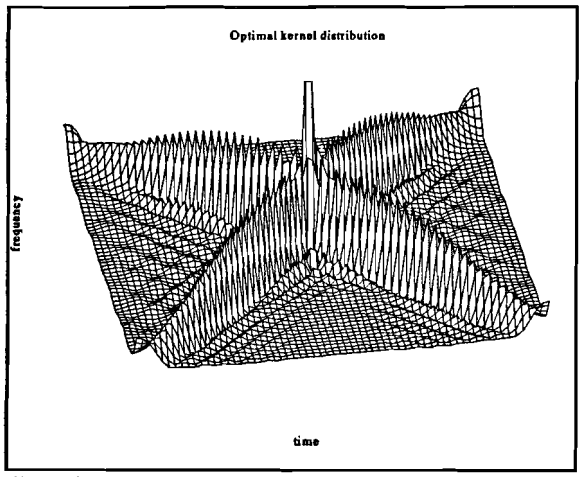
De OKTFD levert een kernel op zoals aangegeven in figuur 13.



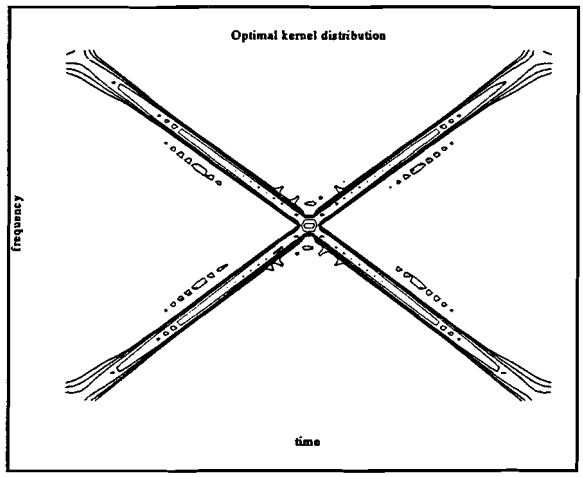
Figuur 13: Optimale kernel voor signaal s3(t).

De representatie van het signaal in het (v,τ)-vlak lijkt sterk op die in het (t,f)-vlak (figuur 12). Ook daar zijn er twee lijnen met signaal-termen te herkennen met daaromheen de interferentie-termen. In figuur 13 zien we dat de kernel dezelfde vorm aanneemt als de signaal-termen in het (v,τ)-vlak.

Na vermenigvuldiging van de optimale kernel met de ambiguity functie en terug-transformatie hiervan levert dit de volgende OKTFD op:



Figuur 14a: 3D-plot van de OKTFD van s3(t).



Figuur 14b: Contourplot van de OKTFD van s3(t).

We zien dat interferentie-termen vrijwel verwijderd zijn en dat de signaal-termen behouden blijven.

4 De Minimum-Cross-Entropy methode

In §2.4 hebben we de minimum eis gesteld dat de TFD moet voldoen aan:

a) $P_x(t,f)$ moet voldoen aan de tijd-marginaal (eis P4),
b) $P_x(t,f)$ moet voldoen aan de frequentie-marginaal (eis P5),
c) $P_x(t,f) \geq 0$,
d) $P_x(t,f)$ geeft correlaties tussen tijd- en frequentie-inhouden van signaal x weer.

Na het verkrijgen van een OKTFD wordt tot op zekere hoogte voldaan aan eisen c en d. Tot op zekere hoogte want niet alle interferentie-termen kunnen door de optimale kernel verwijderd worden en ook zullen de signaal-termen nooit volledig doorgelaten worden. Het gevolg hiervan is dat de OKTFD op bepaalde plaatsen nog negatief kan zijn. Deze negatieve waarden zijn echter klein ten opzichte van de signaal-termen zodat men kan zeggen dat aan eis c en d bij benadering voldaan wordt.

Bij constructie van de OKTFD is niet gelet op de marginaal-eisen a en b. Dit proberen we in dit hoofdstuk te corrigeren door gebruik te maken van de Minimum-Cross-Entropy (MCE) methode. De OKTFD dient hierbij als een uitgangspunt.

4.1 Principe van de MCE-methode

Bij de MCE-methode wordt gebruik gemaakt van een beginschatting van een distributie (a priori distributie) en informatie over de 'juiste' en gewenste distributie (a posteriori distributie) in de vorm van marginalen of andere beschikbare informatie. Om van a priori naar a posteriori distributie te komen wordt de cross-entropy tussen deze twee distributies geminimaliseerd aan de hand van de extra toegereikte informatie.

Het MCE principe voldoet aan een aantal axioma's die gebaseerd zijn op het algemene principe: Als er meerdere manieren zijn om dezelfde informatie in een resultaat te verwerken, moet het verkregen resultaat consistent zijn. De 4 axioma's zijn [ShoJon80]:

a **Uniciteit**: De a posteriori distributie moet uniek zijn voor de gegeven informatie.
b **Invariantie**: Het zelfde probleem oplossen in een ander coördinatenstelsel moet leiden tot een zelfde resultaat.
c **Systeem onafhankelijkheid**: Formulering van onafh. informatie over onafh. systemen afzonderlijk via onafh. distributies moet het zelfde opleveren als het formuleren van die informatie samen via een simultane distributie. Dus voor prior $p = p_1 p_2$ behoort $q = q_1 q_2$ de bijbehorende a posteriori distributie te zijn.

d **Subset onafhankelijkheid**: Gebruik maken van sets van afzonderlijke conditionele distributies moet het zelfde resultaat opleveren als bij gebruik van de totale distributie.

Het is aangetoond [ShoJon80], [ShoJon81] dat MCE de enige functie is die voldoet aan deze axioma's. De MCE-methode is daarom bij voorkeur de methode voor het verkrijgen van een onbekende distributie gegeven een hoeveelheid beperkte informatie over deze distributie. Deze informatie over de onbekende distributie bestaat uit een beginschatting (de a priori distributie) plus lineaire voorwaarden waar de onbekende distributie aan moet voldoen.

Voor de toepassing voor het verkrijgen van een TFD met gunstige eigenschappen kunnen we het principe van MCE als volgt formuleren:

> Gegeven een positieve beginschatting P'(t,f) met eenheid energie van de onbekende distributie P(t,f), plus de marginaal-voorwaarden, dan is de optimale a posteriori distributie Q(t,f) die distributie, die op unieke wijze de cross-entropy tussen de a priori en a posteriori distributie minimaliseert. De minimalisatie wordt bewerkstelligd door de marginaal-voorwaarden in acht te nemen.

Voor niet-negatieve TFD's wordt de cross-entropy tussen de beginschatting P'(t,f) en a posteriori distributie Q(t,f) gegeven door:

$$\Delta H(Q,P') = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Q(t,f)\ln\frac{Q(t,f)}{P'(t,f)}\,dtdf \qquad (10)$$

De MCE-distributie Q(t,f) voldoet aan de marginaal-voorwaarden en minimaliseert (10).

4.2 De MCE-oplossing

Het probleem van het vinden van een TFD door middel van minimalisatie van de cross-entropy is een optimalisatie probleem met voorwaarden. Bij het stellen van de marginaal voorwaarden:

$$\int_{-\infty}^{\infty} Q_s(t,f)\,df = |s(t)|^2 = m_1(t)$$

$$\int_{-\infty}^{\infty} Q_s(t,f)\,dt = |S(f)|^2 = m_2(f)$$

kan de cross-entropy geminimaliseerd worden met behulp van Lagrange-multipliers. Op deze manier vinden we ook de a posteriori distributie Q(t,f). Het resultaat is:

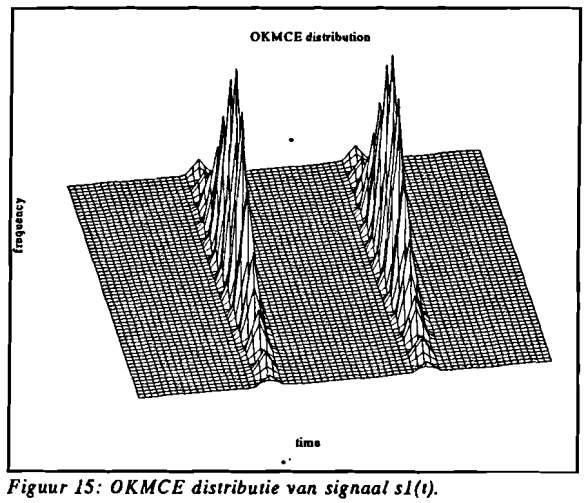
$$Q_s(t,f) = P'(t,f)\,e^{-(\theta_1(t)+\theta_2(f))}$$

waarin $\Theta_1(t)$ en $\Theta_2(f)$ onbekende Langrange multipliers zijn, die gevonden kunnen worden aan de hand van de gestelde voorwaarden [Loug92(3)]. Hoe dit geïplementeerd wordt met behulp van een algoritme zal in hoofdstuk 6 uitgelegd worden. Nu zullen we eerst wat resultaten bekijken van de MCE-methode.

4.3 MCE-distributies van de testsignalen

Als a priori distributie kiezen we, zoals al eerder opgemerkt, de OKTFD van de testsignalen (zie hoofdstuk 3). Wanneer een OKTFD negatieve waarden heeft als gevolg van restanten van interferentie-termen, dan worden deze verwijderd zodat geldt: $P'(t,f) \geq 0$. Dit is een voorwaarde die nodig is voor het kunnen toepassen van de MCE-methode. Omdat gebruik wordt gemaakt van een OKTFD en van een MCE-methode noemen we de resulterende a posteriori distributie een OKMCE distributie.

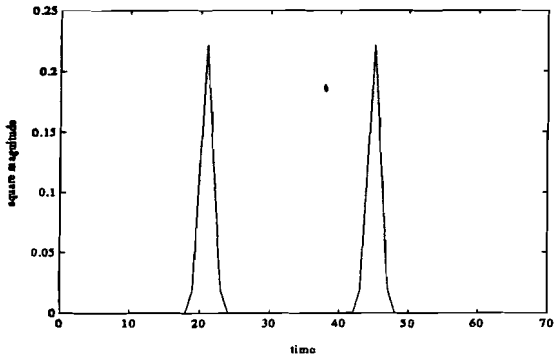
Van signaal s1(t) is de OKMCE distributie in figuur 15 weergegeven:



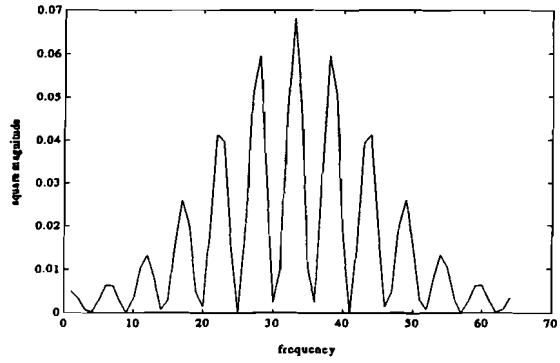
Figuur 15: OKMCE distributie van signaal s1(t).

Te zien is dat de distributie nul is daar waar het signaal ook nul is. Dit geldt zowel in het tijddomein als in het frequentiedomein.

De OKMCE voldoet per definitie aan de marginalen. Dit kan ingezien worden als men de marginalen uit figuren 16a en 16b vergelijkt met de vorm van de OKMCE.



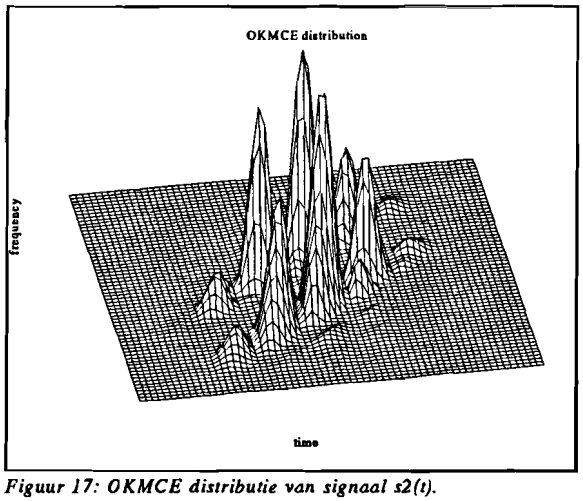
Figuur 16a: Tijd-marginaal van signaal s1(t).



Figuur 16b: Frequentie-marginaal van signaal s1(t).

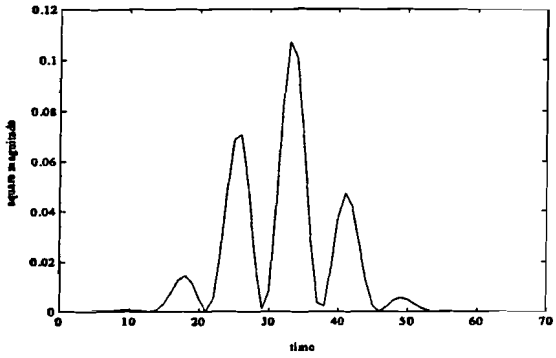
De maximale afwijking tussen de marginalen berekend uit de OKMCE en de werkelijke marginalen is bij alle voorbeelden kleiner of gelijk aan $1x10^{-7}$.

Signaal s2(t) levert de volgende OKMCE op:

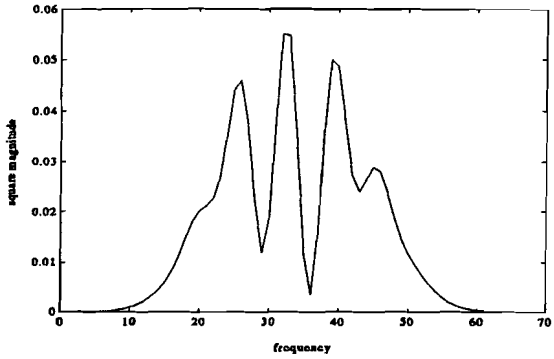


Figuur 17: OKMCE distributie van signaal s2(t).

De pulsachtige vorm van deze distributie geeft aan dat de energie niet gelijkmatig over de twee lijnen van momentele frequentie verdeeld is, maar dat de energie bijdrage alleen ongelijk aan nul is als de tijd- en frequentie-marginalen ongelijk aan nul zijn. De corresponderende marginalen zijn afgebeeld in de figuren 18a en 18b.



Figuur 18a: Tijd-marginaal van signaal s2(t).



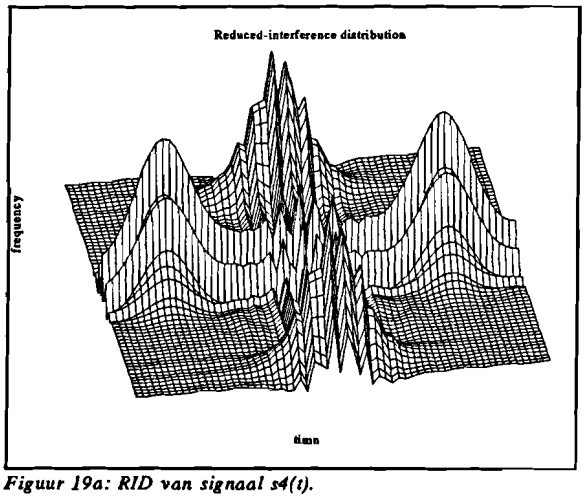
Figuur 18b: Frequentie-marginaal van signaal s2(t).

Tenslotte nog een signaal waarmee het verschil aangetoond wordt tussen een RID en de OKMCE distributie. We nemen hiervoor een signaal s4(t) dat bestaat uit de som van 4 complexe Gaussische pulsen.

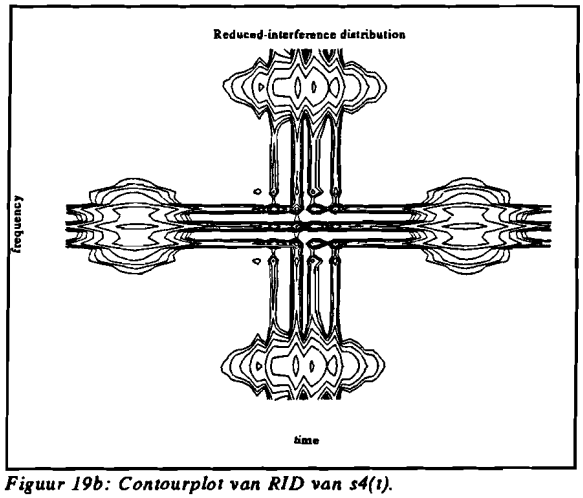
$$s_4(t) = e^{-a(t-T_1)^2} + e^{-a(t-T_2)^2-j\omega t} + e^{-a(t-T_2)^2+j\omega t} + e^{-a(t-T_3)^2}$$

parameters: a = 0.24, T_1 = 10, T_2 = 33, T_3 = 53 en w = 1.22.

Van de RID zijn in de volgende figuren een 3D-representatie en een contourplot weergegeven.

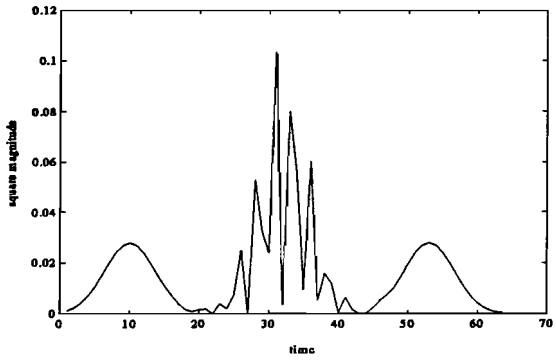


Figuur 19a: RID van signaal s4(t).

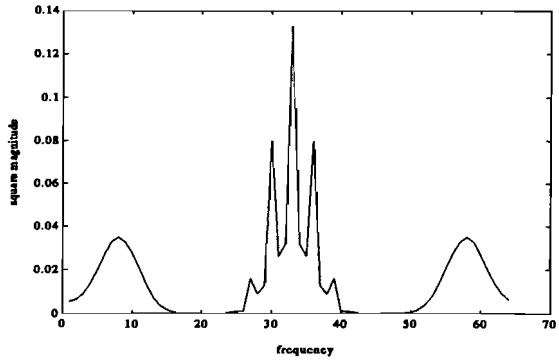


Figuur 19b: Contourplot van RID van s4(t).

De RID zorgt voor het voldoen aan de marginaaleisen. De corresponderende marginalen hebben de volgende vorm:



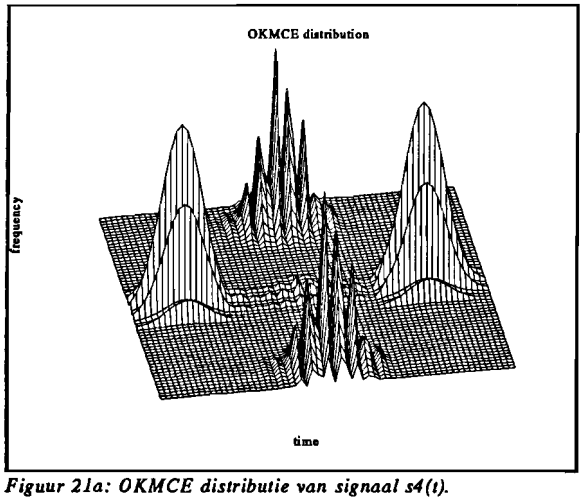
Figuur 20a: Tijdmarginaal van signaal s4(t).



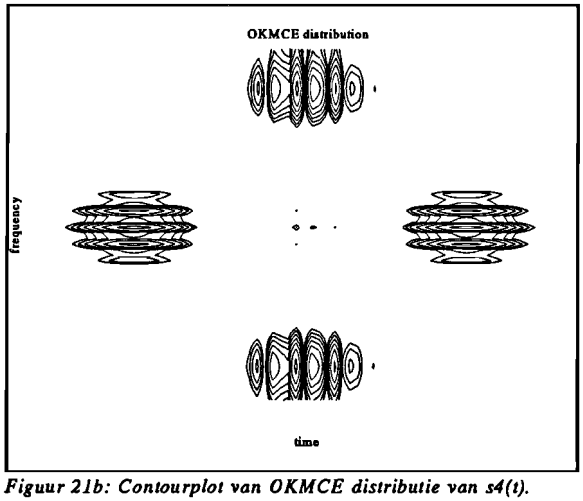
Figuur 20b: Frequentiemarginaal van signaal s4(t).

Doordat een RID een element is uit de Cohen-klasse moet er altijd een compromis gesloten worden tussen de interferentie-onderdrukking en signaal-term resolutie. De eisen met betrekking tot de interferentie-onderdrukking en de eisen met betrekking tot de marginalen zijn ook niet met elkaar te verenigen, zodat zich ook hier een compromis aftekent. Dit compromis komt voor dit signaal tot uiting door het aanwezig zijn van energiebijdragen tussen de 4 Gaussische pulsen en door het lokaal negatief zijn van de RID.

Wanneer we eerst een OKTFD bepalen door middel van een signaalafhankelijke kernel en vervolgens de MCE-methode toepassen, verkrijgen we de volgende OKMCE distributie:



Figuur 21a: OKMCE distributie van signaal s4(t).



Figuur 21b: Contourplot van OKMCE distributie van s4(t).

Zichtbaar is dat de OKMCE distributie bijna volledig de interferentie-termen tussen de 4 pulsen wegwerkt. Zowel De RID als de OKMCE distributie voldoen aan de marginaal-voorwaarden maar de OKMCE distributie zorgt ervoor dat de benodigde informatie zich concentreert rond (t,f)-punten waar ook echt een energie bijdrage verwacht kan worden. Een ander voordeel ten opzichte van de RID is dat een OKMCE distributie altijd positief is.

5 Algoritme voor het ontwerp van een optimale kernel

Voor het verkrijgen van een optimale kernel is het nodig een lineair optimalisatie probleem op te lossen. Dit probleem is in hoofdstuk 3 geformuleerd in continue variabelen. Wanneer we het probleem willen oplossen ten behoeve van praktische doeleinden dan moet het optimalisatie probleem opgelost worden met behulp van een computer. Dit betekent dus dat we het probleem in discrete termen moeten formuleren.

5.1 Discrete optimalisatie formulering

Na discretisatie van (9a) tot en met (9d) verkrijgen we het volgende optimalisatie probleem [BarJon94]:

Lineair programma 1 (LP1):

$$\max(\phi) \sum_{m=-N/2}^{N/2-1} \sum_{n=-N/2}^{N/2-1} |A(m,n)|^2 |\phi(m,n)|^2 \qquad (11a)$$

$$\phi(0,0) = 1 \qquad (11b)$$

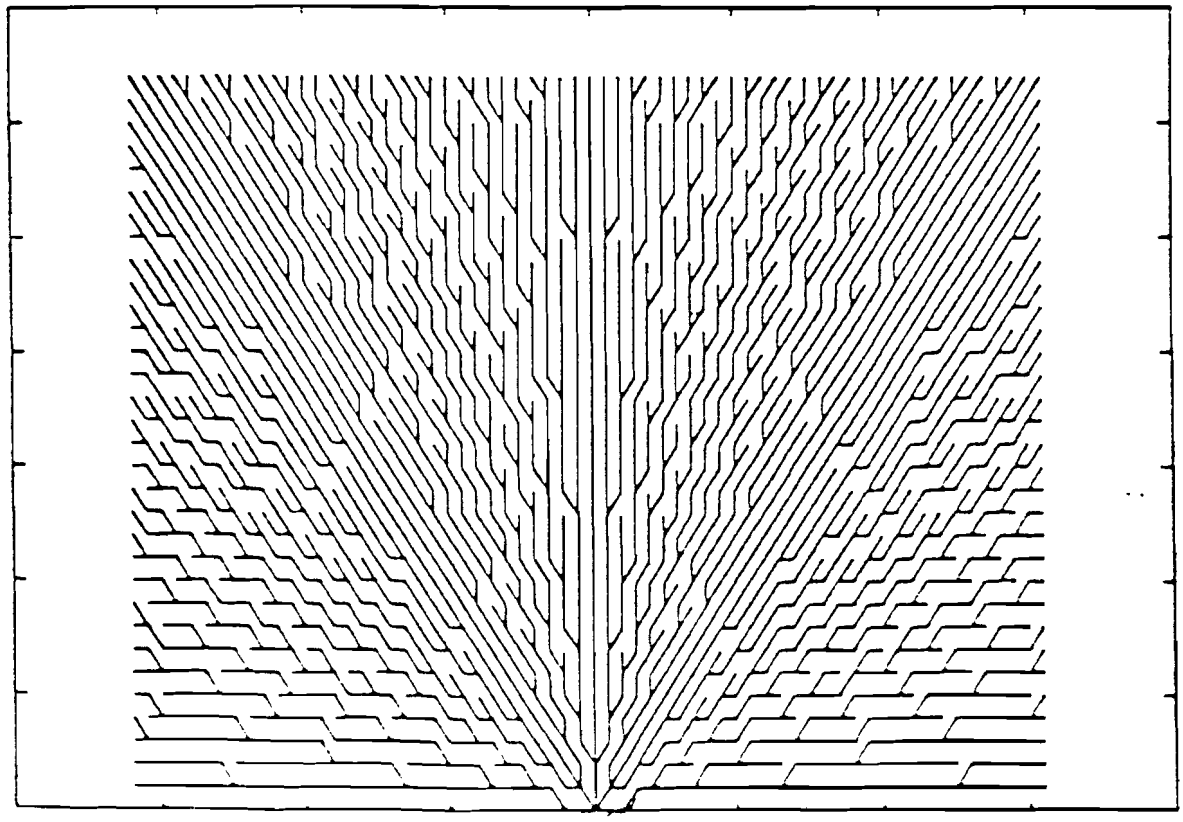
$$\phi(m,n) \quad \textit{is radiaal niet stijgend} \qquad (11c)$$

$$\frac{\Delta_v \Delta_\tau}{2\pi} \sum_{m=-N/2}^{N/2-1} \sum_{n=-N/2}^{N/2-1} |\phi(m,n)|^2 \leq \alpha \qquad (11d)$$

Hierbij is A(m,n) een NxN discrete AF van het te onderzoeken signaal en zijn Δ_v en Δ_τ constanten die de afstanden tussen de monsters in v en τ domein aangeven. Om aliasing te voorkomen moet gelden: $\Delta_v = 2\pi/NT$ en $\Delta_\tau = 2T$, met T de bemonstertijd en N het aantal genomen monsters van het betreffende signaal.

De ambiguity functie AF kan verkregen worden door directe calculatie of via een Fourier transformatie van de Wigner distributie. Deze laatste indirecte methode is gebruikt omdat er reeds een snel algoritme voor de WD beschikbaar was. Een hiervoor benodigde 2-dimensionale Fourier transformatie in de vorm van FFT's, levert geen relevante vertraging van het algoritme op. De totale orde complexiteit blijft gelijk.

Alle monsters van de AF worden berekend op een rechthoekig rooster (m,n) in het (v,τ)-domein. Eis (11c) kan echter alleen in discrete vorm gesteld worden als de monsters op een polair rooster liggen. We moeten namelijk beschikken over de informatie van de AF en $\phi(v,\tau)$ op radiale lijnen in (v,τ)-vlak. Om dit te realiseren zouden we kunnen proberen een AF uit te rekenen op een polair rooster. Hier bestaat echter geen snel algoritme voor. Een andere optie is een 2-dimensionale interpolatie van A(m,n) op rechthoekig rooster, naar een A(p,q) op een polair rooster. Dit algoritme is echter kwa berekeningstijd ook te duur. Een sneller algoritme wordt gevonden door de radiale lijnen in een rechthoekig rooster te benaderen met behulp van een boomstructuur. Een dergelijke boomstructuur is weergegeven in figuur 22.



Figuur 22: Approximatie van radiale lijnen door middel van een boomstructuur.

De gediscretiseerde kernel-voorwaarde (11c) wordt nu geïmplementeerd door te stellen:

$$\phi(m,n) \leq \phi(p(m,n))$$

waarin p(m,n) een roosterpunt voorstelt die het eerste gevonden wordt wanneer we vanuit (m,n) over een benaderde radiale lijn van de boomstructuur, richting de oorsprong gaan. De takken van de boom zijn zodanig geconstrueerd voor een monster dat de afwijking ten op zichte van de radiaal door dat monster minimaal is.

Wanneer de monsterdichtheid vergroot wordt convergeren de takken van de boom naar 'echte' radialen. Hoe een dergelijke boom geconstrueerd wordt is beschreven in [BarJon93].

Omdat kwaliteitsindex (11a) en voorwaarden (11b) tot en met (11d) lineair in $|\phi(m,n)|^2$ zijn kan de optimale kernel gevonden worden door middel van lineair programmeren. Wanneer we hiervoor traditionele methoden gebruiken zal de orde complexiteit van het algoritme veel te hoog worden om voor praktische doeleinden interessant te zijn. In [BarJon94] wordt een algoritme beschreven dat gebruik maakt van de speciale vorm van LP1. Het resultaat is een algoritme met een complexiteit $O(N^2 \log N)$, dat wil zeggen het algoritme heeft dezelfde complexiteit als algoritmen voor de berekening van TFD's met een vaste kernel. In de volgende paragraaf wordt hierop in hoofdlijnen in gegaan.

5.2 Algoritme voor de oplossing van LP1

Het resultaat dat het algoritme moet opleveren is een 1/0-masker dat bij voorkeur nul is op plaatsen waar zich interferentie-termen bevinden in het (v,τ)-vlak. Om dit masker te krijgen moeten we LP1 oplossen. Het oplossen van LP1 is geen probleem als geldt:

$$|A(m,n)|^2 \quad \textit{is radiaal niet-stijgend} \tag{12}$$

De optimale kernel kan dan gevonden worden door de kernel één te maken daar waar de $|A(m,n)|^2$ de grootste waarden heeft, net zolang tot het volume α verbruikt is. Natuurlijk voldoen niet alle AF's aan (12), maar een oplossing van dit probleem is een eerste stap naar een oplossing van LP1.

Wanneer (12) geldt kan LP1 opgelost worden door eerst alle elementen van $|A(m,n)|^2$ te sorteren, met de grootste waarde vooraan in de gesorteerde rij. Vervolgens wordt kernel-volume toegekend voor de eerste α_d elementen uit de gesorteerde rij, waarbij:

$$\alpha_d = \lceil 2\pi\alpha / (\Delta_v \Delta_\tau) \rceil = \lceil N\alpha/2 \rceil$$

met Δ_v en Δ_τ de bemonsteringsintervallen in het (v,τ)-domein.

Het algoritme wordt Sort and Select Algorithm (SSA) genoemd en is in pseudocode hieronder in de tekst opgenomen.

Input: $|A(m,n)|^2 \geq 0$ en $|A(m,n)|^2$ is radiaal niet-stijgend,
$\alpha_d \geq 0$;

Output: $\phi(m,n)$;

Init: $\phi(m,n) := 0$ voor alle m en n,
count := 0 (volume teller);

Loop:
while(count$<\alpha_d$) do
Find: (m,n) := argmax{ $|A(m',n')|^2 : \phi(m',n') = 0$};
Set: $\phi(m,n) := 1$;
count := count + 1;
end
end

De regel (m,n) := argmax{ $|A(m',n')|^2 : \phi(m',n') = 0$} moet geïnterpreteerd worden als: Vind de indices (m,n) waarvoor geldt dat $|A(m,n)|^2$ maximaal is en waar $\phi(m,n) = 0$.

In het hier gegeven SSA is α_d afgerond naar boven. Dit afronden is niet persé noodzakelijk. Er kan gekozen worden voor een algoritme dat in de laatste stap een fractie van 1 aan een kernel-positie toekent. Er wordt zo een oplossing verkregen die een niet-strikt 1/0-masker oplevert. Zie hiervoor [BarJon94]. Voor AF's die radiaal niet-stijgend zijn is de oplossing optimaal want (11a) wordt gemaximaliseerd en aan (11b), (11c) en (11d) is voldaan.

Omdat we doorgaans AF's hebben die lokaal radiaal stijgende functies zijn, moeten we het verkregen algoritme aanpassen. Deze aanpassing bestaat uit een aanpassing van de input. Niet $|A(m,n)|^2$ wordt aangeboden maar een daaruit afgeleide radiaal niet-stijgende functie B(m,n). Als B(m,n) eenmaal verkregen is kan LP1 opgelost worden met het zojuist gepresenteerde algoritme. Wat nu rest is de beschrijving van de omzetting van $|A(m,n)|^2$ naar B(m,n). Deze omzetting wordt verkregen door lokaal radiaal stijgende delen van de AF te isoleren en de daarin aanwezige data als eenheid te zien. Iedere regio, in [BarJon94] aangeduidt als een supernode, die daarvoor in aanmerking komt krijgt nu een waarde die het gemiddelde is van alle in de supernode aanwezige elementen (supernodevalue (SNV)). De SNV is dus altijd kleiner of gelijk aan de grootste waarde van de elementen in de supernode. Zo worden de lokaal radiaal stijgende delen ingepast in de overige delen van de AF, net zolang tot een totale radiaal niet-stijgende functie B(m,n) ontstaat. Voor details en bewijzen van optimaliteit wordt verwezen naar [BarJon94].

Een optimale kernel wordt verkregen door eerst de $|A(m,n)|^2$ om te zetten in een B(m,n) en dan hierop het SSA los te laten. We kunnen de 2 stappen echter ook efficiënt combineren. Het resulterende algoritme heet Condensed Sort and Select Algorithm (CSSA). In pseudocode:

Input: $|A(m,n)|^2 \geq 0$,
Boom T met benaderde radiale paden,
$\alpha_d \geq 0$;

Output: $\phi(v,\tau)$;

Init: Voor alle triviale supernodes met index (m,n):
SNV(m,n) = $|A(m,n)|^2$, (supernodevalue)
n(m,n) = 1, (aantal elementen in supernode)
$\phi(m,n) = 0$;
Andere initialisaties:
count = 0, (volume teller)
$\phi(p(0,0)) = 1$;

Loop:
```
while(count<α_d) do
  Find: S := argmax{SNV(S'):φ(S') = 0};
  if (φ(p(S)) = 1) then
    Set: φ(S) := 1;
    count := count + n(S);
  else
      Voeg S en p(S) samen in een enkele supernode S en bereken gemiddelde -->
      SNV;
      n(S) := n(S) + n(p(S));
      Hou φ(S) = 0;
  end
 end
end
```

Een supernode wordt hier aangeduidt met S en kan meerdere knooppunten bevatten. Het aantal knooppunten in een supernode wordt bijgehouden door de variabele n. Bij het samenvoegen van supernodes S1 en S2 in een nieuwe supernode S, krijgen we een nieuwe SNV. Deze is gelijk aan:

$$snv(S) = \frac{snv(S_1)\,n(S_1) + snv(S_2)\,n(S_2)}{n(S_1) + n(S_2)}$$

terwijl het aantal knooppunten in de nieuwe supernode oploopt tot: n(S) = n(S1) + n(S2).

In het CSSA komt de statement p(S) voor. Deze statement is bedoeld voor het vinden van een voorganger (parent) van een supernode. voor het vinden hiervan is de eerder genoemde boomstructuur nodig die een benaderde representatie is van de radiale lijnen door de oorsprong van het (v,τ)-vlak. Voor de constructie hiervan is gebruik gemaakt van minimum L_2-normen, zie ook [BarJon93].

Het CSSA is geïmplementeerd met behulp van C++ programmatuur. De hiervoor benodigde source-files zijn te vinden in de bijlage A. Daar is tevens een toelichting te vinden over de gekozen implementatie methode. Verdere details over de programmatuur zijn te vinden in hoofdstuk 7.

5.3 Afvlakking van de optimale kernel

Wanneer een kernel met alleen énen en nullen gebruikt wordt bij de filtering van een WD, zal dit osillatorische verschijnselen opleveren in het (t,f)-vlak. Dit verschijnsel dat ook wel 'ringing' wordt genoemd, wordt veroorzaakt door de scherpe overgang in de kernel-functie van 1 naar 0 en is het sterkst voor kleine waarden van α. Om dit verschijnsel te voorkomen wordt de kernel-functie afgevlakt met behulp van een Gaussische functie:

$$g(k) = e^{(k^2/\sigma)}$$

Bij alle voorbeelden in dit verslag is gebruikt gemaakt van $\sigma = 1.5$.

6 Algoritme voor het bepalen van een MCE-distributie

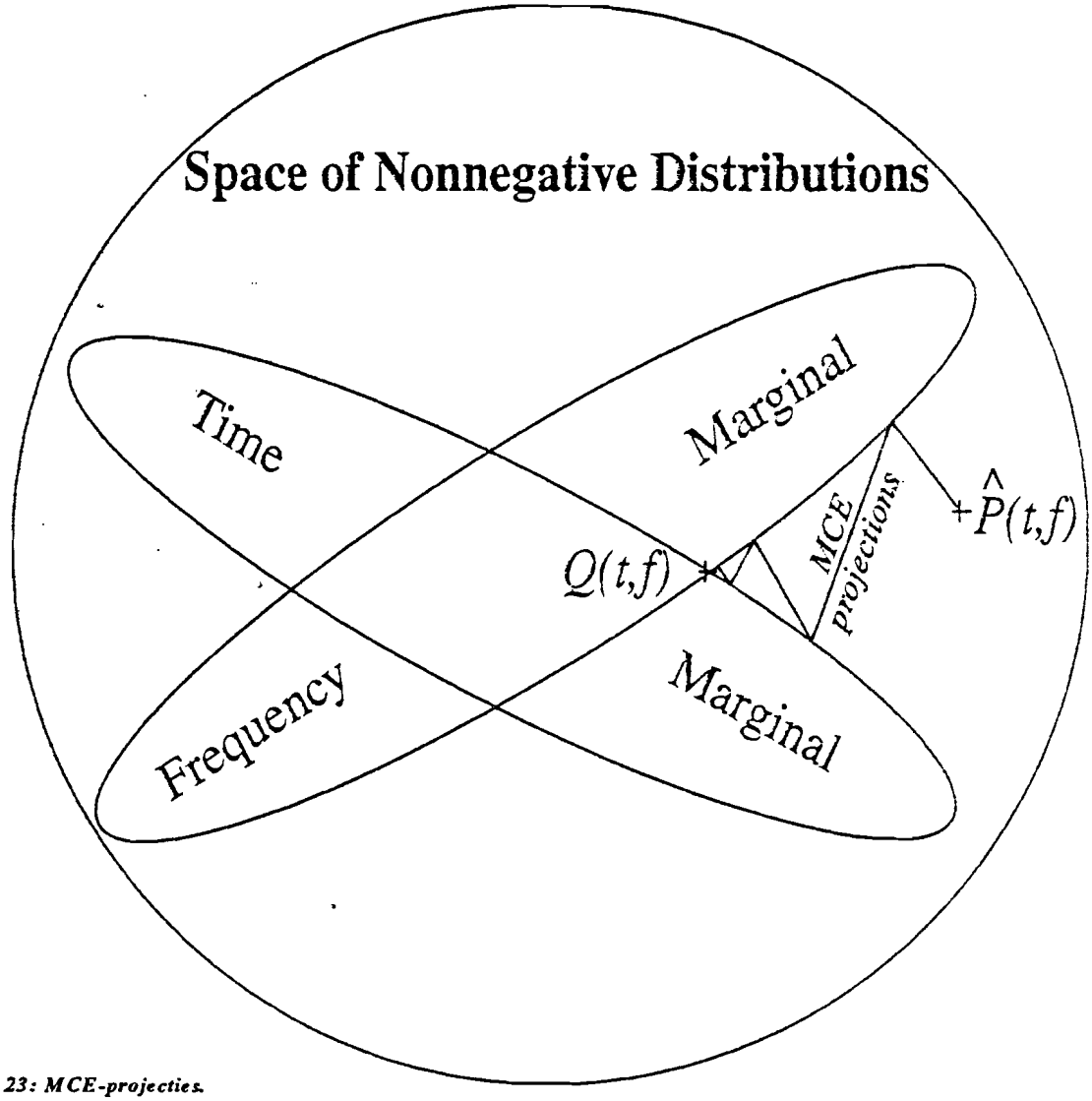
In §4.2 is aangegeven dat de MCE-distributie Q(t,f) gevonden kan worden door de Lagrange-multipliers $\Theta_1(t)$ en $\Theta_2(f)$ te berekenen:

$$Q_s(t,f) = P'(t,f)\, e^{-(\theta_1(t)+\theta_2(f))}$$

We beschrijven nu de manier om $\Theta_1(t)$ en $\Theta_2(f)$ te bepalen.

6.1 Iteratieve oplossing van de Lagrange-multipliers

Lagrange-multipliers en dus Q(t,f) kunnen gevonden worden door tussen de gestelde voorwaarden te itereren [Alsaka88], [Lough92(3)]. We illustreren dit aan de hand van onderstaande figuur:



Figuur 23: MCE-projecties.

We nemen eerst één van de marginaal-eisen in acht en projecteren de beginschatting P'(t,f) op de set met 'distributies die voldoen aan deze marginaal-eis. Vervolgens transformeren we dit resultaat door middel van een MCE-projectie naar de set van distributies die voldoen aan de andere marginaal-eis. Dit proces herhalen we tot convergentie optreedt.

De eerste iteraties berekenen we nu als volgt. Bij de nulde iteratie hebben we:

$$Q^{(0)}(t,f) = P'(t,f)\, e^{-\theta_1(t)}$$

Om $\Theta_1(t)$ op te lossen gebruiken we de tijd-marginaal-voorwaarde om te verkrijgen:

$$\int_{-\infty}^{\infty} Q^{(0)}(t,f)\, df = \int_{-\infty}^{\infty} P'(t,f)\, e^{-\theta_1(t)}\, df$$

$$m_1(t) = m'(t)\, e^{-\theta_1(t)}$$

$$\theta_1(t) = \ln[m'(t)/m_1(t)]$$

waarin $m_1(t)$ de gewenste tijd-marginaal is van de distributie en m'(t) de tijd-marginaal van P'(t,f). We verkrijgen nu dus:

$$Q^{(0)}(t,f) = P'(t,f)\,\frac{m_1(t)}{m'(t)}$$

Voor de volgende iteratie wordt $Q^{(0)}(t,f)$ gebruikt als de nieuwe 'begin' schatting. $Q^{(1)}(t,f)$ wordt verkregen via de frequentie-marginaal:

$$Q^{(1)}(t,f) = Q^{(0)}(t,f)\, e^{-\theta_2(f)}$$

Door van beide kanten van de vergelijking de frequentie-marginaal te berekenen kunnen we $\Theta_2(f)$ oplossen:

$$\int_{-\infty}^{\infty} Q^{(1)}(t,f)\, dt = \int_{-\infty}^{\infty} Q^{(0)}(t,f)\, e^{-\theta_2(f)}\, dt$$

$$m_2(f) = m'(f)\, e^{-\theta_2(f)}$$

$$\theta_2(f) = \ln[m'(f)/m_2(f)]$$

waarin $m_2(f)$ de gewenste frequentie-marginaal is van de distributie en m'(f) de frequentie-marginaal van $Q^{(0)}(t,f)$. Nu geldt:

$$Q^{(1)}(t,f) = Q^{(0)}(t,f)\frac{m_2(f)}{m'(f)}$$

Na deze iteratie voldoet $Q^{(1)}(t,f)$ aan de frequentie-marginaal, maar niet noodzakelijk meer aan de tijd-marginaal. We herhalen hiervoor de eerste stap, maar nu met prior $Q^{(1)}(t,f)$. Het alterneren tussen de tijd- en frequentie-marginalen gaat door tot aan een convergentiecriterium voldaan is. Als dit na M iteraties bereikt is, dan is $Q^{(M)}(t,f)$ de MCE-distributie Q(t,f). Het hier gekozen convergentiecriterium luidt:

$$\max(u)\,|m'(u)-m(u)| < 10^{-7}$$

met u de tijd- of frequentie-variabele, afhankelijk van de iteratie.

De convergentiesnelheid blijkt in de praktijk afhankelijk te zijn van het signaal, maar voor de meeste signalen wordt aan het criterium voldaan na circa 20 iteraties.

Het algoritme is geïmplementeerd met behulp van MatLab programmatuur. De corresponderende m-file is te vinden in bijlage C.

7 Programmatuur

Het grootste deel van de programmatuur is geschreven in de Matlab-language. Het hoofdprogramma tfd.m dient ook onder de Matlab prompt opgestart te worden. Na het opstarten van dit programma krijgt de gebruiker de keuze uit de volgende opties:

1) Berekening van een TFD uit de Cohen-klasse.
2) Berekening van OKTFD.
3) Berekening van een OKMCE-distibutie.

De programmatuur voor optie 1 is beschreven in [Tim94]. Na het maken van een keuze wordt de gebruiker gevraagd naar de lengte van het signaal (aantal monsters). Voor optie 2 en 3 geldt nu nog een maximum lengte van 64 punten. Deze beperking wordt veroorzaakt doordat er voor de constructie van een optimale kernel gebruik wordt gemaakt van een C++ programma dat gebonden is aan de DOS-grenzen en welke dus niet willekeurig grote data-structuren kan verwerken. Voor alle drie opties geldt dat de signaallengte een macht van 2 moet zijn. Wanneer de signaallengte bekend is, kan de gebruiker kiezen uit een aantal beschikbare signalen die opgeslagen zijn op schijf. Hiernaast bestaat de mogelijkheid dat de gebruiker een signaal genereert via de faciliteiten van Matlab. Na het vaststellen van de signalen moet voor optie 2 en 3 opgegeven worden hoe groot het gebruikte kernel-volume mag zijn, waarna de berekeningen voor de generatie van de uiteindelijke TFD volgen. Hierna kan gekozen worden voor verschillende presentatievormen, zoals een 3D-plot of een contourplot.

De interne organisatie van de programmatuur die behoort bij optie 2 en 3, is als volgt:

- Voor de berekening van een optimale kernel is een C++ programma geschreven. De communicatie met de buitenwereld verloopt voor dit programma via files. De files hebben een binair formaat dat voldoet aan de specificaties van een door Matlab weggeschreven matrix. Input van het C++ programma is een volumeparameter α_d en een halve AF-matrix $|A(m,n)|^2$. Een halve matrix is voldoende want er geldt $|A(m,n)|^2 = |A(-m,-n)|^2$. Output is een kernelfunctie bestaande uit een masker met énen en nullen. Het programma bestaat uit een aantal source- en include-files die, inclusief een nadere toelichting, te vinden zijn in bijlage A.
- Voor optie 2 is een m-file (Matlab-language) nodig voor de berekening van de Ambiguity functie, voor het aanroepen van het C++ programma en voor de afvlakking van de kernel. Deze m-file is te vinden in bijlage B.
- Voor optie 3 is een m-file nodig voor de constructie van een MCE-distributie. Zie hiervoor bijlage C. Daar is tevens een toelichting te vinden over de inhoud en het gebruik van het algoritme.

Alle Matlab-programmatuur is geschikt voor Matlab386. Bij gebruik van Matlab voor 486 zijn er waarschijnlijk een aantal aanpassingen nodig, met name op het gebied van de grafische presentatie.

8 Conclusies en aanbevelingen

Door het gebruik van signaalafhankelijke kernels is het mogelijk gebleken een tijd-frequentie distributie te maken die een minimum aan interferentie-termen heeft terwijl hierbij de resolutie van de signaal-termen behouden blijft. Deze resultaten worden verkregen door een optimale kernel te construeren die afgeleidt is van het signaal en die optimaal is volgens de in (9) aangelegde criteria. Hierdoor is de interferentie-term onderdrukking voor een brede klasse van signalen goed. Doordat de constructie van de optimale kernel met behulp van een algoritme geschiedt en dus automatisch is, is er geen voorkennis van het signaal nodig. De resulterende distributie wordt een Optimale-Kernel TFD (OKTFD) genoemd.

Bij signalen waarbij interferentie-termen en signaal-termen elkaar niet overlappen in het (v,τ)-domein, biedt de methode een betere oplossing vergeleken met TFD's uit de Cohen-klasse met een vaste kernel. Voor signalen waarbij dit wel het geval is bestaat er tot op heden nog geen enkele tijd-frequentie representatie die de signaal-termen behoudt, terwijl interferentie-termen onderdrukt worden.

Doordat de nadruk bij deze methode ligt op het onderdrukken van de interferentie-termen, gaan een aantal andere gunstige eigenschappen verloren. Met name het niet meer voldoen aan de marginaal-eisen is een nadeel.

Een oplossing voor dit probleem is gepresenteerd in de vorm van een MCE-methode. De voordelen van deze methode zijn:

- De resulterende distributie is positief.
- De distributie voldoet aan de marginalen.
- Het resultaat is uniek en optimaal voor de gegeven informatie.

Deze voordelen zorgen ervoor dat de distributie beter geïnterpreteerd kan worden als een energiedistributie van het signaal in tijd- en frequentie-domein.

Nadelen zijn:

- Het resultaat kan niet voor alle signalen en voor de gegeven voorwaarden in een analytische vorm gegeven worden.
- Bij een iteratief algoritme is de convergentie snelheid afhankelijk van het signaal.

Wel blijkt uit de praktijk dat het algoritme een rekentijd nodig heeft die in dezelfde orde grootte ligt als een berekening van een TFD uit de Cohen-klasse.

Door als beginschatting van de MCE-distributie een OKTFD te nemen, dat wil zeggen een TFD met een signaalafhankelijke kernel, zorgen we ervoor dat de

interferentie-termen zo weinig mogelijk invloed hebben op het uiteindelijke resultaat.

Voor verdere ontwikkelingen dient onderzocht te worden:

- De huidige algoritmen zijn blok georiënteerd en kunnen daarom alleen eindige signalen verwerken. Voor praktische toepassingen dient een venstertechniek gebruikt te worden zodat ook niet-eindige signalen verwerkt kunnen worden.
- Algoritmen kunnen versneld worden door direct een AF te berekenen in plaats van dit via de WD te doen.
- Het nu gebruikte C++ programma gebruikt veel geheugenruimte. Omdat dit programma onder DOS geschreven is, levert dit een beperking op van de signaallengte van 64 monsters. Door de datastructuren te optimaliseren met betrekking tot het geheugengebruik kan er zeker een factor twee verbetering verkregen worden in de maximale signaallengte. Dit zal echter aanzienlijk complexere code vereisen en zal eventueel ook ten koste gaan van bereken-tijd.
- De MCE-methode maakt nu gebruik van twee voorwaarden. Dit kan uitgebreid worden naar meerdere voorwaarden. Men kan bijvoorbeeld nog voorwaarden stellen betreffende de momentane frequentie en de group-delay. De volgende benadering voor het verkrijgen van een TFD met zoveel mogelijke gunstige eigenschappen lijkt interessant genoeg om nog onderzocht te worden:
  - Bereken de WD en leidt daar alle gunstige eigenschappen in numerieke vorm uit af, zoals er zijn: de tijd-marginaal, frequentie-marginaal, group-delay en de momentane frequentie. Voor het verkrijgen van deze data is het misschien nodig de oneven samples uit de discrete WD mee te nemen. Deze worden bij meeste toepassingen weggelaten, maar zijn wel belangrijk voor het verkrijgen van de eigenschappen [PeyPro86].
  - Bereken de OKTFD op de in dit verslag beschreven manier.
  - Forceer zoveel mogelijk eigenschappen in de TFD via de MCE-methode, door de genoemde eigenschappen als voorwaarden te stellen en de OKTFD als beginschatting te nemen.

  Het algoritme voor de MCE-methode bestaat nu uit een alternatie tussen de 4 voorwaarden totdat convergentie bereikt wordt. Voor een opsomming van alle gunstige eigenschappen van de WD zie [HlaBou92].

Voor het uitvoeren van bovenstaande opmerkingen kan goed gebruik gemaakt worden van de in dit verslag gepresenteerde programmatuur.

9 Referenties

[Alsaka88] Alsaka,Y.A. et al., "An efficient algoritm for implementing the relative entropy method," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing-ICASSP'88, 1988, pp.2384-2387.

[BarJon90] Baraniuk,R.G and D.L.Jones, "Optimal kernels for time-frequency analysis," in Proc. SPIE Int. Soc. Opt. Eng., vol. 1384, pp.181-187, 1990.

[BarJon91] Baraniuk,R.G and D.L.Jones, "A radially-Gaussian, signal-dependent time-frequency representation," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing ICASSP'91, 1991, pp.3181-3184.

[BarJon93] Baraniuk,R.G and D.L.Jones, "A signal-dependent time-frequency representation: Optimal kernel design," in IEEE Trans. Signal Processing, vol. 41, pp.1589-1602, Apr. 1993.

[BarJon94] Baraniuk,R.G and D.L.Jones, "A signal-dependent time-frequency representation: "Fast algorithm for optimal kernel design," in IEEE Trans. Signal Processing, vol.42,pp.134-146, jan. 1994.

[Cohen89] Cohen,L., "Time-frequency distributions- a review," in Proc. IEEE, vol. 77, pp.941-981, july 1989.

[CohPos85] Cohen,L. and T.E.Posch, "Positive time-frequency distribution functions," in IEEE Trans. Acoust., Speech, SignalProcessing, vol. 33, pp.31-37, feb. 1985.

[HlaBou92] Hlawatch F. and G.F.Boudreaux-Bartels, "Linear and quadratic time-frequency representations," in IEEE SignalProcessing Mag., vol. 9, pp.21-67, Apr. 1992.

[JeoWil92] Jeong,J and W.Williams, "Kernel design for reduced interference distributions," IEEE Trans.Sig.Proc., vol.40, no.2, pp.402-412, 1992.

[Jones92] Jones,D.L. and T.W.Parks, "A resolution comparison of several time-frequency representations," in IEEE Trans. Signal Processing, vol. 40, pp.413-420, Feb. 1992.

[Loug92(1)] Loughlin,P.J. et al., "An information theoretic approach to positive time-frequency distributions," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing-ICCASP'92, 1992, pp.V125-V128.

[Loug92(2)] Loughlin,P.J. et al., "Proper time-frequency energy distributions and the Heisenberg uncertainty principle," in Proc. IEEE-SP Int. Symp. Time-Frequency Time-Scale Analysis'92, pp.151-154, Oct 1992.

[Loug92(3)] Loughlin,P.J., "Time-frequency energy density functions: Theory and synthesis, Ph.D. dissertation, University of Washington, 1992.

[Loug93] Loughlin,P.J. et al., "Bilinear time-frequency representations: New insights and properties," in IEEE Trans. Signal Processing, vol. 41, pp.750-767, Feb. 1993.

[Oh90] Oh,S. et al., "Kernel synthesis for generalized time-frequency distributions using the method of projection onto convex sets," in Proc. SPIE Int. Soc. Opt. Eng., vol. 1348, pp.197-207, 1990.

[PeyPro86] Peyrin,P and R.Prost, "A unified definition for the discrete-time, disrete-frequency Wigner distributions." IEEE Trans. Acoust., Speech, Signal processing, vol. ASSP-34, pp858-867. 1986.

[ShoJon80] Shore,J.E. and R.W.Johnson, "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy," in IEEE Trans. Inf. Theory, vol. IT-26, pp.26-37, Jan. 1980.

[ShoJon81] Shore,J.E. and R.W.Johnson, "Properties of cross-entropy minimization," in IEEE Trans. Inf. Theory, vol. IT-27, pp.472-482, Jul. 1981.

[Tim94] Timmerman,P.H. "Berekening van tijd-frequentie distributies uit de Cohen-klasse.", stage verslag, vakgroep ESP, TU-Eindhoven, 1994.

Bijlage A: C++ source-files

In deze bijlage wordt een overzicht gegeven van de programmatuur die nodig is voor het berekenen van een optimale kernel-functie. Van iedere source-file wordt hier een korte beschrijving gegeven. Tenslotte volgen de uitdraaien van de C++ source-files.

matrix.cpp

Deze file bevat routines die behoren bij een matrix object. Er zijn routines voor het lezen en schrijven van een MatLab-matrix van en naar een bestand. Deze kunnen worden aangeroepen met behulp van de insertion- en extraction-operators. Tevens is er een differentiatie gemaakt voor matrices met integers en matrices met doubles (wat misschien efficiënter uitgevoerd kan worden met templates). Het is mogelijk matrices te gebruiken die groter zijn dan een segment (64k) omdat de index van de matrix geconstrueerd wordt met behulp van huge-pointers.

matrix.h

Deze include-file behorende bij matrix.cpp bevat de benodigde definities van de objecten IntMatrix en DblMatrix.

rtree.cpp

Deze file bestaat uit een routine voor het genereren van een boomstructuur die benaderde radiale paden op een rechthoekig rooster bevat. De boomstructuur is geïmplementeerd met behulp van een IntMatrix-object. rtree.h is de corresponderende include-file.

snode.cpp

Deze file bevat routines die behoren bij een SuperNode object. Hieronder bevinden zich routines voor het samenvoegen van 2 SuperNodes en voor het toekennen van kernel-volume. De corresponderende include-file is snode.h

oktfd.cpp

Dit is het hoofdprogramma. De volgende stappen worden uitgevoerd:

- Lees de matrix $|A(m,n)|^2$ uit het bestand amb.mat en ken deze toe aan de variabele amb.
- Lees het volume af uit amb(N/2+1,0), met N de signaallengte.
- Voer het CSSA algoritme uit gegeven amb en het volume.
- Retourneer de resulterende optimale kernel via een bestand genaamd optker.mat

Het hoofdprogramm maakt gebruik van alle eerder genoemde source-files.

matrix.h

```
//
// MATRIX.H
//
// Include-file for using classes Matrix, DblMatrix and IntMatrix
//
#if !defined( _MATRIX_H )
#define _MATRIX_H

#if !defined( _STRNG_H )
#include <strng.h>
#endif

#if !defined( _FSTREAM_H )
#include <fstream.h>
#endif

#define INTTYPE 30  //Matlab requirement
#define DBLTYPE  0  //Matlab requirement

enum ACCESS {REAL, IMAG, CPLX, NONE};
enum MTYPE {REALM, IMAGM, CPLXM};

typedef struct{
  long type;
  long mrows;
  long ncols;
  long imagf;
  long namlen;
} Fmatrix;

class Matrix {

protected:

     long rws;
     long cols;
     void far *preal;
     void far *pimag;

public:

     String* name;

     Matrix(long,long,int,MTYPE,String&);
     Matrix(int);
     ~Matrix();
     int bounds(long r,long c) {return !(r<0 || r>=rws || c<0 || c>=cols);}
     int access() {return accessflag;}
     int isReal() {return (accessflag == REAL);}
     long columns() {return cols;}
     long rows() {return rws;}
     friend ifstream& operator>>(ifstream&, Matrix&);
     friend ofstream& operator<<(ofstream&, Matrix&);

private:

     ACCESS accessflag;
     int elemsize;

     void checkheap(long);
};
```

```
class IntMatrix : public Matrix {

public:
  IntMatrix(long m,long n,String& s=String())
    : Matrix(m,n,sizeof(int),REALM,s) {}
  friend ifstream& operator>>(ifstream&, Matrix&);
  friend ofstream& operator<<(ofstream&, Matrix&);
  int far& operator()(long,long);
};


class DblMatrix : public Matrix {

int rwflag;   //indicates real or imaginary read/write
double value; //garbage collector for illegal write;

public:
    DblMatrix(long,long,MTYPE,String&);
    DblMatrix(): Matrix(sizeof(double)) {rwflag = REAL; value = 0.0;}
    friend ifstream& operator>>(ifstream&, Matrix&);
    friend ofstream& operator<<(ofstream&, Matrix&);
    int rwf(int rw) {rwflag  = rw; return rwflag;}
    double far& operator()(long,long);
};

#endif
```

matrix.cpp

```
//
// MATRIX.CPP
//
// Code for classes Matrix, DblMatrix and IntMatrix used for interfacing
// with MatLab via files.
//

#include <fstream.h>
#include <iostream.h>
#include <strng.h>
#include <string.h>
#include <stdlib.h>
#include <alloc.h>
#include "matrix.h"

Matrix::Matrix(long r,long c,int els,MTYPE type,String& nm=String())
{
long bytes;
  if (r<=0 || c<=0 || els <= 0) {
    cerr << "\nError in Matrix declaration\n";
    exit(1);
  }
  cols = c; rws = r;
  elemsize = els;
  accessflag = (ACCESS)type;
  bytes = els * r * c;
  name = new String(nm);  // always allocate space for string
  if (type == CPLX) checkheap(2*bytes);
    else checkheap(bytes);
  switch(type) {
    case IMAGM: pimag = farcalloc(bytes,sizeof(char));break;
    case CPLXM: pimag = farcalloc(bytes,sizeof(char));
    case REALM: preal = farcalloc(bytes,sizeof(char));
  }
}
```

```
Matrix::Matrix(int els = 0)
{
// now no allocation occurs yet for real and imag. part
  accessflag = NONE;
  cols = 0;
  rws = 0;
  elemsize = els;
  name = new String();  // always allocate space for string
}

Matrix::~Matrix()
{
  delete name;
  switch(accessflag) {
    case CPLX:farfree(pimag);
    case REAL:farfree(preal);break;
    case IMAG:farfree(pimag);
  }
}

void Matrix::checkheap(long n)
{
  if (farheapcheck() == _HEAPCORRUPT) {
    cerr << "\nfar heap is corrupt\n";
    exit(1);
  }
  if (farcoreleft() < n+32) {
    cerr << "\nout of memory\n";
    exit(1);
  }
}


int far& IntMatrix::operator()(long m,long n)
{
// overloading function call operator
  if (!bounds(m,n)) {
    cerr << "\nMatrix index out of bounds\n";
    exit(1);
  }
  // huge pointer necessary for large matrices
  return (int far&)((int huge*)preal)[m+n*rws];  // columnwise storage
}


DblMatrix::DblMatrix(long m,long n,MTYPE type,String& s = String())
  : Matrix(m,n,sizeof(double),type,s)
{
  rwflag = (type == IMAG) ? IMAG : REAL;
  value = 0.0;
}
```

```
double far& DblMatrix::operator()(long m,long n)
{
// overloading function call operator
// data is stored and retrieved columnwise
  if (!bounds(m,n)) {
    cerr << "\nMatrix index out of bounds\n";
    exit(1);
  }
  value = 0.0;
  switch(access()) {
  // huge pointers necess. for large matrices
    case CPLX:
      return (rwflag == REAL) ? (double far&)((double huge*)preal)[m+n*rws]
                              : (double far&)((double huge*)pimag)[m+n*rws];
    case REAL:
      return (rwflag == REAL) ? (double far&)((double huge*)preal)[m+n*rws]
                              : (double far&)value;
    case IMAG:
      return (rwflag == IMAG) ? (double far&)((double huge*)pimag)[m+n*rws]
                              : (double far&)value;
  }
}


ifstream& operator>>(ifstream& stream, Matrix& m)
{
// overloading extraction operator
Fmatrix fm;
int size;
long bytes;
  stream.read((char*) &fm, sizeof(fm));            // read header struct
  switch (fm.type) {
    case INTTYPE: size = sizeof(int); break;
    case DBLTYPE: size = sizeof(double); break;
    default: cerr << "\nillegal header struct\n"; exit(1);
  }
  bytes = fm.ncols * fm.mrows * size;
  if (m.elemsize) {                      // restriction on element size
    if(m.elemsize != size) {
      cerr << "\nIncompatible matrices\n";
      exit(1);
    }
  }
  if (m.accessflag != NONE) {          // matrixspace already allocated
    if (m.cols!=fm.ncols ||       // so check parameters
      m.rws!=fm.mrows || (!m.isReal())!=fm.imagf) {
      cerr << "\nIncompatible matrices\n";
      exit(1);
    }
  }
  else  // matrixspace not allocated yet
  {
    m.cols = fm.ncols;
    m.rws = fm.mrows;
    m.accessflag = REAL;
    m.elemsize = size;
    if (fm.imagf) m.checkheap(2*bytes);
      else m.checkheap(bytes);
    m.preal = farcalloc(bytes,sizeof(char));
    if (fm.imagf) {
      m.accessflag = CPLX;
      m.pimag = farcalloc(bytes,sizeof(char));
    }
  }
  char* buffer = new char[fm.namlen]; // buffer necessary?
  stream.read(buffer,(int)fm.namlen);
  *m.name = buffer;
```

```
    delete buffer;
    // read real data
    long i = 0;
    char c;
    while(stream && i<bytes) {
      stream.get(c);
      ((char huge*)m.preal)[i] = c;
      i++;
    }
    // read imag. data
    if (fm.imagf) {
      i = 0;
      while(stream && i<bytes) {
        stream.get(c);
        ((char huge*)m.pimag)[i] = c;
        i++;
      }
    }
    return stream;
}

ofstream& operator<<(ofstream& stream, Matrix& m)
{
// overloading insertion operator
Fmatrix fm;
long bytes;
  fm.ncols = m.cols;     // fill header struct
  fm.mrows = m.rws;
  fm.imagf = (m.accessflag == CPLX);
  fm.namlen = strlen((const char*)((String&)*m.name)) + 1;
    // does overloaded typecast operator only work on reference?
  switch(m.elemsize) {
    case sizeof(int): fm.type = INTTYPE; break;
    case sizeof(double): fm.type = DBLTYPE; break;
    default: cerr << "\nMatrix does not contain MatLab type\n"; exit(1);
  }
  bytes = fm.ncols * fm.mrows * m.elemsize;
  // write header struct
  stream.write((char*)&fm,sizeof(fm));
  // write name
  stream.write((const char*)(String&)*m.name,(int)fm.namlen);
  // write real data
  long i=0;
  char c;
  while(stream && i<bytes) {
    c = ((char huge*)m.preal)[i];
    stream.put(c);
    i++;
  }
  // write imag. data
  if (fm.imagf) {
    i = 0;
    while(stream && i<bytes) {
      c = ((char huge*)m.pimag)[i];
      stream.put(c);
      i++;
    }
  }
  return stream;
}
```

rtree.h

```
//
//  RTREE.H  enums en prototypes for usage radial tree
//

#if !defined( _RTREE_H )
#define _RTREE_H

#if !defined( _MATRIX_H )
#include "matrix.h"
#endif

enum RTCODE {
  ROOT    = 0x0,
  RFORK   = 0x18,
  LFORK   = 0x3,
  UPFORK  = 0x11,
  RVERT   = 0x1e,
  LVERT   = 0xf,
  UP      = 0x1b,
  RDIAG   = 0x1d,
  LDIAG   = 0x17,
  VRDIAG  = 0x1c,
  VLDIAG  = 0x7,
  UPRDIAG = 0x19,
  UPLDIAG = 0x13,
  STOP    = 0x1f
};

int smallestnorm(double,double,int,int,int,int,int,int,IntMatrix&);
IntMatrix& make_rtree(IntMatrix&);

#endif
```

rtree.cpp

```
//
// RTREE.CPP generatie van radial tree aan de hand van L2-norm
//

#if !defined( _MATRIX_H )
#include "matrix.h"
#endif

#if !defined( _RTREE_H )
#include "rtree.h"
#endif

#include <fstream.h>

int convert(int,RTCODE);   //private prototype

int smallestnorm (double d1,double d2,int r1,int c1,int r2,int c2,
                  int r, int c, IntMatrix& rtree)
{
  double a,b;
  int M,path;
  RTCODE k;
  M = rtree.rows();
  k = (RTCODE)rtree(r1-1,c1+M);
  if (k != LVERT && k != STOP) r1--;
  k = (RTCODE)rtree(r2,c2+M);
  if (k == LDIAG || k == STOP) r2--;
  c1++; c2++;
  if (r1==r2 && c1==c2) return (d1<d2) ? 1 : ((d2<d1) ? 2 : 0);
```

```
  a = r*c1 - r1*c; a *= a;
  d1 = d1 + a;
  b = r*c2 -r2*c; b *= b;
  d2 = d2 + b;
  path = smallestnorm(d1,d2,r1,c1,r2,c2,r,c,rtree);
  if (!path)  path = (a<b) ? 1 : ((b<a) ? 2 : 0);
  return path;
}

int convert(int k, RTCODE c)
{
  RTCODE r;
  switch(c) {
    case VLDIAG: r = (k==2) ? UPLDIAG : ((k==3) ? UPRDIAG : VRDIAG);break;
    case LDIAG: r = (k==2) ? LDIAG : ((k==3) ? RDIAG : RDIAG);break;
    case LVERT: r = (k==2) ? UP : ((k==3) ? UP : RVERT);break;
    case STOP: r = STOP;
  }
  return r;
}

IntMatrix& make_rtree(IntMatrix& rtree)
{
  int i,j,M,path1,path2;
  double d1,d2;
  RTCODE code;
  M = rtree.rows();
  rtree(0,M-1) = ROOT;
  for (j=-1;j>-M;j--) {
    path2 = (1==1);
    for (i=0;i<=-j;i++) {
      d1 = (j-1)*(i+1) - j*(i+1);
      d1 *= d1;
      d2 = (j-1)*i - j*(i+1);
      d2 *= d2;
      path1 = (smallestnorm(d1,d2,i+1,j,i,j,i+1,j-1,rtree) == 2);
      if (path1 && path2) code = VLDIAG;
      if (path1 && !path2) code = LDIAG;
      if (!path1 && path2) code = LVERT;
      if ((!path1 && !path2) || (j==(1-M))) code = STOP;
      rtree(i,j+M-1) = code;
      if (i!=(M-2)) rtree(-j,M-1-i) = convert(2,code);
      if (i!=(M-1)) rtree(-j,M-1+i) = convert(3,code);
      if (j!=(1-M)) rtree(i,M-1-j) = convert(4,code);
        else rtree(i,2*M-3) = STOP;
      path2 = !path1;
    }
  }
  rtree(1,M-2) = LFORK;
  rtree(1,M) = RFORK;
  rtree(2,M-1) = UPFORK;
  rtree(M-1,1) = STOP;          // close hole
  return rtree;
}
```

snode.h

```
//
// SNODE.H
//

#if !defined( _SNODE_H )
#define _SNODE_H

#if !defined( _SORTABLE_H )
#include <sortable.h>
#endif

#if !defined( _CLSTYPES_H )
#include <clstypes.h>
#endif

#if !defined( _MATRIX_H )
#include "matrix.h"
#endif

class Coord {

public:
  Coord* next;
  unsigned char row;
  unsigned char column;

  Coord(unsigned char r, unsigned char c,Coord* n)
    {row = r; column = c; next = n;}
  int isEqual(Coord* c)
    {return ((row==c->row) && column==c->column);}
  int isLessThan(Coord* c)
    {return (row!=c->row) ? (row < c->row) : (column < c->column);}
  ~Coord() {}
};

class SuperNode;

class ChildRec {

public:

  ChildRec* next;
  SuperNode* akid;

  ChildRec(ChildRec* c,SuperNode* s) {next = c; akid = s;}
  ~ChildRec() {}
};


class SuperNode : public Sortable {

protected:

  enum {SNodeClass = __firstUserClass +1};

private:

  int n;
  char notzero;

public:
  double snv;
  SuperNode* parent;
  Coord* xys;
  ChildRec* kids;
```

```
   SuperNode(double,SuperNode*,unsigned char,unsigned char);
   ~SuperNode();
   virtual classType isA() const {return SNodeClass;}
   virtual char* nameOf() const {return "SuperNode";}
   virtual hashValueType hashValue() const {return 0;}
   virtual int isEqual(const Object& s) const;
   virtual int isLessThan(const Object& s) const;
   virtual void printOn(ostream& os) const
     { os << snv << " " << n << " " << notzero;}
   void condense(SuperNode&);
   IntMatrix& setkernel(IntMatrix&,unsigned int);
   char kernelnotzero() {return notzero;}
   char parentkernelnotzero();
   int size() {return n;}
   void assignkid(SuperNode*);
};


#endif
```

snode.cpp

```
//
// SNODE.CPP
//
//
#if !defined( _MATRIX_H )
#include "matrix.h"
#endif

#if !defined( _SNODE_H )
#include "snode.h"
#endif

#include <checks.h>

SuperNode::SuperNode(double v,SuperNode* p,unsigned char r,unsigned char c)
{
  PRECONDITION(v>=0);
  snv = v;
  parent = p;
  n = 1;
  notzero = (1==0);
  xys  = new Coord(r,c,NULL);
  kids = NULL;
}

SuperNode::~SuperNode()
{
  ChildRec* p;
  // first delete all kids and corresponding ChildRec's
  while(kids) {
    delete kids->akid;
    p = kids;
    kids = kids->next;
    delete p;
  }
  Coord* q;
  // now delete all Coord structs;
  while(xys) {
    q = xys;
    xys = xys->next;
    delete q;
  }
}
```

```
int SuperNode::isLessThan(const Object& t) const
{
  int r;
  SuperNode& s = (SuperNode&)t;
  r = ((snv < s.snv) || ((snv == s.snv) && xys->isLessThan(s.xys)));
  return r;
}

int SuperNode::isEqual(const Object& t) const
{
  SuperNode& s = (SuperNode&)t;
  return ((snv == s.snv) && xys->isEqual(s.xys));
}

void SuperNode::condense(SuperNode& s)
{
  // *this always has at least 1 kid and 1 Coord record
  // *this is parent of *s
  PRECONDITION(xys);
  PRECONDITION(kids);
  PRECONDITION(this == s.parent);
  snv = (snv * n + s.snv * s.n) / (n + s.n);
  n = n + s.n;
  // append Coord structs;
  Coord *p,*q;
  p = xys;
  while(p) {
    q = p;
    p = p->next;
  }
  q->next = s.xys;
  s.xys = NULL;
  ChildRec *u,*v;
  // delete the ChildRec with pointer to condensed child
  v = u = kids;
  while ((SuperNode huge*)(u->akid) != (SuperNode huge*)&s) {
    v = u;
    u = u->next;
    CHECK(u);
  }
  if (u==kids) kids = u->next;
    else v->next = u->next;
  delete u;
  // append kids, kids may be NULL now
  v = u = kids;
  while(u) {
    v = u;
    u = u->next;
  }
  if (v) {
    v->next = s.kids;
    v = v->next;
  }
  else v = kids = s.kids;
  s.kids = NULL;
  // notify kids that parent changed its address
  while(v) {
    (v->akid)->parent = this;
    v = v->next;
  }
  // now delete condensed child
  delete &s;
}
```

```
IntMatrix& SuperNode::setkernel(IntMatrix& m,unsigned int v)
{
  Coord* p = xys;
  notzero = (1==1);
  int i = (v>n) ? n : v;
  while(p && i>0) {
    m(p->row,p->column) = 1;
    p = p->next;
    i--;
  }
  return m;
}

char SuperNode::parentkernelnotzero()
{
  if (parent) return parent->kernelnotzero();
    else return (1==1);
}


void SuperNode::assignkid(SuperNode* s)
{
  kids = new ChildRec(kids,s);
}
```

oktfd.cpp

```
//
// OKTFD.CPP main program for generating optimal kernel
// this method uses an excessive amount of memory!
//

#include <btree.h>
#include <fstream.h>
#include <checks.h>
#include <strng.h>
#include <stdlib.h>
#include <new.h>
#include <alloc.h>
#include "matrix.h"
#include "rtree.h"
#include "snode.h"

#define __DEBUG 2
#define ORDER   3
#define L       0x10  // masks for determining direction in fillbtree
#define LD      0x8
#define U       0x4
#define RD      0x2
#define R       0x1

// prototypes

SuperNode* fillbtree(unsigned char,unsigned char,IntMatrix&,
                     SuperNode*,SuperNode*,DblMatrix&,Btree&);
IntMatrix& oktfd(unsigned int,IntMatrix&,Btree&);

SuperNode* fillbtree(unsigned char r,unsigned char c,IntMatrix& rt,
                     SuperNode* s,SuperNode* p,DblMatrix& af,Btree& bt)
{
  SuperNode* a;
  s = new SuperNode(af(r,c),p,r,c);
  bt.add(*s);
  RTCODE code = (RTCODE)rt(r,c);
  if (code != STOP) {
    if (L &  ~code) {
```

```
      a = fillbtree(r,c-1,rt,a,s,af,bt);
      s->assignkid(a);
    }
    if (LD &  ~code) {
      a = fillbtree(r+1,c-1,rt,a,s,af,bt);
      s->assignkid(a);
    }
    if (U &  ~code) {
      a = fillbtree(r+1,c,rt,a,s,af,bt);
      s->assignkid(a);
    }
    if (RD &  ~code) {
      a = fillbtree(r+1,c+1,rt,a,s,af,bt);
      s->assignkid(a);
    }
    if (R &  ~code) {
      a = fillbtree(r,c+1,rt,a,s,af,bt);
      s->assignkid(a);
    }
  }
  return s;
}


IntMatrix& oktfd(unsigned int volume,IntMatrix& kernel,Btree& bt)
{
  unsigned int max,vcount = 0;
  SuperNode *p, *q;
  while(vcount < volume) {
    max = bt.getItemsInContainer() - 1;
    p = &((SuperNode &)(bt[max]));
    CHECK(*p!=NOOBJECT);
    CHECK(bt.hasMember(*p));
    bt.detach(*p);
    CHECK(!bt.hasMember(*p));
    if(p->parentkernelnotzero()) {
      p->setkernel(kernel,volume-vcount);
      vcount += p->size();
    }
    else {
      q = p->parent;
      CHECK(bt.hasMember(*q));
      bt.detach(*q);
      CHECK(!bt.hasMember(*q));
      q->condense(*p);  // *p is destroyed!
      bt.add(*q);
      CHECK(bt.hasMember(*q));
    }
  }
  return kernel;
}


void trap_allocation_failures()
{
  cerr << "\nMemory allocation failure using new\n";
  exit(1);
}
```

```
void main()
{
  set_new_handler(trap_allocation_failures);

 // cout << farcoreleft() << "\n";
  {
  const char AMB [] = "amb.mat";
  const char KER [] = "optker.mat";
  SuperNode* root;
  unsigned int volume;
  int r,c;

  ifstream in_file(AMB,ios::binary);
  if (!in_file) {
      cerr << "\nCouldn't open file " << AMB << "\n";
      return;
  }

  Btree& bt = *(new Btree(ORDER));
  bt.ownsElements(0);

  DblMatrix& amb = *(new DblMatrix());
  in_file >> amb;
  r = (int)amb.rows()-1;
  c = (int)amb.columns();

  CHECK(2*(r-1)==c);

  volume = (unsigned int)amb(r,0);

  IntMatrix& kernel = *(new IntMatrix(r,c,String("kernel")));
  IntMatrix& rtree = *(new IntMatrix(r,c,String("rtree")));

  rtree = make_rtree(rtree); // rtree as a class?
  root = fillbtree(0,r-1,rtree,root,NULL,amb,bt);
  delete &amb;
  delete &rtree;

  ofstream out_file(KER,ios::binary);
  if (!out_file) {
    cerr << "\nCouldn't open file " << KER << "\n";
    return;
  }
  out_file << oktfd(volume,kernel,bt);

  delete &kernel;
  delete &bt;
  ChildRec *p;
/*  while(root->kids) {
    delete root->kids->akid;
    p = root->kids;
    root->kids = root->kids->next;
    delete p;
  } */
  delete root;
/*  struct farheapinfo hi;
  hi.ptr = NULL;
  while(farheapwalk(&hi) == _HEAPOK)
    cout << hi.size << " " << (hi.in_use ? "used\n" : "free\n");
  */
  }
  //cout << farcoreleft() << "\n";
}
```

Bijlage B: m-file voor OKTFD

Deze bijlage bevat 1 enkele uitdraai van de m-file oktfd2.m. Deze MatLab-file bevat de volgende instructies:

- Bereken uitgaande van de gegeven Wigner distributie het modulus-kwadraat van de Ambiguity functie uit.
- Schrijf de resulterende matrix weg onder de naam amb.mat.
- Roep het C++ programma aan voor de berekening van de optimale kernel.
- Lees de optimale kernel uit het bestand optker.mat.
- Vlak de kernel, die bestaat uit énen en nullen, af door de kernel 2-dimensionaal te convolueren met een Gaussische functie.

De resulterende kernel wordt vervolgens geretourneerd.

Op de volgende pagina volgt een uitdraai van de file oktfd2.m.

oktfd2.m

```
function [kernel] = oktfd2(W,v);

% Function for calculation of optimal kernel
% W is Wigner distribution
% v is volume parameter

[N,dim] = size(W);
W = [W(N/2+1:N,:);W(1:N/2,:)];

% get ambiguity function
A = fftshift(fft(ifft(W).').');

% take absolute square and assign volume
Asqr = zeros(N/2+2,N);
Asqr(1:N/2+1,:) = abs(A(N/2+1:-1:1,:)).^2;
v = floor(N*v/4);   %transform continuous volume to discrete volume
Asqr(N/2+2,1) = v;

% calculate optimal 1/0 kernel
save amb Asqr;
!oktfd;
load optker;

% mess around to get proper kernel
upker(1:N/2+1,:) = kernel(N/2+1:-1:1,:);
lowker(1:N/2-1,2:N) = kernel(2:N/2,N:-1:2);
lowker(1:N/2-1,1) = zeros(N/2-1,1);
kernel = [upker;lowker];

% smooth 1/0 kernel for preventing ringing in TFD

M = 1; sigma = 1.5;

k = -M:M;
g = exp(-k.^2/sigma);
G = g'*g;
G = G/sum(sum(G));
kernel = conv2(G,kernel);
kernel = kernel(M+1:N+M,M+1:N+M);
kernel = fftshift(kernel);

return;
```

Bijlage C: m-file voor OKMCE distributie

Deze bijlage bevat 1 enkele uitdraai van de m-file mce.m. Deze MatLab-file bevat de volgende instructies:

- Neem van een gegeven a priori distributie alleen het positieve deel.
- Geef deze beginschatting een offset van 1%. Dit wordt gedaan om de convergentiesnelheid te vergroten.
- Normaliseer de beginschatting zodanig dat de totale energie gelijk is aan 1.
- Bereken de gewenste marginalen uitgaande van het gegeven signaal. Geef ook deze een (zeer kleine) offset. Deze offset is nodig om het delen door nul te voorkomen. Normaliseer vervolgens ook de marginalen.
- Start met het itereren tussen de twee marginaal-voorwaarden. De cyclus wordt afgebroken als er voldaan wordt aan het convergentiecriterium of als er 100 iteraties gedaan zijn. In het laatste geval hebben we geen MCE-distributie.

De resulterende a posteriori distributie wordt vervolgens geretourneerd.

Op de volgende pagina volgt een uitdraai van de file mce.m.

mce.m

```
function [Q] = mce(N,x,Q)

epsilon = 1e-7;

Q = real(Q.*(real(Q>0)));   % only positive part

Q = Q - min(min(Q));
offset = 1e-2*max(max(Q));  % for improving convergence
Q = Q + offset;

Q = Q/sum(sum(Q));          % normaliziation

% calculate desired marginals and take aliasing into acount

Tm = abs(x).^2;
offset = 1e-7*max(Tm);
Tm = Tm + offset;

Fm = abs(fftshift(fft([zeros(N/4,1);x(1:2:N-1);zeros(N/4,1)]))).^2;
offset = 1e-7*max(Fm);
Fm = Fm + offset;

Tm = Tm'/sum(Tm);           % normalization
Fm = Fm'/sum(Fm);           % normalization


% calculate marginals of Q and convergence criteria
T = sum(Q);
condT = max(abs(T-Tm)) > epsilon;
F = sum(Q');
condF = max(abs(F-Fm)) > epsilon;

%loop
iter = 0;
while ((condT | condF) & iter<100),
  iter = iter + 1;
  if condF,
    mul = Fm./F;
    for i = 1:N,
      Q(:,i) = Q(:,i).*(mul.');
    end
  end
  T = sum(Q);
  condT = max(abs(T-Tm)) > epsilon;
%  disp(['T: ',num2str(iter),'    ',sprintf('%15f',(max(abs(T-Tm))))]);
  if condT,
    mul = Tm./T;
    for i = 1:N,
      Q(i,:) = Q(i,:).*mul;
    end
  end;
  F = sum(Q.');
  condF = max(abs(F-Fm)) > epsilon;
%  disp(['F: ',num2str(iter),'    ',sprintf('%15f',(max(abs(F-Fm))))]);
end
```