Windowing

Die meisten Informationen über Windows sind in Englisch erhältlich. Die Grundsätze von Windowing sind schnell und einfach verständlich. Tieferes Verständnis ohne gute FFT-Kenntnisse ist aber schwierig.
Gute Quellen:

http://www.reed-electronics.com/tmworld/article/CA187572

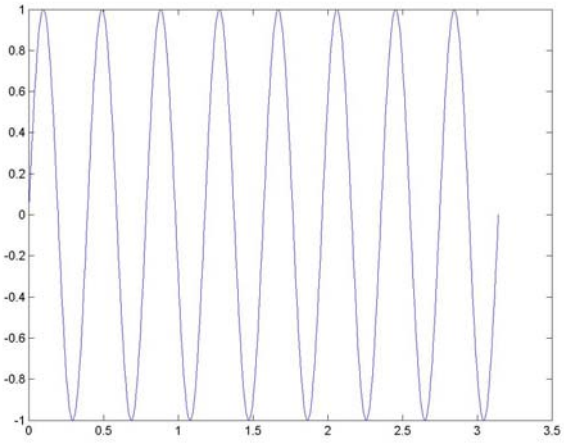
http://www.lds-group.com/docs/site_documents/AN014%20Understanding%20FFT%20Windows.pdf

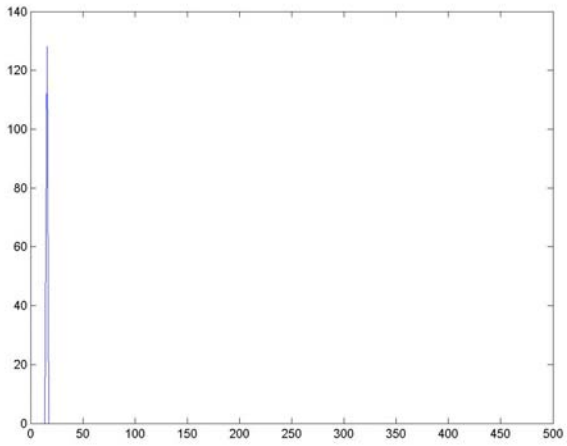
http://www.dsp-guide.com/

Wieso brauchts Windowing

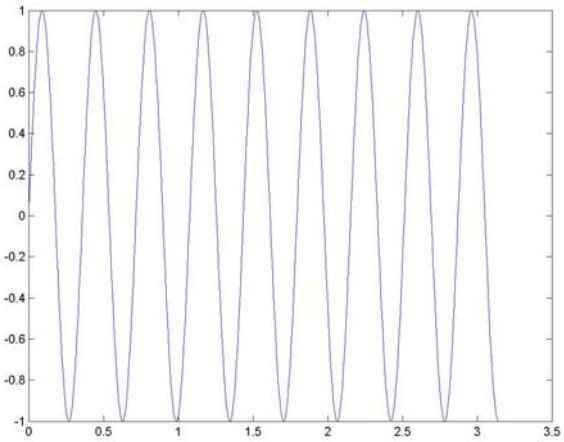
Da ein Computer mit diskreten Signalen arbeitet, bekommen wir stets nur Samples mit. Die FFT arbeitet mit einem Buffer (typischerweise 1024, 2048,4096, 8192,...).
Da die FFT auf repetiven Signalen aufbaut entstehen oft sogenannte Leakage. Diese haben ihren Ursprung, wenn die Grundfrequenz im Buffer nicht ganzzahlig oft vorkommt und dadurch am Ende Sprungstellen entstehen:

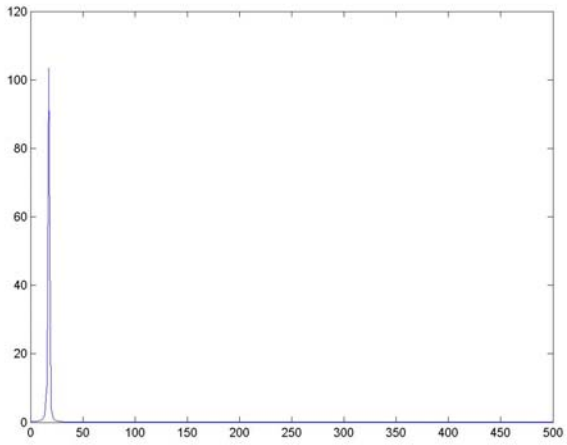
Wiederholendes Signal-Stück:





„Springendes“ Signal-Stück:

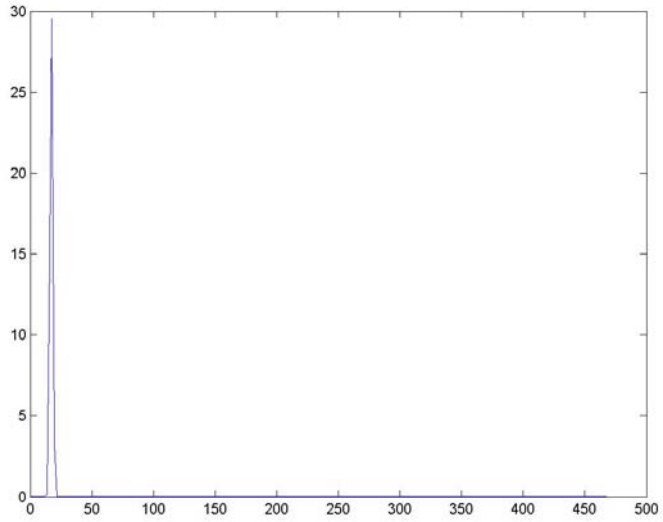




Signaldauer: 0-pi
Samples: 512
1. Signal: sin(16*x)
2. Signal : sin(17*x)

Es ist im Frequenzbereich schön ersichtlich wie beim 1. Signal nur ein scharfer Piek vorhanden ist. Beim 2. Signal findet jedoch ein sogenanntes Leakage statt. Anteile der Frequenz werden auf die „Nachbar-Frequenzen“ verteilt.

Wendet man jedoch ein Windowing an, wird das Leakage vermindert:
FFT-Ergebnis von sin(17.5*x) mit Windowing:

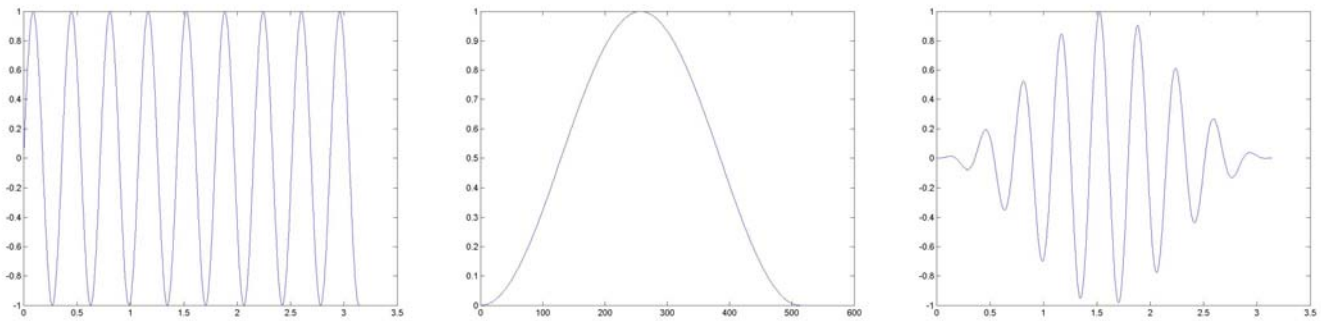


Der Effekt kann jedoch nicht völlig verhindert werden.

Was macht Windowing

Windowing ist eigentlich eine ganz einfache Sache. Auf die gesampleten Daten wird ein Window gelegt. Dieses hat die selbe Grösse wie der Buffer. Das Bekannteste Window ist das Hanning-Window. Ziel der Windows ist es die Sample-Stücke so umzugestalten, dass sie wiederholt werden können ohne Sprungstellen.

Dies heisst, die Ränder werden gegen Null gebracht:

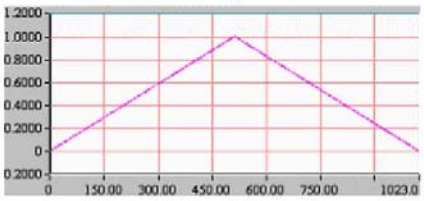


Hanning: w(n)=0.5-0.5*cos(2*pi*n/N)

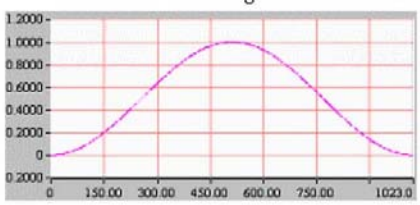
Verschiedenste Windows und ihr Zweck

Window	Best for these Signal Types	Frequency Resolution	Spectral Leakage	Amplitude Accuracy
Barlett	Random	Good	Fair	Fair
Blackman	Random or mixed	Poor	Best	Good
Flat top	Sinusoids	Poor	Good	Best
Hanning	Random	Good	Good	Fair
Hamming	Random	Good	Fair	Fair
Kaiser-Bessel	Random	Fair	Good	Good
None (boxcar)	Transient & Synchronous Sampling	Best	Poor	Poor
Tukey	Random	Good	Poor	Poor
Welch	Random	Good	Good	Fair

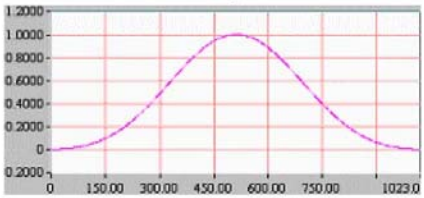
Bartlett Highest Side lobe: -26dB



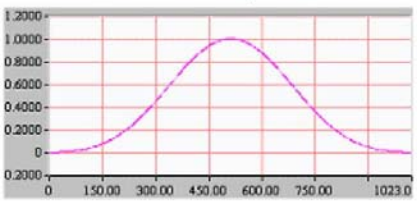
Hanning Highest side lobe: -32 dB



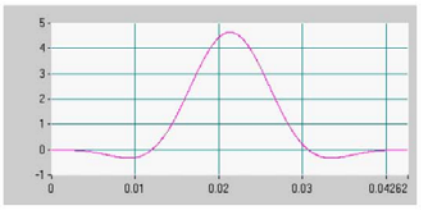
Blackman Highest side lobe: -74 dB



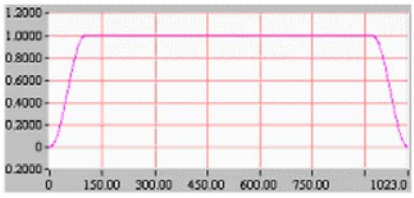
Kaiser-Bessel Highest side lobe:-70 dB



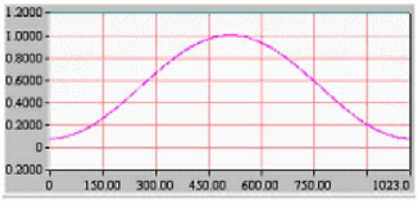
Flat Top Highest side lobe: -93dB



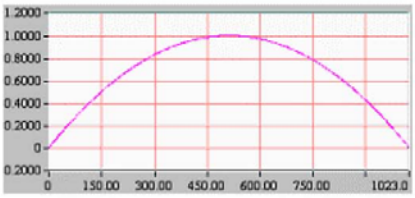
Tukey Highest side lobe: -13 dB



Hamming Highest side lobe-42dB



Welch-Highest side lobe: -21dB



Quelle: http://www.lds-group.com/docs/site_documents/AN014%20Understanding%20FFT%20Windows.pdf

Windowing-Class Version 1

Header-File:

```
#include <vector>

#ifndef windowing_H
#define windowing_H

const double PI = 3.141592653;

class windowing{
public:
    windowing();
    windowing(int percent, int buffersize);
    ~windowing(){}

    void setBuffersize(int buffersize);
    void setPercent(int percent);
    int getBuffersize();
    int getPercent();
    void applyWindow(float * samples);

    void calcHanning();
    void calcHamming();
    void calcBartlett();
    void calcFlatTop();
    void calcBlackman();
    void calcTukey(float r);


private:
    int _percent;
    int _buffersize;
    std::vector<float> _window;
};

#endif
```

Verwendete Hanning und Hamming Filter-Funktionen:
i : Sampleposition (0..._buffersize)
Hanning: `0.5-0.5*cos(2*PI*i/(_buffersize-1))`

Hamming: `0.54-0.46*cos(2*PI*i/_buffersize)`

Weitere im SourceCode ersichtlich.

Bisherige Erfahrungen bezüglich Anwendung

Bei den ersten Versuchen ging es gründlich schief, da anstatt auf den Rohdaten das Windowing versehentlich erst nach der FFT durchgeführt wurde. Theoretisch soll dies aber sogar möglich sein, jedoch muss dafür das Window in den Frequenzbereich gebracht werden.

Weitere Versuche gelangen dann erfolgreicher. Es ist aber sehr schwierig überhaupt einen Unterschied zwischen mit und ohne Window festzustellen.

Dies liegt wohl daran, dass Windowing eher gedacht ist, um Störsignale rauszubekommen. Musik besteht aber aus so vielen Signalen, dass man da nur schwer zwischen Signal und Störsignal unterscheiden kann.

Deshalb ist eines der nächsten Ziele abzuklären ob der Einsatz von Windowing bei Audiodaten überhaupt sinnvoll ist.

Ausführliche Testerfahrungen mit Windowing

Wie bereits angenommen ist der Einsatz von Windowing bei Audiodaten fraglich. Grösstenteils werden verschiedenste „Klopf bzw. Rauschsignale“ hinzugefügt und der Klang verglichen mit den nicht gewindowdten Daten deutlich schlechter.

DSP-Guide

Das Buch DSP-Guide (www.dspguide.com) bringt viele nützliche Informationen bezüglich Digitaler Signal Verarbeitung (es kann wohl als Standardwerk bezeichnet werden). Eine Lektüre der wichtigsten Kapitel kann wohl jedem empfohlen werden, der mit Signalen „rumspielen“ will.
Leider fand ich dieses frei verfügbare Buch erst relativ spät und viele der zu Beginn getätigten Erfahrungen hätten erspart werden können.

Empfehlenswert für Audio sind sicher die Kapitel:
6 – Convolution
12 – The Fast Fourier Transformation

18 – FFT Convolution
weitere interessante Kapitel zur Grundlagenerarbeitung nach Bedarf (Kapitel 6-13)

Faltung vs. Multiplikation

Meistens muss in der Zeit-Domain eine Faltung vorgenommen werden. Dies ist jedoch etwas aufwendig. Diese Faltung entspricht in der Frequenz-Domain einer Multiplikation. Durch das die FFT sehr schnell ist, lohnt es sich die vorhandenen Daten via FFT in die Freuquenz-Domain zu bringen, dort mit dem Filter zu multiplizieren und dann mit IFFT wieder in die Zeit-Domain zu bringen. Dadurch wird die Anwendung von Filter deutlich vereinfacht.
Die erste Implementation der Windowing-Class enthält dadurch noch Fehler.
Eine Behebung dieses Fehler bei **Hanning** brachte aber, wie verschiedensten Ortes erwähnt, keine Verbesserung, sondern fügte dem originalen Audiosignal ein Klopfen hinzu.

Overlapping

Beim Overlapping gibt es grundsätzlich zwei Arten:
Overlap-Add
Overlap-Save

Zuerst einmal nur das Vorgehen für Overlap. Später was den Unterschied Add und Save ausmacht.
Wie so oft bringen neue Kenntnisse Verbesserungen in den alten Anwendungen, so auch hier. Für den Uebergang einer Faltung im Zeitbereich zu einer Multiplikation im Frequenzbereich ist die Anwendung von Overlap-Add nötig.

Die Faltung zweier Vektoren u und v der Länge m=length(u) und n=length(v) ergibt den neuen Vektor w der Länge k=m+n-1:

$$w(k) = \sum_j u(j)v(k+1-j)$$

wenn m=n ergibt sich somit folgendes:

w(1) = u(1)*v(1)
w(2) = u(1)*v(2)+u(2)*v(1)
w(3) = u(1)*v(3)+u(2)*v(2)+u(3)*v(1)
...
w(n) = u(1)*v(n)+u(2)*v(n-1)+ ... +u(n)*v(1)
...
w(2*n-1) = u(n)*v(n)

Mit Overlap-Add kann man nun anstatt dieser Faltung im Frequenzbereich eine Multiplikation vornehmen. Die gesampleten Daten müssen mit Nullen aufgefüllt

werden. Danach kann die FFT stattfinden und im Frequenzbereich kann dann eine normale Multiplikation getätigt werden:
Sei

X = fft([x zeros(1,length(y)-1)])
Y = fft([y zeros(1,length(x)-1)])

So gilt

conv(x,y) = ifft(X.*Y)
(Formeln aus der Matlab-Hilfe)

Die Grafik aus dem DSP-Guide zeigt das Ganze nocheinmal visuell.

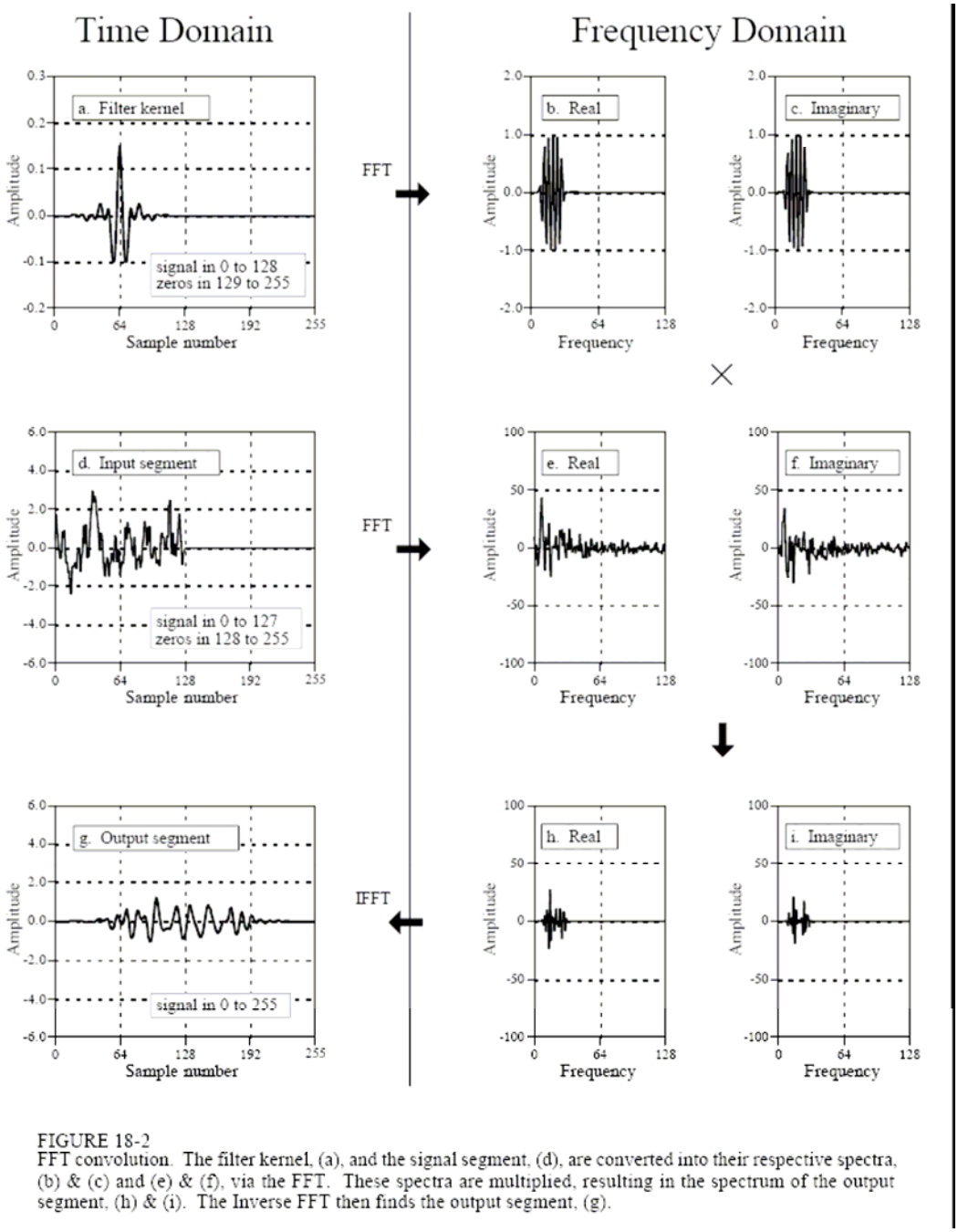
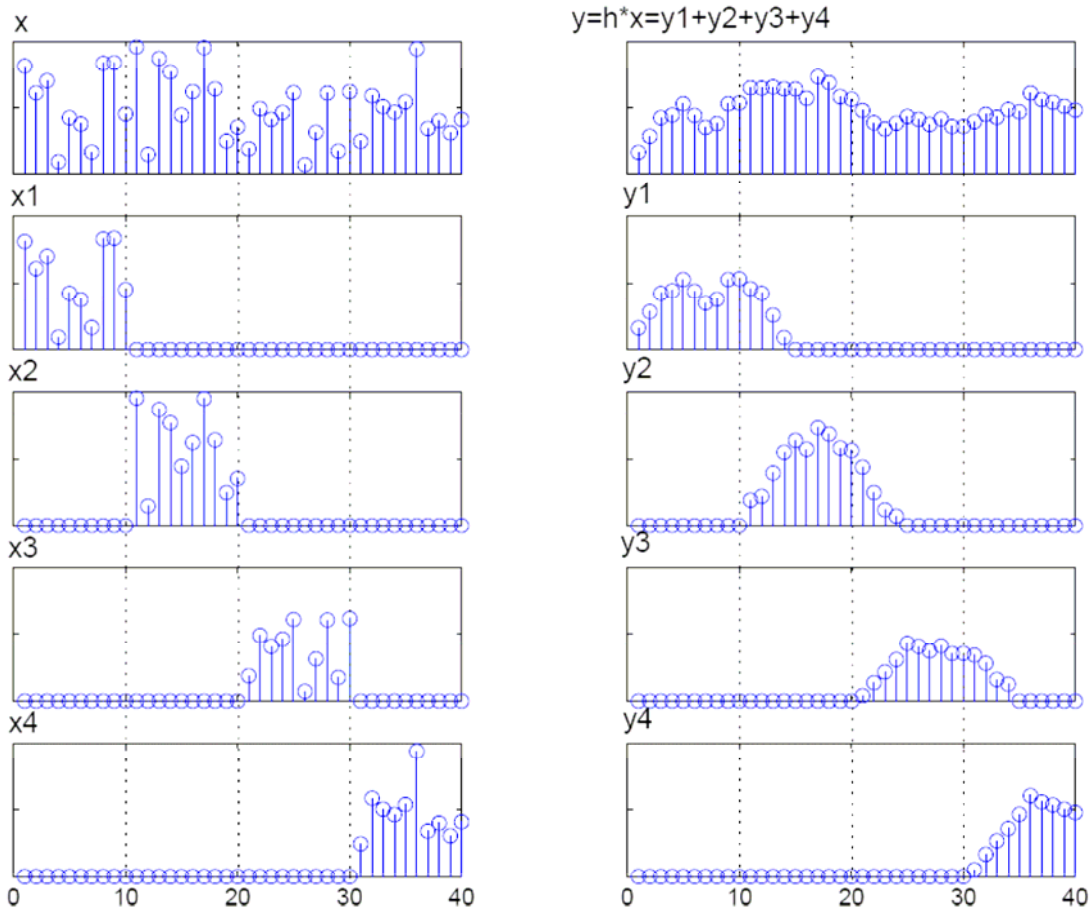


FIGURE 18-2
FFT convolution. The filter kernel, (a), and the signal segment, (d), are converted into their respective spectra, (b) & (c) and (e) & (f), via the FFT. These spectra are multiplied, resulting in the spectrum of the output segment, (h) & (i). The Inverse FFT then finds the output segment, (g).

Quelle: DSPGuide, Seite 315

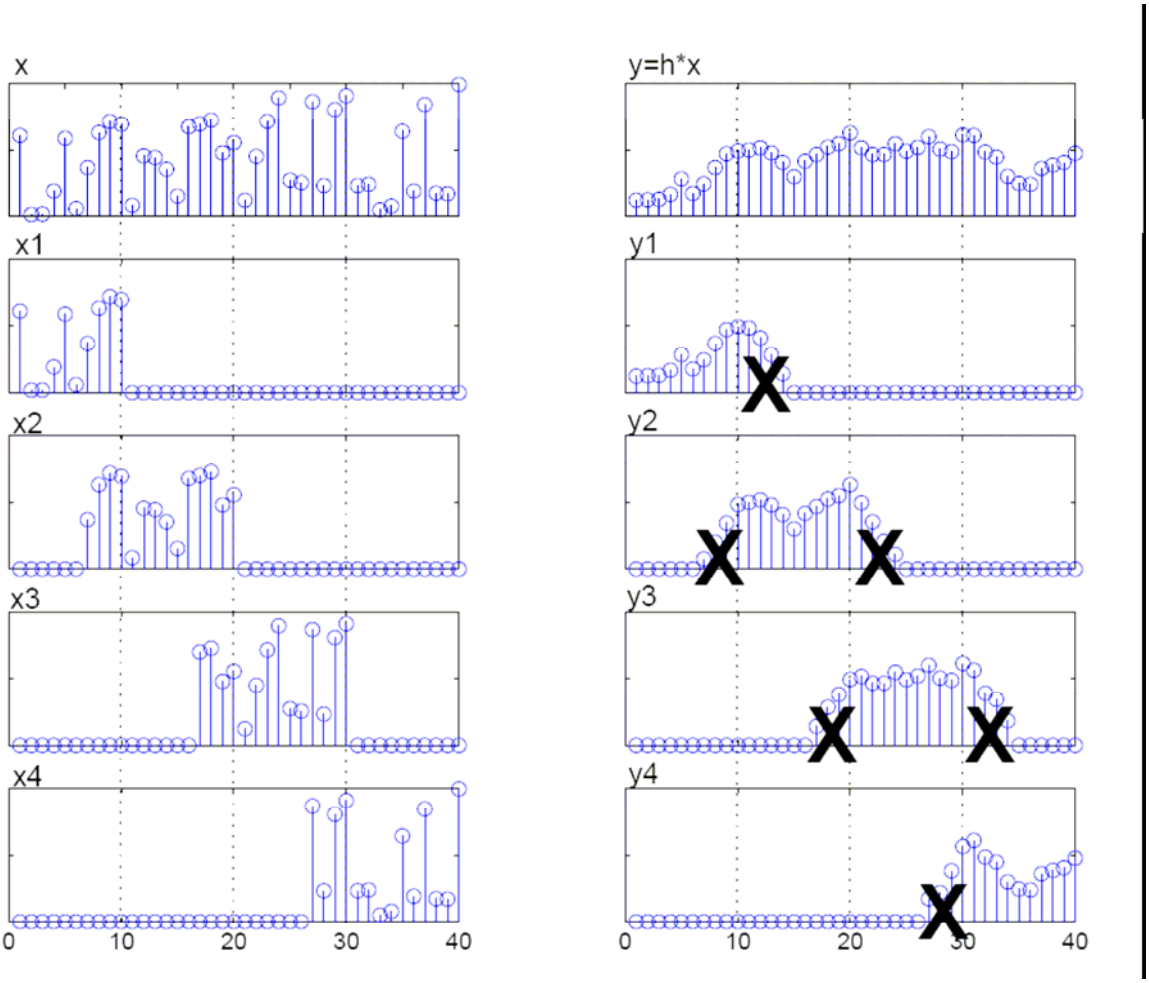
Zusammenfassend kann gesagt werden ohne das Anwenden von Overlap entspricht die Faltung im Zeitraum NICHT der Multiplikation im Frequenzbereich.

Nun zum Unterschied zwischen Overlap-Add und Overlap-Save.
Der Unterschied befindet sich nur darin was vom Outputsignal genommen wird.
Beim Overlap-Add werden die überlappenden Signale einfach addiert:



Overlap-Add (http://www.medialab.ch/archiv/pdf_studien_diplomarbeiten/1sa03/audio_equalizer_mittels_simulink.pdf)

Beim Overlap-Save hingegen werden die Überlappenden bereiche „abgeschnitten“:



Overlap-Save (http://www.medialab.ch/archiv/pdf_studien_diplomarbeiten/1sa03/audio_equalizer_mittels_simulink.pdf)

Nun stellt sich die grosse Frage, welche Methode (Overlap-Add oder Overlap-Save) für die Audioverarbeitung sinnvoller ist.
Weitere Nachforschungen und/oder Test werden darüber genauer Aufschluss geben. Allgemein liest man öfters vom Overlap-Add, jedoch ist dies keine Garantie, dass damit auch wirklich die besseren Ergebnisse erreicht werden.

Die Implementation des ganzen Verfahrens stellte sich als etwas komplizierter dar als angenommen. Mittlerweile gibt es keine Fehler mehr beim Speicherzugriff. Jedoch ist das ganze trotzdem noch nicht testbar, da beim Output einfach nichts mehr zu hören ist, da alle Signale nahe zu 0 sind.