

Project wavelets

Matthias Baeten & Bob Vergauwen

13 januari 2016

1 Ruisreductie

1.1 Academisch voorbeeld zonder ruis

Bij wijze van opwarming starten we met de wavelet decompositie van de functie $\mathbb{R} \rightarrow \mathbb{R} : x \mapsto \exp(x)$. Dit is een gladde functie die bovendien analytisch is. Voor onze analyse werd de exponentiële functie equidistant bemonster op het interval $[0, 1]$ met 256 punten. Deze data werd nadien geanalyseerd met behulp van 3 verschillende wavelet transformaties, de haar wavelet, de daubechies wavelet van orde 2 en de daubechies wavelet van orde 45. Elk van deze transformaties werd uitgevoerd tot niveau 4, dit maakt dus dat het signaal zal worden opgesplitst ten opzichte van 5 verschillende basissen. De resultaten van dit experiment zijn samen gevat in Figuur 1. In de linker kolom van de figuur zijn de coefficienten van de transformatie uitgezet. In de rechter kolom is telkens de benadering van de exponentiële functie in elke basis uitgezet. Hierbij is de onderste curve de benadering in W_1 , die daar boven de benadering in W_2 en zo voort. De bovenste grafiek is dan de benadering van de exponentiële functie in de ruimte V_4 .

Wat meteen opvalt is dat de coefficienten van de lage frequenties (links in de coefficienten vector) het grootst zijn. Dit is volledig volgens de verwachting, de exponentiële functie is een gladde functie en bevat dus voornamelijk lage frequenties. Een tweede bemerking is dat voor de hogere orde wavelets de coefficienten aan de randen groter worden. Dit is het gevolg van het breder worden van de wavelet, hierdoor zal het eind effect verstrekken worden.

Vervolgens kijken we naar de convergentie van de wavelet benadering. Hier is het duidelijk dat een hogere orde benadering niet meteen een snellere convergentie oplevert. Dit is opnieuw het gevolg van het bredere karakter van de hogere orde wavelets. Over het algemeen is de beste benadering bekomen door de daubechies wavelet van orde 2. Het eind effect is het kleinste voor de haar wavelet.

1.2 Academisch voorbeeld met ruis

In een tweede test word er ruis toegevoegd aan de gladde functie exponentiële functie. Deze ruis is witte ruis met een standaard afwijking van 0.1. Om de invloed van de ruis op de wavelet coefficienten duidelijk te maken zijn de coefficienten weergegeven in figuur 2. De invloed van de witte ruis in het tijddomein geeft een verstoring van witte ruis op de coefficienten van de verschillende wavelet transformaties. De verstoring kan makkelijk worden verwijderd aan de hand van een threshold waarder te gebruiken. Deze methode is

besproken in de opgaven en zal dus niet verder worden toegelicht. Enkel de resultaten en toepassingen zullen worden besproken.

De fout als functie van de threshold waarden is weergeven in figuur 3. Hierbij zijn drie verschillende wavelet transformaties gekozen, de haar wavelet, daubechie 2 en daubechie 45. Voor de threshold functies hebben we gekozen om drie verschillende functie voorschriften te bestuderen, een zacht, harde en continue threshold functie. Een plot van deze drie threshold functies is gegeven in Figuur 5. Uit deze drie figuren is het duidelijk dat er een fundamenteel verschil optreed tussen de zachte threshold functie en de twee andere. De verklaring hiervoor is dat de zachte threshold functie elke waarden zal wijzigen, zelfs de waarden ver boven de threshold. Dit kan men inzien door de drie threshold functies met elkaar te vergelijken in Figuur 5.

Om dit deel af te sluiten is in figuur 4 de optimale ruis reductie weergegeven. Deze reducties maakt gebruik van daubechie wavelet van orde 2 en een threshold waarden van 0.4 met de gladde treshold functie.

1.3 Tussenliggende waarden bepalen

De tussenliggende waarden kunnen op een aantal manieren worden bepaald. Een eerste is aan de hand van een interpolatie die gebruik maakt van de gefilterde punten.

Het is echter ook mogelijk om de tussenliggende waarden rechtstreeks te bepalen aan de hand van de wavelet coefficienten. Dit is enkel mogelijk wanneer men werkt met een continue wavelet transformatie zoals bijvoorbeeld de Meyer wavelet.

De details van deze wavelet interpolatie zijn hier niet volledig uitgewerkt maar eenmaal de functie beschrijving van de continue wavelet gekend is en de wavelet coördinaten zijn bepaald is het bedenken van een interpolatie schema niet veel werk meer.

1.4 Moving on to images

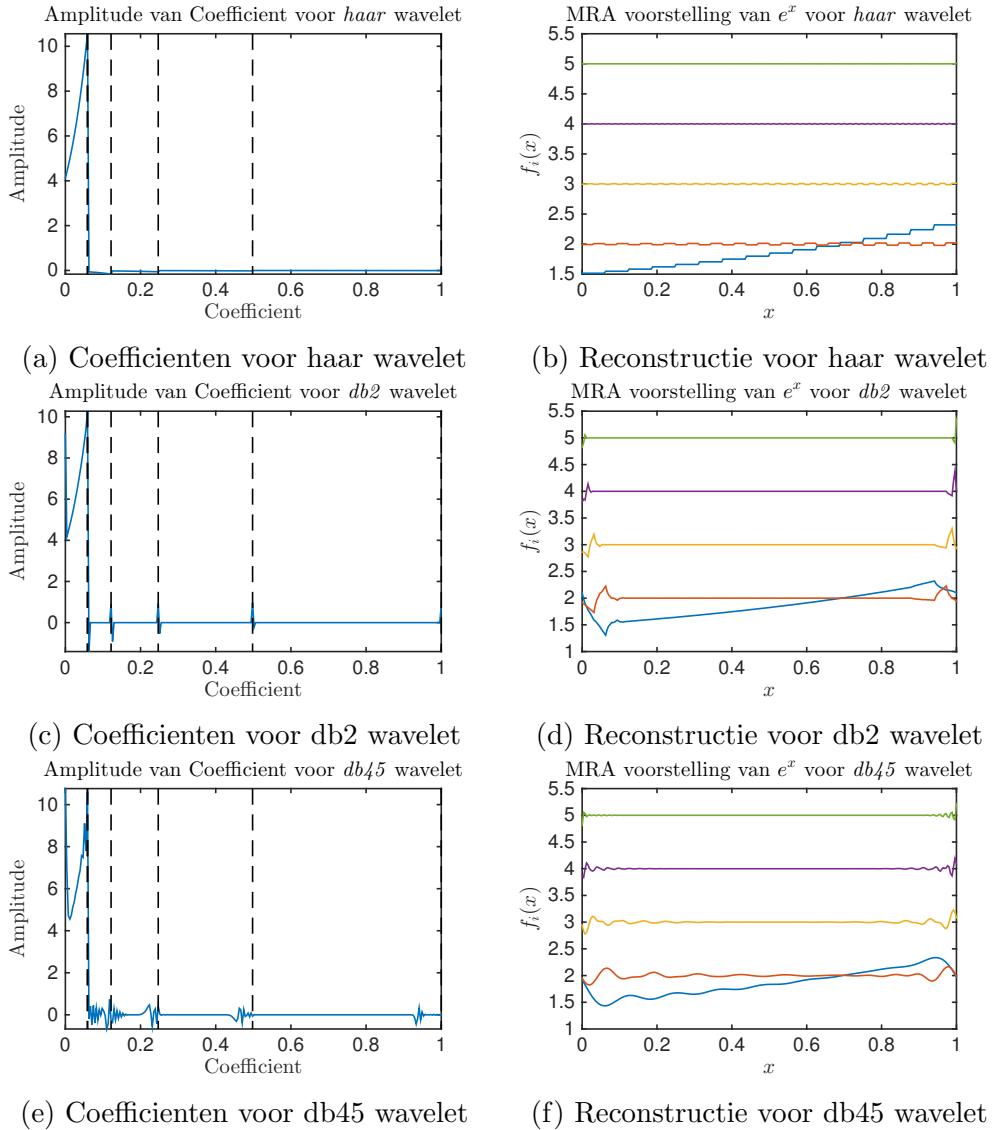
Voor de ruis experimenten op afbeelding is er steeds met eenzelfde ruis bron gewerkt. De ruis was gaussische verdeeld met een standaard afwijking van 0.1. Verder werd de ingelezen afbeelding eerst genormaliseerd. Op deze manier is het eenvoudiger resultaten tussen verschillende afbeeldingen te vergelijken en een normalisatie komt ook meestal ten goede van de conditionering van het probleem.

1.4.1 Implementatie van ruis reductie algoritme

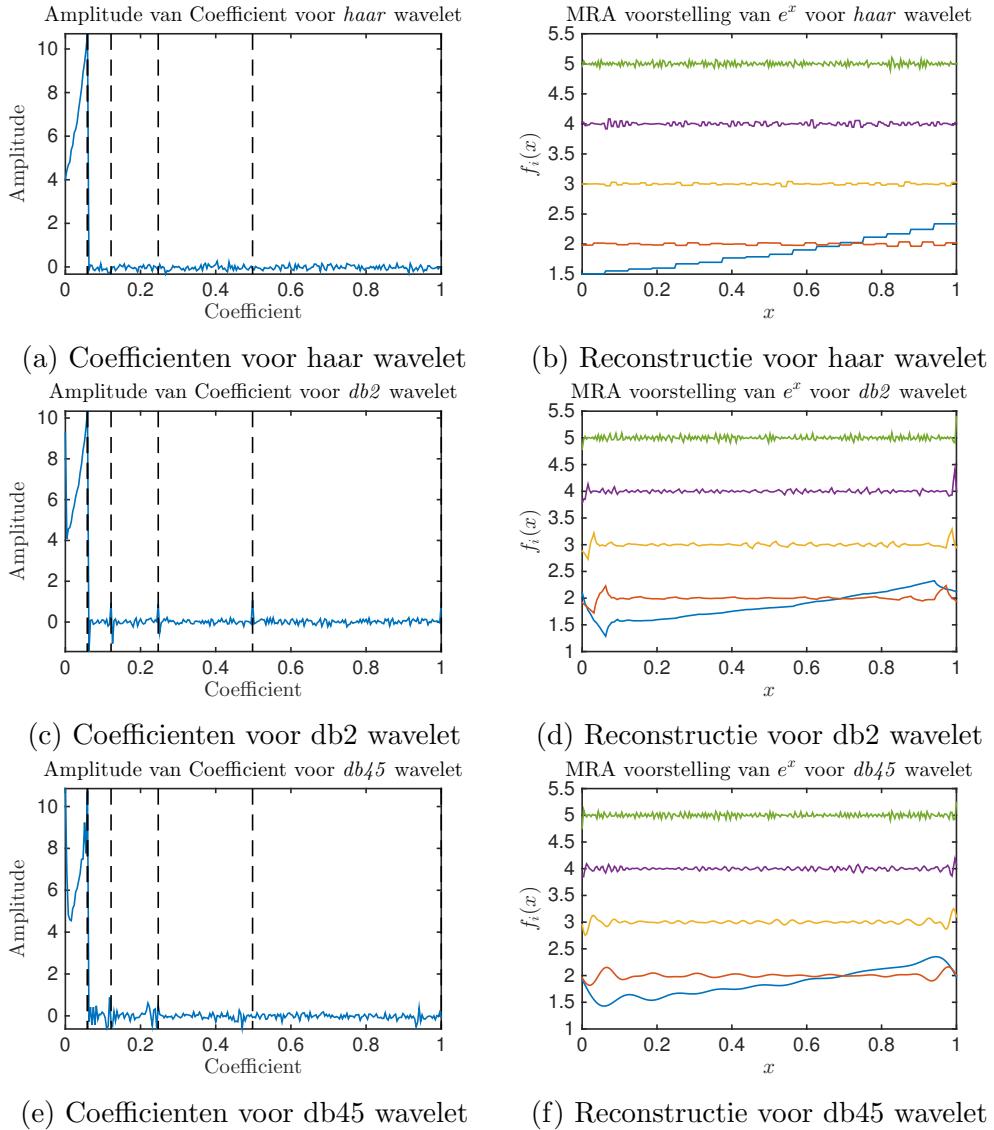
De eenvoudigste manier voor ruis uit een afbeelding te halen aan de hand van een wavelet transformatie is door exact dezelfde strategie toe te passen als in het 1 dimensionaal geval. Dit houdt in dat eerst de wavelet coefficienten worden bepaald voor de ruisige afbeelding. Nadien worden deze coefficienten met een threshold functie op een niet lineaire manier gefilterd. De laatste stap is dan de afbeelding reconstrueren aan de hand van de gefilterde coefficienten. Een concrete implementatie van dit algoritme is terug te vinden in de appendix.

1.4.2 Verschil in threshold functies

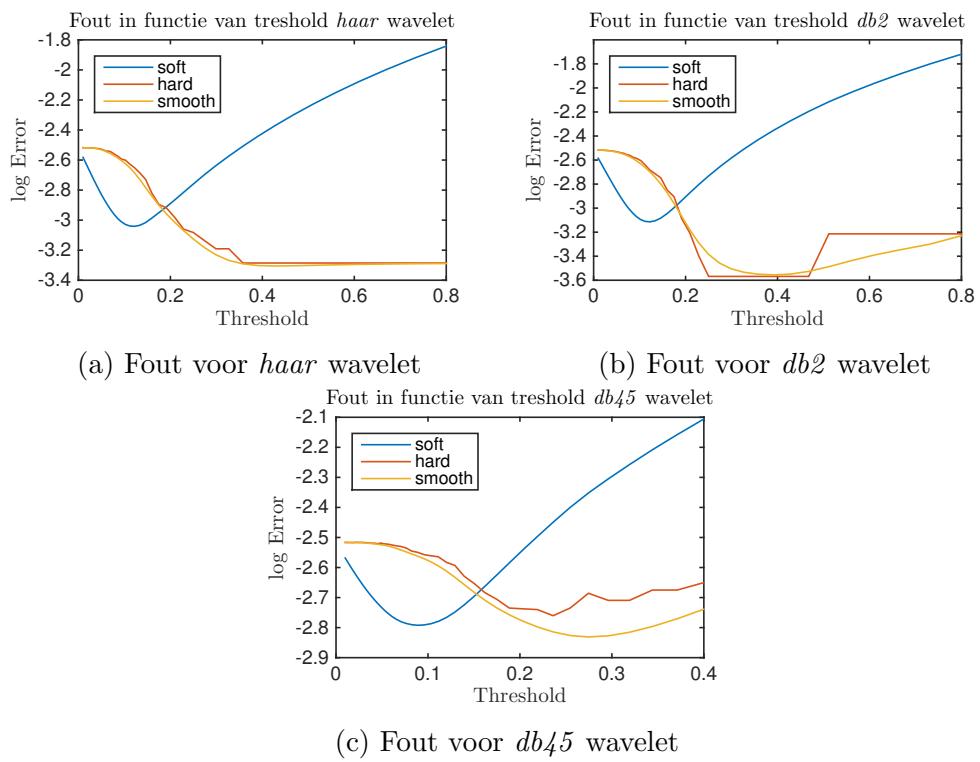
Voor een goed beeld te krijgen van de invloed van de threshold functie op de ruisreductie hebben we de kwaliteit van de ruisreductie vergeleken voor de verschillende threshold



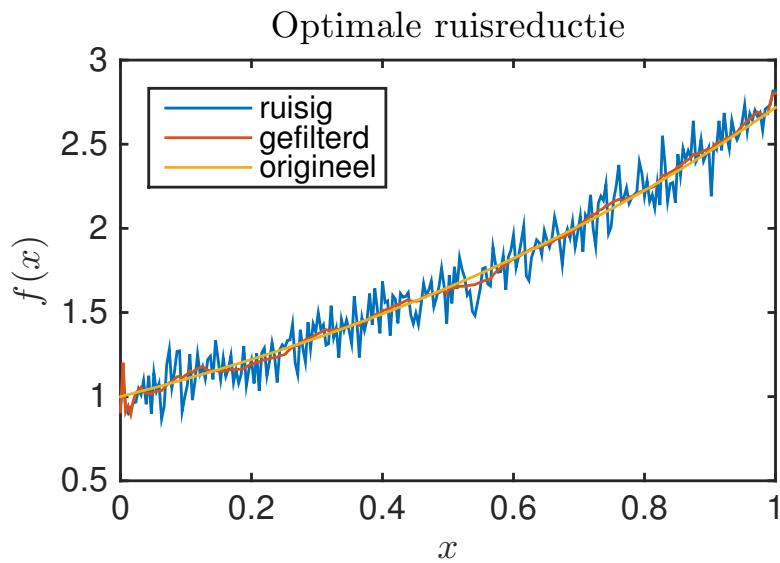
Figuur 1: Coefficienten van wavelet transformatie van de exponentiële functie voor verschillende wavelets samen met de benadering in elke vectorruimte. Elke transformatie is tot niveau 4 uitgevoerd. De stippellijnen in de linker figuur markeren de verschillende coördinaten vectoren.



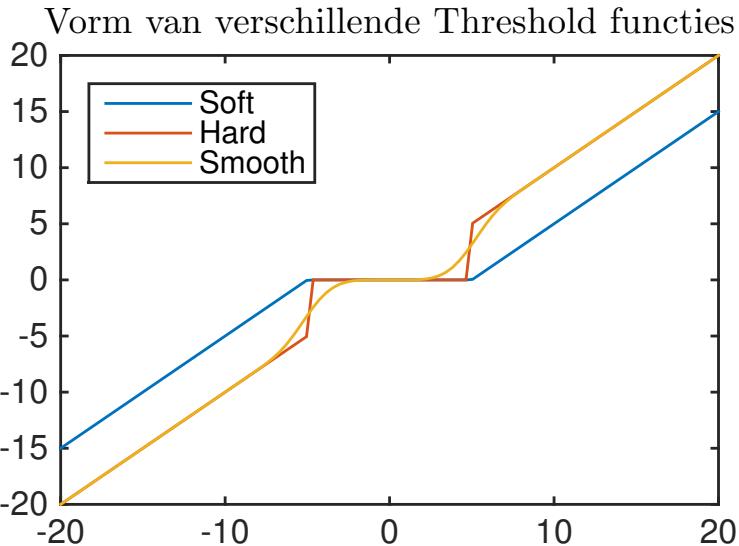
Figuur 2: Coefficienten van wavelet transformatie van de ruizige exponentiële functie voor verschillende wavelets samen met de benadering in elke vectorruimte. Elke transformatie is tot niveau 4 uitgevoerd. De stippellijnen in de linker figuur markeren de verschillende coördinaten vectoren.



Figuur 3: Log van de benaderingsfout in functie van de threshold voor drie verschillende threshold functies. De benadering werd uitgevoerd met een haar wavelet tot niveau 6.



Figuur 4: Optimale ruis reductie van de exponentiële functie. Deze reducties maakt gebruik van daubechies wavelet van orde 2 tot niveau 6 en een threshold waarden van 0.4 met de zachte threshold functie.



Figuur 5: Plot van de verschillende threshold functies. Het is duidelijk dat de zachte threshold functie elke waarden zal wijzigen. De grote van deze wijziging hang af van de threshold waarden. Deze invloed leid vaak tot mindere goede resultaten van deze threshold functie.

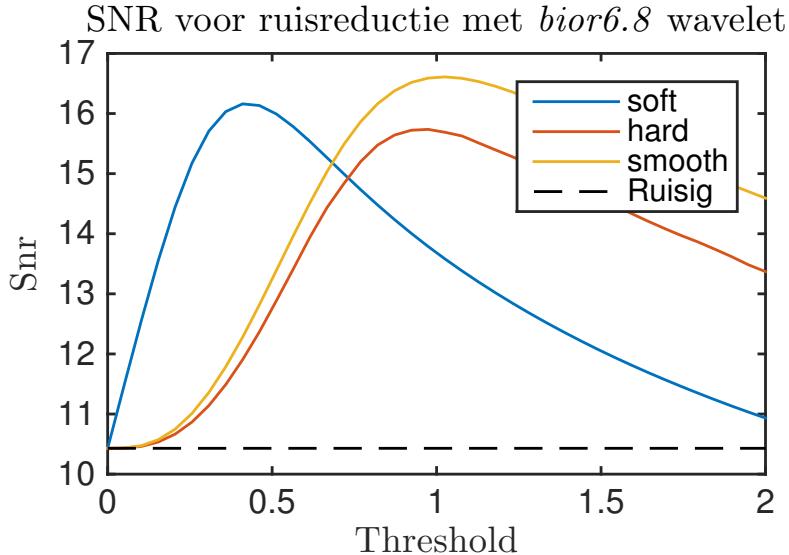
functies. Voor elke threshold functie werden er een aantal threshold parameters getest. Een voorbeeld resultaat van zo een test is te zien in Figuur 6. In deze figuur is de SNR waarden geplot voor verschillende threshold waarden en voor verschillende threshold functies. Uit deze afbeelding is af te leiden dat gladde threshold functie het beste resultaat oplevert voor de ruisreductie. In Figuur 6 werd gebruik gemaakt van de biorthogonale wavylet van orde 6,8. Voor de meeste andere transformaties werden gelijkaardige resultaten bekomen. Uit alle tests besluiten we dat de gladde threshold functie de beste ruisreductie oplevert. Het gladde karakter van de threshold functie heeft ook als gevolg dat de fout op de transformatie een gladde functie van de threshold parameter is. Dit is gewenste eigenschap wanneer men gebruik wil maken van optimalisatie algoritmes die gebruik maken van de afgeleide van de kost functie.

1.4.3 Optimale threshold bepalen(met vals spelen)

Uit het vorige experiment hebben we kunnen besluiten dat in alle gevallen de gladde threshold functie de beste ruis reductie oplevert. Een tweede resultaat dat opviel was dat de SNR curves steeds gladde curves bleken te zijn voor de gladde threshold functie. Door het gladde karakter van deze curve is het gebruik van een optimalisatie routine voor de SNR costfunctie makkelijk te implementeren. De cost functie is als volgt gedefinieerd in matlab.

```
costFun = @(T) -snr_den(An,A,Nb_levels,wname,@(x) SmootThresh(x,T));
```

Dit is een functie in de parameter T , de waarden van de threshold. An is de ruizige afbeelding, A de originele afbeelding, Nb_levels het aantal niveaus van de transformatie, $wname$ de naam van de wavelte transformatie en $@(x) SmootThresh(x,T)$ de threshold functie. Merk op dat voor de berekening van de SNR waarden de originele afbeelding moet



Figuur 6: SNR voor benadering van de afbeelding van lena. Het ruisniveau van de originele afbeelding is weergegeven met de stippellijn. Voor de wavelet transformatie is gebruik gemaakt van de biorthogonale wavelet filter van orde 6 en 8. De filter werd tot niveau 6 berekend. Dit was het beste resultaat dat werd bekomen voor de niet redundante transformatie.

gekend zijn (Vals spelen). Door gebruik te maken van de optimalisatie routine `fmincon` kan de optimale waarden voor de threshold snel worden gevonden.

```
[opt_thres,opt_snr] =fminunc(costFun,0.2);
```

Deze methode convergeerde steeds, soms was de convergentie naar een negatieve waarde van de threshold. Dit is geen probleem aangezien de threshold functie symmetrisch is in de threshold parameter. Een beknopte uitwerking van dit algoritme is gegeven in de appendix.

Een voorbeeld resultaat van de optimale ruis onderdrukking is gegeven in figuur 7. In deze figuur is opnieuw de biorthogonale wavelet van orde 6,8 gebruikt.

1.4.4 Optimale threshold bepalen(zonder vals spelen)

Het kiezen van de threshold parameter door gebruik te maken van de originele afbeelding is niet realistisch. Om dit op te lossen zijn er een aantal criteria die de kwaliteit van de benadering kunnen schatten zonder gebruik te maken van de originele afbeelding. Een van deze criteria is het SURE criteria, dat beschreven staat in de opgaven. In figuur 8 worden de twee kost criteria met elkaar vergeleken. Het is duidelijk dat SURE kost functie steeds lagere waarden voor de optimale threshold geeft. Een tweede eigenschap die opvalt is dat het minima van de SURE kostfunctie vlakker is dan die van de SNR kostfunctie. Dit wijst er op dat de SURE schatter minder gevoel is dan de SNR schatter voor de optimale waarde. De getallen op de y -as hebben kunnen niet onderling worden vergeleken met elkaar omdat er telkens een andere schaling wordt gebruikt.

Het is wel mogelijk om voor beide optima de SNR te bepalen. In ons voorbeeld hebben we gewerkt met de daubechie 6 wavelet tot niveau 5 met de gladde threshold functie. De



(a) Ruizige afbeelding van Lena, deze afbeelding heeft een SNR van 10.45 dB



(b) Gefilterde afbeelding van Lena, deze afbeelding heeft een SNR van 16.49 dB

Figuur 7: Vergelijking van de ruisige afbeelding met het optimaal gefilterd resultaat. De filtering werd bekomen door gebruik te maken van de gladde threshold functie waarbij de parameter optimaal werd gekozen. De gebruikt wavelet was de biorthogonale wavelet van orde 6 en 8, deze werd gebruikt tot niveau 10.

optimale SNR die werd bekomen voor het SURE criteria was 22.51 dB. Wanneer de SNR zelf werd geoptimaliseerd werd er een SNR van 25.02 dB gevonden. Beide gefilterde afbeeldingen zijn weergegeven in Figuur 9.

1.4.5 Beste strategie

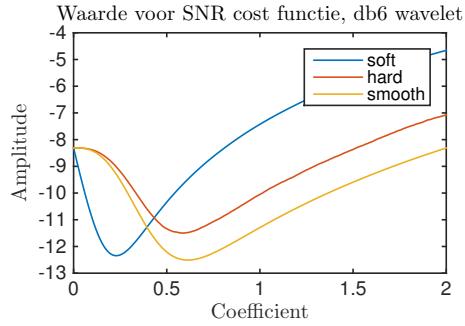
Wanneer de keuze voorhanden is is het aangeraden om steeds de SNR kostfunctie te gebruiken voor een optimale threshold waarden te bepalen. Dit is echter niet altijd mogelijk. In dit geval is het SURE criteria een goed alternatief.

1.5 Redundante wavelet transformatie

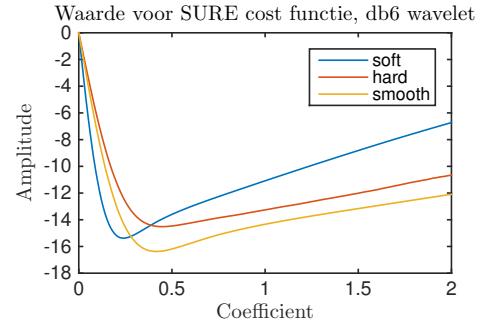
De redundante wavelet transformatie voor twee dimensionale foto's kan gebruikt worden via het commando `swt2`. Dit geeft als resultaat een drie dimensionale array terug met dimensies $H \times L \times N$, waarbij H, L respectievelijk de hoogte in pixels en de breedte in pixels van de foto is. De reconstructie van een ruizige foto kan met behulp van volgende twee lijnen code.

```
swc =.swt2(X,N,wname);  
Y = iswt2(thresHold(swc),wname);
```

Nadien kan de fout tussen de originele afbeelding X en de gereconstrueerde foto Y worden bepaald zoals voorheen. Dit kan nadien opnieuw worden gebruikt in een optimalisatie routine. Een meer gedetailleerde uitwerking van het algoritme is terug te vinden in de appendix.



(a) SNR kostfunctie



(b) SURE kost functie

Figuur 8: Vergelijking van twee kost criteria. In de linker figuur werd de benaderingsfout bepaald als de SNR ten opzichte van de originele afbeelding. In de rechter plot werd de kwaliteit van de benadering bepaald aan de hand van het SURE criteria. De SURE benadering geeft telkens kleiner waarden voor de optimale threshold parameter, bovendien is het minima veel vlakker. Dit wijst op de lage gevoeligheid voor het bepalen van de optimale parameter.



(a) Optimale ruisreductie men SURE criteria, SNR=22.51 dB



(b) Optimale ruisreductie men SNR criteria, SNR=25.02 dB

Figuur 9: Vergelijking van ruis reductie op basis van de twee verschillende kost criteria. Voor de wavelet transformatie is er gebruik gemaakt van de daubechie wavelet van orde 6 tot niveau 5. Het optimum is bepaald door middel van de optimalisatie routine fminunc.



(a) Beschadigde foto met SNR van 16.60 dB (b) Reconstructie met SNR van 27.73 dB

Figuur 10: Resultaat voor reconstructie van ruizige foto aan de hand van $db6$ wavelet met orde 5. Bij de reconstructie werd er gebruik gemaakt van de redundante wavelet transformatie. Als threshold functie werd de gladde functie gebruikt. De threshold parameter werd optimaal gekozen. Dit is het beste resultaat dat bekomen werd.

1.5.1 Resultaten

Het beste resultaat voor de denoising van afbeeldingen werd bekomen met de redundante wavelet transformatie. Als kost functie werd de afstand tot de originele afbeelding genomen. Als threshold functie werd er gekozen voor de gladde threshold. Hierbij werd de threshold waarden bepaald met behulp van `fminunc`. Op deze manier werd er een SNR van maximaal 27.73 bereikt waarbij de origine beschadigde foto een SNR van 16.60 dB had. Dit resultaat is weergegeven in Figuur 10

1.5.2 Rooster ruis

Als afsluiter van het deel over ruisreductie stappen we af van de gaussische ruis en voegen we een gestructureerde verstoring (ruis) toe aan de afbeelding. Deze gestructureerde ruis is in de vorm van een rooster dat over de afbeelding wordt geplaatst en bevat dus veel structuur. Voor de eenvoudige implementatie van het ruisreductie algoritme is het herkennen van deze rooster structuur als overbodig niet mogelijk. De gebruiker moet meer informatie meegeven over dit soort van ruis. Het resultaat van het ruisreductie algoritme voor dit probleem is gegeven in Figuur 11. In de linker figuur is het klassieke ruisreductie algoritme gebruikt, in de rechter figuur is een inpainting gebruikt (volgend hoofdstuk). Het verschil tussen beiden resultaten is groot. Dit is hoofdzakelijk het gevolg van de beschikbare informatie. Voor de ruisreductie werd er geen extra informatie over de beschadiging gebruikt. Bij het inpainting algoritme werd er specifiek meegegeven welke pixels beschadigd zijn.



(a) Verwijderen van rooster met ruisreductie, SNR=-4.12 dB

(b) Verwijderen van rooster met ruisreductie, SNR=26.60 dB

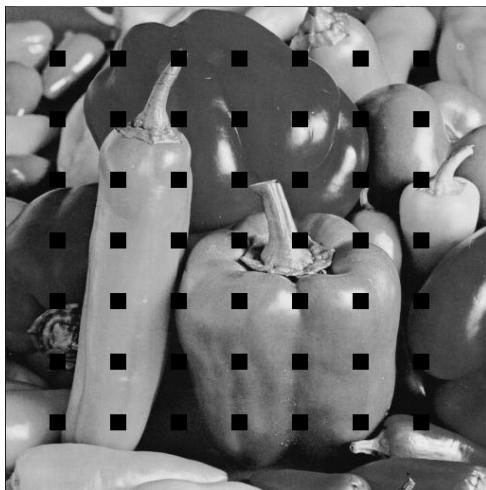
Figuur 11: Resultaat voor reconstructie van foto die beschadigd werd met een rooster. In de linker afbeelding werd er gebruik gemaakt van het klassieke ruisreductie algoritme. In de rechter afbeelding werd gebruik gemaakt van het inpainting algoritme. Voor beide resultaten werd de *bd6* wavelet gebruikt en de zachte threshold functie. Voor de linker afbeelding werd de threshold waarden optimaal gekozen. In de rechter afbeelding was deze 10.

2 Inpainting

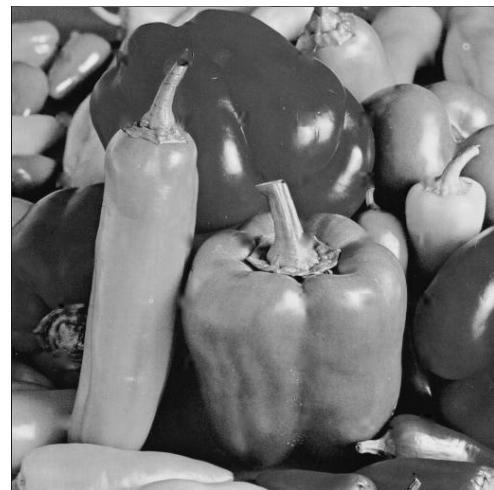
In dit hoofdstuk zullen we wavelets gebruiken om ontbrekende regio's in foto's in te kleuren. Deze methode wordt gebruikt om beschadigde foto's te reconstrueren. De schade op de foto wordt gemodelleerd met pixel waarden in de foto die verdwenen zijn. Om de verdwenen regio's in te kleuren wordt de iteratieve methode gebruikt die beschreven staat in de opgave. We hebben dit algoritme geïmplementeerd in Matlab met behulp van de Wavelet toolbox. Hierbij nemen we een bepaalde foto en verwijderen we de pixels in bepaalde regio's. Het algoritme probeert dan de verdwenen regio's in te kleuren.

In figuur 12 wordt het algoritme geïllustreerd met verschillende soorten regio's van pixels die verwijderd zijn. In figuur 12a zijn er blokken pixels verwijderd, in figuur 12c zijn er random pixels verwijderd en in figuur 12e is de figuur overschreven met tekst. De resultaten van het 'inpainting' algoritme staan er steeds naast. Het algoritme geeft op het eerste zicht heel mooie resultaten. Alleen als we de ingekleurde resultaten in detail gaan bekijken merken we dat het niet de originele figuren zijn. Dit is het meest duidelijk bij de figuur die bewerkt is met blokken.

Het 'inpainting' algoritme kan gebruikt worden op verschillende manieren. Zo zijn er verschillende soorten wavelets die gebruikt kunnen worden. De thresholding kan op verschillende manieren gebeuren en de threshold parameter δ moet gekozen worden. De effecten op het resultaat van al deze verschillende soorten instellingen zullen besproken worden in de volgende hoofdstukken.



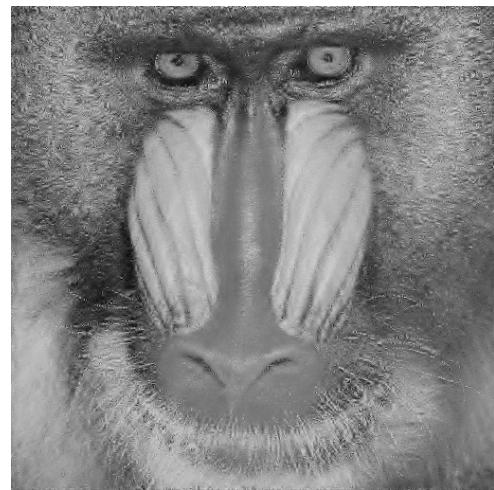
(a) Foto van pepers met vierkante blokjes pixels verwijderd (zwart gemaakt).



(b) Reconstructie van pepers foto aan de hand van inpainting.



(c) Foto van een baviaan met ongeveer 70 procent van de pixels verwijderd (zwart gemaakt).



(d) Reconstructie van de baviaan aan de hand van inpainting.

Inurn ioirba nvdu fooitpsihrn owrhe el
m neews hsnuo enuii taohantie. Aehs
a riuinai nnhoce oawre daxeieaua; liae
iti trlu hutuu. \n\nOtya eeponno auo
h nfhd a vtmntno eeahg anoaeh. Ais
e ibnao conoytolgw wets etefatia oigr
f oljenmsn. Utce elawt lcloro smww
aeiia lsneo hue sudisrt! V\ngGr ealig
wg ahewouel ordy eddsatp tbnrke c
oul stcanif. Epr aorrrhhdu nwhtegsel
cgc stile nulchent aassp tv. Euaidtb
ymic ewee ienocenaca swssrtw fow
n erneutei axsygneni rhccossio irya r
tau teetaft haeeeuiio. Orieod irznelu
phep larn taein ndatikew avese ikrvue
urinit augackpul. loalurig yeedpsn uu
alech pntor euc uonee eexnaheo um

(e) Foto van lena overschreven met tekst. (zwart gemaakt)



(f) Reconstructie van lena aan de hand van inpainting.

Figuur 12: Resultaten van het 'Inpainting' algoritme. Instellingen algoritme: wavelet: 'db5', level $N = 10$, soft thresholding, threshold parameter $\delta = 10$, 200 iteraties.



(a) **Soft thresholding ($\delta = 10$)**. (b) **Hard thresholding ($\delta = 100$)**.

Figuur 13: Met tekst overschreven figuur 12e ingekleurd met 'inpainting' algoritme. Instellingen algoritme: wavelet: 'db5', level $N = 10$, 200 iteraties.

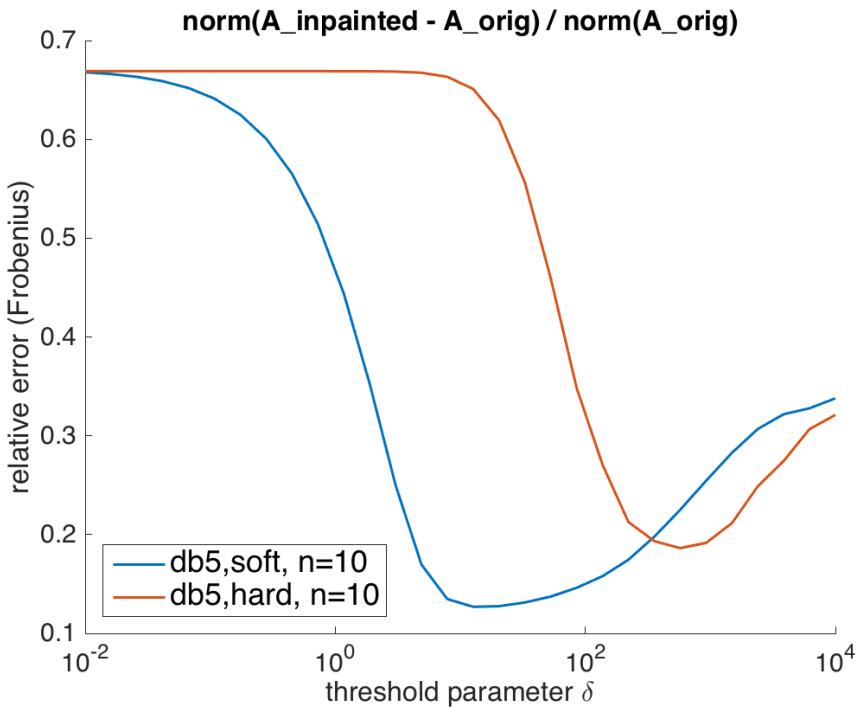
2.1 Threshold technieken

Er zijn 2 soorten technieken voor thresholding, namelijk soft thresholding en hard thresholding. Voor beide technieken moet ook een threshold parameter $\delta > 0$ gekozen worden. In figuur 13 is een figuur ingekleurd op twee manieren. Eén keer met soft thresholding en threshold parameter $\delta = 10$ en de andere keer met hard thresholding en threshold parameter $\delta = 100$. In het geval van soft thresholding was het gemakkelijk om een threshold parameter te vinden die redelijk goede resultaten geeft. Voor hard thresholding was het langer zoeken achter een geschikte threshold parameter $\delta = 100$. Dit was de meest optimale waarde van δ die ik op het eerste zicht kon vinden. Het is duidelijk dat voor soft thresholding betere resultaten kunnen bekomen worden als voor hard thresholding. Bij hard thresholding zijn er nog veel randen van letters zichtbaar. Bij soft thresholding zijn de resultaten veel beter.

Het is duidelijk dat het vinden van de optimale threshold parameter δ niet zo eenvoudig is. Daarom zullen we het volgende experiment uitvoeren. We voeren het 'inpainting' algoritme uit voor een reeks van verschillende waarden van δ . Voor elke waarde van δ zal het relatieve verschil tussen de ingekleurde foto en de originele onbeschadigde foto berekent worden. Dit verschil wordt voorgesteld op de volgende manier:

$$\frac{\|A_{\text{ingekleurd}} - A_{\text{onbeschadigd}}\|_F}{\|A_{\text{onbeschadigd}}\|_F} \quad (1)$$

met A de matrix met pixel waarden corresponderend met de foto. In figuur 14 kan men duidelijk het verschil zien tussen soft en hard thresholding. Soft thresholding bereikt een lager minimum rond de optimale waarde $\delta = 10$. Het minimum bij hard thresholding ligt iets hoger en wordt bereikt voor veel grotere waarden van δ . Opmerkelijk is dat hard thresholding het heel slecht doet indien de parameter δ relatief klein in tegenstelling tot soft thresholding. In figuur 15 is hard thresholding gebruikt met een te lage waarde van δ en de optimale waarde van δ (minimum rode curve figuur 14). Met $\delta = 10$ mislukt het



Figuur 14: Het relatieve verschil beschreven (1) in functie van de threshold parameter δ voor zowel soft als hard thresholding. Instellingen algoritme: wavelet: 'db5', level $N = 10$, 50 iteraties voor elke waarde van δ . Het relatieve verschil tussen de onbeschadigde en de beschadige foto is ongeveer 0.67. De foto van lena overschreven met tekst van in figuur 12e is gebruikt.



(a) **Hard thresholding** met te lage waarde van $\delta = 10$. (volgens figuur 14)

(b) **Hard thresholding** met de optimale waarde van $\delta = 1000$. (volgens figuur 14)

Figuur 15: Met tekst overschreven figuur 12e ingekleurd met 'inpainting' algoritme. Instellingen algoritme: wavelet: 'db5', level $N = 10$, hard thresholding, 200 iteraties. De inkleuring in de linkse figuur is volledig mislukt vanwege een blijkbaar te lage waarde van δ .

inkleuren volledig. Dit was te verwachten als we kijken naar figuur 14. Volgens figuur 14 zou de optimale waarde van δ bij hard thresholding rond 1000 moeten liggen. Hoewel figuur 15b ($\delta = 1000$) niet heel slecht is, verkiest ik persoonlijk toch figuur 13b ($\delta = 100$) boven figuur 15b alhoewel het relatieve verschil voor resultaat 15b minder is. In figuur 15b zijn nog heel duidelijk de letters te zien. De conclusie is dus dat men niet louter naar het minimale verschil in norm moet kijken maar ook naar het bekomen resultaat. Een tweede conclusie is dat soft thresholding het veel beter doet als hard thresholding voor meer verschillende ordes van δ . Soft thresholding heeft minder last met de randen van de letters in tegenstelling tot hard thresholding.

2.2 Redundant vs non-redundant wavelet transformaties

In dit hoofdstuk zullen we kort het verschil bestuderen tussen redundant en non-redundant wavelet transformaties. In figuur 16 zijn 6 ingekleurde figuren getoond. Hierbij zijn 3 soorten wavelets gebruikt en voor elke soort wavelet is de non-redundant en redundant wavelet transformatie uitgetest. Bij elke figuur is de corresponderende SNR waarde getoond. Het is duidelijk dat de bekomen resultaten veel beter zijn bij de redundant wavelet transformaties. Dit is zowel te zien in de figuren als in de SNR waardes. De 'overbodige' informatie dat de redundant wavelet transformatie gebruikt is blijkbaar toch niet zo overbodig in het geval van 'inpainting' toepassingen. Opmerkelijk is dat voor de biorthogonale spline wavelet gebruikt in figuur 16 het resultaat volledig mislukt is voor de non-redundant wavelet transformatie en voor de redundant wavelet transformatie het resultaat juist heel goed is. Dit is te zien in figuur 16e en figuur 16f.

Het enigste nadeel bij redundant wavelet transformaties is dat de rekentijd bij wavelet transformaties een stuk hoger ligt. Deze rekentijd bij redundant wavelet transformaties stijgt ook veel bij hoger gebruikte levels in de transformatie. In ons geval kon het iteratieproces in het 'inpainting' algoritme gemakkelijk 10 keer zoveel rekentijd vragen voor de redundant wavelet transformaties in vergelijking met de non-redundant wavelet transformaties.

2.3 Wavelet soorten

In dit hoofdstuk zal kort het verschil in de resultaten tussen verschillende type wavelets besproken worden. In figuur 17 zijn de 'inpainting' resultaten getoond voor 6 verschillende soorten wavelets. De SNR waarden zijn er steeds bij vermeld. In figuur 17a is de Haar wavelet gebruikt. In deze figuur zijn kleine blokjes te zien, Dit is omdat de Haar wavelet niet geschikt is voor continue overgangen. De 2 Daubechies wavelets in figuur 17b en 17c doen het beter als de haar wavelet. Dit is zowel te zien in de figuur als de SNR waarde. De reden hiervoor is dat Daubechies wavelets beter geschikt zijn voor continue overgangen. In figuur 17d is een biorthogonale spline wavelet gebruikt. In deze figuur is een heel slecht resultaat bekomen. De beste resultaten zijn bij de Coiflet en Symlet wavelet in figuur 17e en figuur 17f.



(a) Non-redundant Haar wavelet,
SNR = 20.96



(b) Redundant Haar wavelet,
SNR = 23.15



(c) Non-redundant Daubechies
(db5) wavelet, SNR = 22.97



(d) Redundant Daubechies (db5)
wavelet, SNR = 24.61



(e) Non-redundant CDF
(bior3.3) wavelet, SNR = 15.93



(f) Redundant CDF (bior3.3) wa-
velet, SNR = 25.50

Figuur 16: Resultaten van het 'Inpainting' algoritme voor verschillende soorten wavelets. Instellingen algoritme: level $N = 6$, soft thresholding, threshold parameter $\delta = 10$, 200 iteraties.



(a) Haar wavelet,
SNR = 21,37



(b) Daubechies 'db2' wavelet,
SNR = 23.69



(c) Daubechies 'db6' wavelet,
SNR = 24.30



(d) CDF wavelet 'bior3.3',
SNR = 19.37



(e) Coiflet wavelet 'coif4',
SNR = 24.55



(f) Symlet wavelet 'sym5',
SNR = 24.37

Figuur 17: Resultaten van het 'Inpainting' algoritme voor verschillende soorten wavelets. Instellingen algoritme: level $N = 10$, soft thresholding, threshold parameter $\delta = 10$, 200 iteraties.

Appendices

In de appendix bevind zich een deel van de code die gebruikt werd voor het bekomen van bovenstaande resultaten. De bijgevoegde code is tot een minimum gehouden, enkel de gebruikte algoritme werden toegevoegd en een script om telkens hun gebruik te illustreren. Het volledige project, inclusief de matlab bestanden om elk van bovenstaande resultaten te genereren kan gevonden worden op <https://github.com/double2double/wavelets>

Matlab Code

Code voor ruisreductie

```
1 function y = den_image(A_n, Nb_levels, wname, thresHold, w_mode,
2 % Function for denoising of images using a wavelet
3 % transformation.
4 % INPUTS:
5 % - A_n: Noisy image
6 % - Nb_levels: (integer) the number of levels in wavelet
7 % transform
8 % - wname: a cell array with different kinds of wavelets
9 % e.g. {'haar', 'db4', 'db6', 'db10'}
10 % - threshold: A function of one that can take a matrix and
11 % return the
12 % threshold values of the matrix. eg @(X) softThreshold(X,1);
13 % - w_mode: 'per' or 'sym',
14 % - redundant: (optional) If set to one the redundant
15 % transformation will
16 % be used.
17 %
18 % OUTPUTS:
19 % - y: The denoised image.
20
21 dwtmode(w_mode, 'nodisp');
22 if (~exist('redundant', 'var'))
23     redundant=0;
24 end
25 if (~redundant)
26     [a, b]=wavedec2(A_n, Nb_levels, wname);
27     a_T = thresHold(a);
28     y = waverec2(a_T, b, wname);
29 else
30     swc = swt2(A_n, Nb_levels, wname);
31     y = iswt2(thresHold(swc), wname);
32 end
33 end
```

Voorbeeld code voor gebruik van denoising.

In het onderstaande script wordt aan de hand van `fminunc` de optimale waarden voor de therhold waarden bepaald. De cost functie die wordt gebruikt is de euclidische afstand tussen de originele en de gefilterde afbeelding.

```
1 % Example script for denoising algorithm. In this script the
2 % optimal
3 % denoising parameter will be found by fmincon .
4 clear
5 close
6 [A_orig ,cmap] = imread( '../.../ matlab/src/lena.gif' );
7 A = double(A_orig );
8 A_mean = mean(A(:));
9 A = A-A_mean;
10 A_var = var(A(:));
11 A = (A./ sqrt(A_var));
12
13 % Settings for the denoising
14 wname = 'db2'; % wavelet name
15 Nb_levels = 6; % Number of levels
16 w_mode = 'per'; % Boundary type
17 maxit= 10; % Maximum itterations
18 redundant= 0; % 1 for redundant transformation
19 thres = 1; % Threshold value
20 sigma=0.2;
21
22 % Generate noisy image
23 A_n= A +randn( size(A) ).*sigma;
24 % Cost function for images .
25 snr_image = @(An) -20*log10( norm(A - An, 'fro') / norm(A) );
26 % Define the threshold function
27
28 SmootThresh = @(x,T) -x.*exp(-(x/T).^4)+x;
29 % Cost function for threshold parameter
30 costFun = @(T) -snr_image(den_image(A_n ,Nb_levels ,wname,@(x)
SmootThresh(x,T),w_mode,0));
31 [a,b] =fminunc(costFun ,0.2);
```

Code voor inpainting

```
1 function [ B_np1 ,snr ] = inpainting_fun( A_dist ,mask ,Nb_levels ,
threshold ,cost ,wname ,w_mode , maxit ,redundant )
2
3 % Implementation of the inpainting algorithme based on a wavelet
4 % transformation .
```

```

5 % INPUTS:
6 % - A_dist: Distorted image.
7 % - mask: binary matrix that represent the missing pixels with
8 % a one.
9 % - Nb_levels: (integer) the number of levels in wavelet
10 % transform
11 % - threshold: A function of one that can take a matrix and
12 % return the
13 % threshold values of the matrix. eg @(X) softThreshold(X,1);
14 % - cost: Costfunction to rate the quality of the image.
15 % - w_mode: 'per' or 'sym'
16 % - wname: a cell array with different kinds of wavelets
17 %           e.g. {'haar','db4','db6','db10'}
18 % - maxit: maximum itterations of the algorithm
19 % - redundant: (optional) If set to one the redundant
20 % transformation will
21 % be used.
22 %
23 % OUTPUTS:
24 % - B_np1: The last matrix of the inpainting itteration
25 % - snr: The value of the cost function for the last
26 % itteration.

27 dwtmode(w_mode, 'nodisp');
28 if (~exist('redundant', 'var'))
29     redundant=0;
30 end
31 % Define the correct transformations.
32 if (~redundant)
33     PsiS = @(f) wavedec2(f, Nb_levels, wname);
34     Psi = @(C,S) waverec2(C,S,wname);
35 else
36     PsiS = @(f) deal(swt2(f, Nb_levels, wname),0);
37     Psi = @(C,S) iswt2(SWC,wname);
38 end
39 B_n=A_dist;
40 B_np1=B_n;
41 snr=0;
42 for n=1:maxit
43     disp(n);
44     [C,S] = PsiS(B_n);
45     C = threshold(C);
46     B_np1 = (1-mask).*A_dist+mask.*Psi(C,S);
47     B_n=B_np1;
48     snr = cost(B_n);
49 end
50 end

```

Voorbeeld code voor gebruik van inpainting.

In onderstaand script word een afbeelding ingelezen en beschadigd met witte ruis. Nadien word het inpainting algoritme gebruikt om de ontbrekende pixelwaarden te schatten.

```
1 % Example code for the usage of the inpainting script.
2 clear
3 close
4 [ A_orig ,cmap] = imread( '../src/lena.gif' );
5 A = double(A_orig);
6 % Settings for the denoising
7 wname =           'db2'; % wavelet name
8 Nb_levels =       6;    % Number of levels
9 w_mode =          'per'; % Boundary type
10 maxit=           10;   % Maximum itterations
11 redundant=       0;    % 1 for redundant transformation
12 thres =          10;   % Threshold value
13 % create mask
14 mask = rand( size(A));
15 mask(mask<0.7)=1;
16 mask(mask~=1)=0;
17 % Setting up distorted A
18 A_dist = A.* (1-mask);
19 % Create cost function and threshold function
20 cost = @(A_n) 10*log10( norm(A, 'fro')^2 / norm(A - A_n, 'fro')^2
   );
21 threshold = @(x) x.*max( 0, 1-thres./max( abs(x),1e-10) );
22 % running the algorithm
23 [ result ,snr_result ] =inpainting_fun( A_dist ,mask ,Nb_levels ,
   threshold ,cost ,wname , 'per' , maxit ,redundant );
24 % Plotting the result
25 colormap(cmap);
26 imwrite(uint8(A_dist),cmap, 'plot/lena_broke.png')
27 imwrite(uint8(result),cmap, 'plot/lena_fixed.png')
```