Matlab Projekt SS17

26.08.2017

Marcel Grandinetti

Vorwort

Diese Dokumentation beschreibt die Umsetzung der einzelnen Aufgaben und die Vorgehensweise zu deren Lösung. Die Fragestellung zu den Aufgaben kann der PDF Datei entnommen werden(Projekt Matlab SS 2017.pdf). Das Projekt wurde vollständig in Matlab R2016a umgesetzt.

Inhaltsverzeichnis

Aufgabe 1

In Aufgabe 1 Wird zuerst ein 16-Bit Bild in 2 8-Bit Bilder aufgeteilt. Eines davon muss erst noch bearbeitet werden damit es richtig angezeigt werden kann. Anschließend wird das Waldmeistereis zum Erdbeereis umgewandelt.

Dateien: Aufgabe1.m
Aufgabe1.png
Aufgabe1_MSB.png
Aufgabe1_LSB.png
Aufgabe1_Erdbeereis.png

Teil 1: 16-Bit Bild (Aufgabe1.png) in zwei 8-Bit Bilder aufteilen.

Ziel ist es ein RGB Bild mit 16-Bit pro Pixel in zwei Bilder aufzuteilen.
Der Aufbau der zwei Bilder ist folgendermaßen:

RGB-Ursprungsbild	(1111 1111 0101 0101	= 16 Bit)
MSB-Bild	(1111 1111	= 8 Bit)
LSB-Bild	(0101 0101	= 8 Bit)

(Bitzustände sind nur zum Verdeutlichen)

Aufgabe1.png

Um das 8-Bit MSB-Bild zu erhalten muss jeder Pixelwert 8 Bit nach rechts verschoben werden. (Ziel = 0000 0000 1111 1111)

```
%HSB-Bild exportieren
bildMSB=bitshift(originalbild,-8);
bildMSB=uint8(bildMSB);
```

Mit *bitshift* und dem Parameter *-8* werden die Bits nach rechts verschoben (mit positivem Vorzeichen nach links). Durch *uint8* wird das Format des MSB-Bildes von uint16 in uint8 geändert.

MSB-Bild

Um das 8-Bit LSB-Bild zu erhalten muss jeder Pixelwert erst 8-Bit nach links verschoben werden, damit die MSB-Bits gelöscht werden und anschließend wieder zurück geschoben werden. (Schieben links= 0101 0101 0000 0000, Ziel = 0000 0000 0101 0101)

```
%LSB-Bild exporiteren
bildLSB=bitshift(originalbild,8);
bildLSB=bitshift(bildLSB,-8);
bildLSB=uint8(bildLSB);
```

Wieder wird das Bild im Anschluss an das Verschieben in das Format uint8 gebracht.
Das LSB-Bild ist jedoch noch nicht Richtig jede zweite Zeile ist in der Mitte gedreht.

```
%Jede zweite zeile der Matrix in der mitte Flippen
bildLSB(2:2:size(bildLSB,1),:,:)=flip(bildLSB(2:2:size(bildLSB,1),:,:),2);
```

Die Bildmatrix besteht aus 3 Elementen BildLSB(Zeilen, Spalten, Ebenen). Die Ebenen stehen für die Farbanteile Rot Grün und Blau (1, 2 & 3) und jede zweite Zeile ist in der mitte gedreht. Durch *2:2:size(bildLSB,1)* wird für die gesamte Zeilenanzahl (*size(bildLSB,1)* gibt die Anzahl an Zeilen der Matrix aus) beginnend mit der 2. Zeile jede zweite Zeile ersetzt. Der Befehl *flip(...,2)* dreht die Zeilen in der Mitte und bei der Elementauswahl der Bildmatrix steht der Doppelpunkt (b*ildLSB(...,:,:)* für jedes Element. Das bedeutet, dass jede zweite Zeile aber auch jede alle Spaltenelemente der Zeile auf allen drei Ebenen der Matrix ersetzt werden.
Des Weiteren sind im LSB-Bild zwei Farbebenen vertauscht hierfür wird eine Ebene zwischengespeichert damit sie nicht verloren geht.

```
%Grün  und blau Anteile tauschen
bildLSBgruen=bildLSB(:,:,2);
bildLSB(:,:,2)=bildLSB(:,:,3);
bildLSB(:,:,3)=bildLSBgruen;
clear bildLSBgruen;
```

Der Doppelpunkt wird als Stellvertreter für alle Elemente verwendet und zum Schluss wird mit *clear bildLSBgruen* der Zwischenspeicher für die grünen Werte wieder gelöscht.

Teil 2: Waldmeistereis in Erdbeereis umwandeln.

Ziel ist es die Farbe des Eis zu ändern, dass sie wie Erdbeereis aussieht.
Damit die Farbe der Hand nicht verfälscht wird muss das RGB bild zuerst in den HSV Farbraum transferiert werden das geschieht folgendermaßen:

```
%HSB-Bild in HSV-Bild umwandeln
bildHSV=rgb2hsv(bildMSB);
```

rgb2hsv ist hier für das umwandeln des Formates zuständig. Der HSV Farbraum ist in Matlab folgendermaßen aufgebaut: Bild(Zeilen, Spalten, Ebenen) wobei die Ebenen sich von RGB unterscheiden (1=Farbe, 2=Sättigung, 3=Helligkeit). Im HSV Farbraum wir die Farbe in Grad angegeben (0°-360°) , in Matlab hingegen wir die Farbe von 0-1 angegeben deswegen ist hier eine Umrechnung notwendig.

```
%Grenzwerte zum Ersetzen definieren
multiplikator=1/360;
untereGrenze=30;
untereGrenze=untereGrenze*multiplikator;
obereGrenze=220;
obereGrenze=obereGrenze*multiplikator;
```

Der *multiplikator* dient zur Umrechnung von Grad in den Matlab-Bereich. Die *obereGrenze* definiert die Farbe ab der die Farbe im Bild ausgetauscht wird. Ebenso die *untereGrenze* die Farbe definiert bis zu der die Farbe im Bild ausgetauscht wird. Obere und untere Grenze sind in Grad angegeben und werden deswegen mit dem *multiplikator* verrechnet.

```
%Grünwerte anhand der Grenzen finden und ersetzen
bildHSV(find(bildHSV(:,:,1)>untereGrenze & bildHSV(:,:,1)<obereGrenze))=1;
```

Mit der Funktion *find()* wird jetzt in der Farbebene des Bildes alle werte gesucht die zwischen der unteren und der oberen Grenze liegen und diese Werte werden durch 1 ersetzt. Die 1 entspricht im Matlab Farbraum der 360° im HSV Farbraum und somit der Farbe Rot.
Im Anschluss wird das HSV-Bild zurück in den RGB Farbraum gewandelt. Hierfür dient die Funktion *hsv2rgb*.

```
%HSV-Bild in RGB-Bild zurückwandeln
bildErdbeer=hsv2rgb(bildHSV);
```

Erdbeereis

Zum Schluss werden alle 3 Bilder (MSB, LSB und Erdbeer) mit *imshow(Bild)* angezeigt und mit *imwrite(Bild,'Name')* gespeichert.

```
%HSB-Bild
figure(1);
imshow(bildMSB);
title('MSB-Bild');
imwrite(bildMSB,'Aufgabe1_MSB.png');

%LSB-Bild
figure(2);
imshow(bildLSB);
title('LSB-Bild');
imwrite(bildLSB,'Aufgabe1_LSB.png');

%Erdbeereis
figure(3);
imshow(bildErdbeer);
title('Erdbeereis');
imwrite(bildErdbeer,'Aufgabe1_Erdbeereis.png');
```

Aufgabe 2

In Aufgebe 2 wird ein Audiofile bearbeitet. Nach dem einlesen wir es ins Spektrum transformiert und entstört. Zum Schluss wird das gefilterte Audiofile wiedergegeben und gespeichert. Zur Verdeutlichung wird das Signal vor und nach dem Filtern im Zeitbereich wie auch im Frequenzbereich dargestellt.

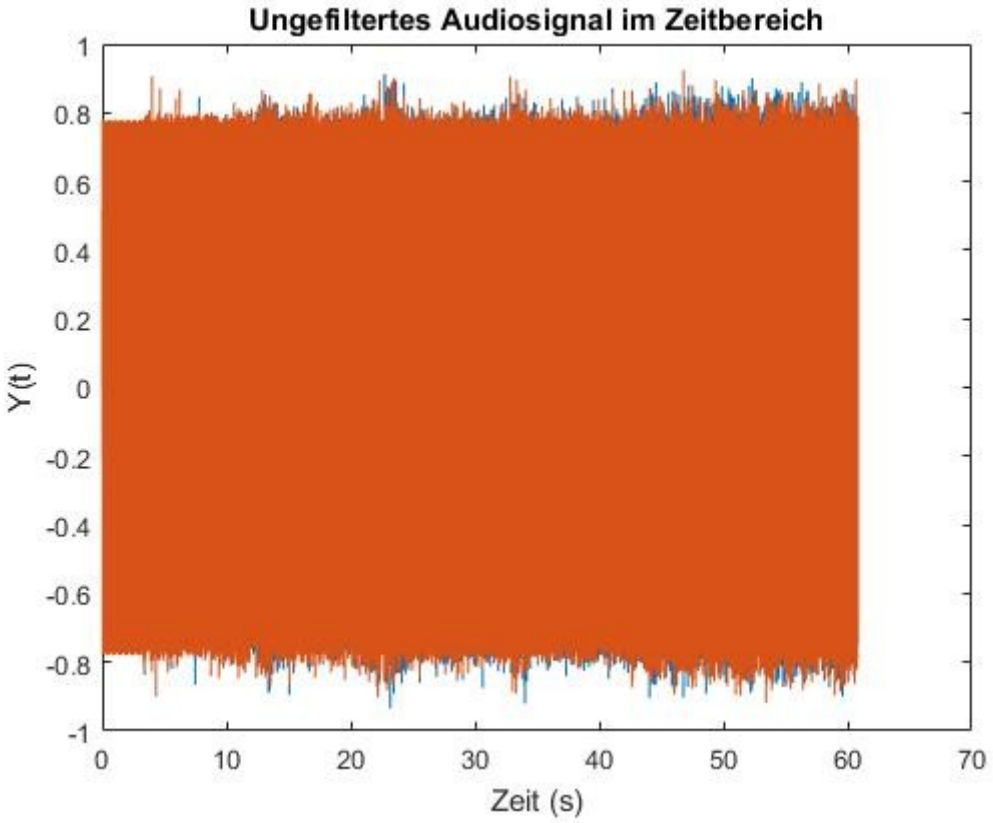
Dateien:
- Aufgabe2.m
- Aufgabe2.wav
- Aufgabe2Entstoert.wav
- Ungefiltertes_Audiosignal_Zeitbereich.jpg
- Ungefiltertes_Spektrum.jpg
- Gefiltertes_Audiosignal_Zeitbereich.jpg
- Gefiltertes_Spektrum.jpg

Einlesen des Audiofiles und auslesen der Abtastrate. Anschließend wird die Signallänge berechnet, um die Wiedergabedauer für die X-Achse im Zeitbereich du definieren.

```
%Audiodatei einlesen
[tonspur,abtastrate]=audioread('Aufgabe2.wav');
%Signallänge berechnen
signallaenge=size(tonspur,1)/abtastrate;
```

Für das Darstellen des Signals im Zeitbereich wird zuerst die X-Achse definiert (*tSigPlot*). Problem hierbei ist, dass der Wertebereich von 0 bis zur Signallänge geht und somit ein wert zu viel in der Achse definiert wird. Dies wird mit *signallaenge-1/abtastrate* behoben, indem die Signallänge um ein Element verringert wird.

```
%Darstellen des Audiosignals im Zeitbereich
tSigPlot=0:1/abtastrate:signallaenge-1/abtastrate;
subplot(2,2,1);
plot(tSigPlot,tonspur);
title('Ungefiltertes Audiosignal im Zeitbereich');
xlabel('Zeit (s)');
ylabel('Y(t)');
```

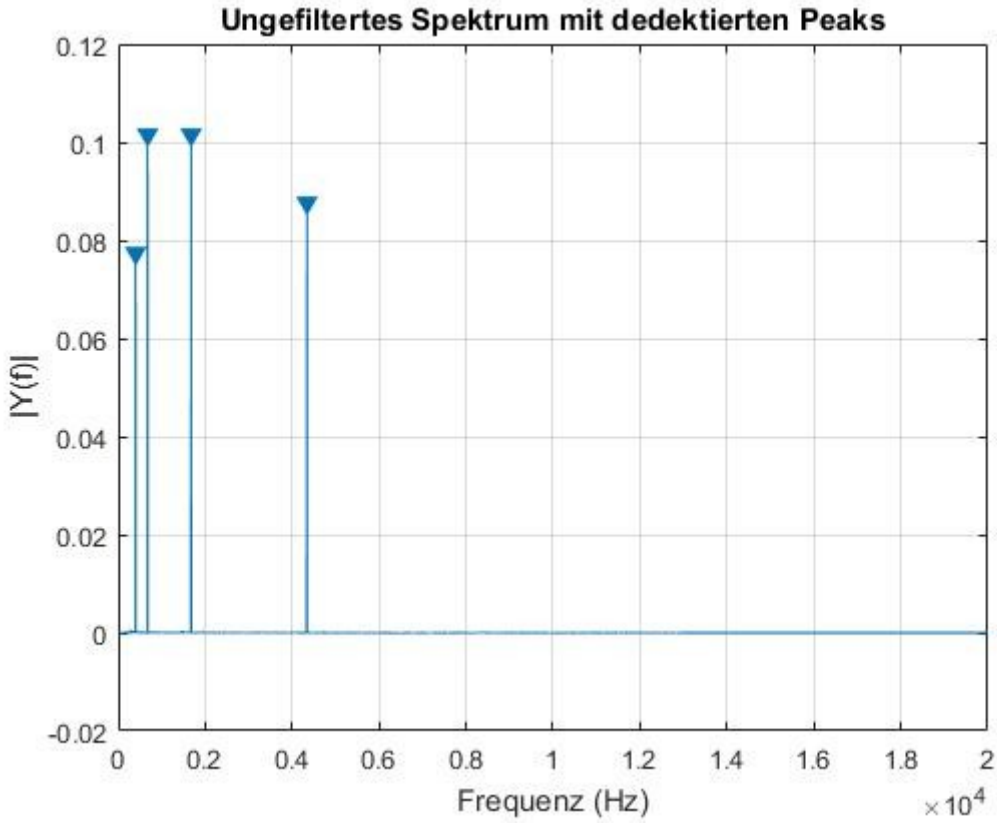


Im Anschluss daran wird das signal in das Spektrum Transfornmiert. Hier wird zuerst die *fftRate* berechnet, welche für die Fourier Transformation benötigt wird. Mit *fft(tonspur,fftRate)* wird eine Fourier Transformation durchgeführt und mit */fftRate* wird das Spektrum Normiert.

```
%FFT
fftRate=size(tonspur,1);
spektrum =fft(tonspur,fftRate)/fftRate;
```

Zum Darstellen des Spektrums muss zunächst eine Frequenzachse definiert werden. Die Frequenzachse hat als maximaler Wert 20kHz und wird mit *linspace(Anfang,Ende,Anzahl)* generiert. *linspace()* sorgt für eine gleichmäßige Verteilung der Werte vom Anfang bis zum Ende mit der Anzahl an Werten, die alle drei in der Klammer angegeben werden. Bevor das Spektrum geplottet wird, wird es mit *fftshift(spektrum)* in die richtige Form gebracht. Zum Ausblenden des negativen Spektrums werden alle Werte von 0 bis zur Hälfte des Spektrums nun gelöscht. Jetzt wird das Spektrum mit *plot(fAchse,abs(spektrumPeaks))* geplottet. Zum Anzeigen der Peaks wir die Funkion *findpeaks* verwendet, durch den Parameter *'Threshold'* kann eine minimale Distanz zum nächsten Peak eingestellt werden, damit nicht alle Frequenzen als Peak detektiert werden.

```
%Plot detektierte Peaks
subplot(2,2,2);
fAchse=linspace(0,20000,size(spektrum,1)/2);
spektrumPeaks=fftshift(spektrum);
spektrumPeaks=spektrumPeaks((size(spektrum,1)/2)+1:size(spektrum,1),:);
plot(fAchse,abs(spektrumPeaks)); % Plot Amplitudenspektrum
findpeaks(abs(spektrumPeaks(:,1)),fAchse,'Threshold',1e-3);
title('Ungefiltertes Spektrum mit dedektierten Peaks');
xlabel('Frequenz (Hz)');
ylabel('|Y(f)|');
```



Um die Störfrequenzen gerundet auszugeben wird wieder die Funktion *findpeaks* verwendet, diesmal aber mit Variablen in denen die Eckdaten gespeichert werden. In der Variable *locs* befinden sich die stellen aller detektierten Störfrequenzen. Um den Wert gerundet auszugeben hilft die Funktion *round(locs)*. Durch weglassen des Semikolons werden diese Werte dann direkt ausgegeben.

```
%Peaks ausgeben
[pks,locs,mittlereBreite,p] = findpeaks(abs(spektrumPeaks(:,1)),fAchse,'Threshold',1e-3);
disp('Störfrequenzen:')
round(locs)
```

Nun kann mit dem Filtern der Störer begonnen werden. Dazu wird zuerst das komplexe Spektrum aufgeteilt in einen realen und einen imaginären Anteil. *real()* bzw, *imag()* liefern die jeweiligen Komponenten.

```
%Spektrum in Imaginär- und Realteil aufteilen
spektrumReal=real(spektrum);
spektrumImag=imag(spektrum);
```

Zum Filtern wird die Anzahl aller Frequenzen im Spektrum benötigt. Diese wird folgendermaßen berechnet.

```
%Frequenzen
f=linspace(0,size(spektrum,1),size(spektrum,1));
```

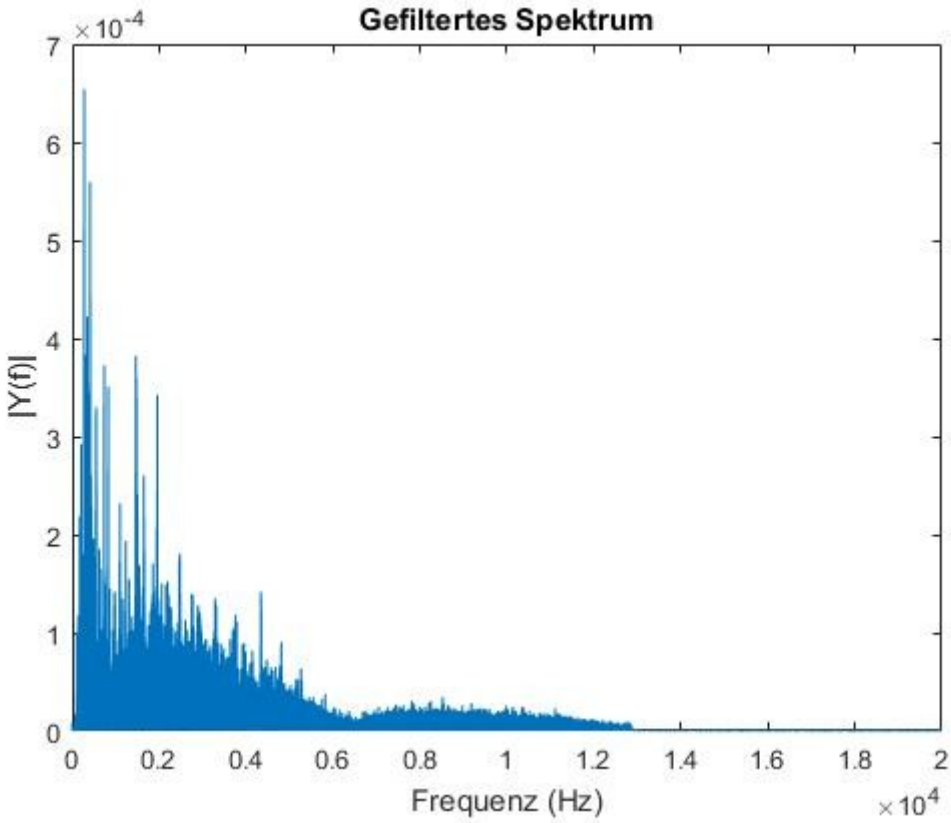
Nun werden die jetzt Zwei Spektren nacheinander gefiltert. Die Funktion *findpeaks* liefert wieder mit *locs* die Stellen der Störer. Nun hat man einen Vektor mit den stellen der Störer angibt, aber da man ja nicht nur den einen Wert löschen muss sondern auch links und rechts vom Peak muss eine Funktion für die zu löschende Bandbreite her. Die Funktion *findpeaks* liefert unter Anderem auch die mittlere Peakbreite (*mittlereBreite*). Diese Werte werden mit dem Faktor 150 multipliziert, damit von dem Peak nichts mehr übrig bleibt. Jetzt können mit den Stellen (*locs*) und der *bandbreite* die zu löschenden Bereiche definiert werden. Da man die *bandbreite* nicht auf einen Vektor anwenden kann muss eine For Schleife zur Unterstützung herangezogen werden um alle detektierten Peaks dann nacheinander zu entfernen. Dies wird für das reale und das Komplexe Spektrum angewendet.

```
%Filtern reales Spektrum
[pks,locs,mittlereBreite,p] = findpeaks(abs(spektrumReal(:,1)),f,'Threshold',1e-3);
locs=int32(locs);
for x = 1:size(locs,2)
    bandbreite=int32(mittlereBreite(x)*150);
    st=locs(x)-bandbreite:1:locs(x)+bandbreite;
    spektrumReal(st,:)=0;
end
```

Nachdem die beiden Spektren getrennt voneinander gefiltert wurden, werden sie jetzt wieder zu einem zusammengefasst. Die Funktion *complex(Realtei, Imaginärteil)* macht aus den Zwei Spektren wieder ein Komplexes.

```
%Komplexes Spektrum wieder zusammensetzen
spektrumZsm=complex(spektrumReal,spektrumImag);
```

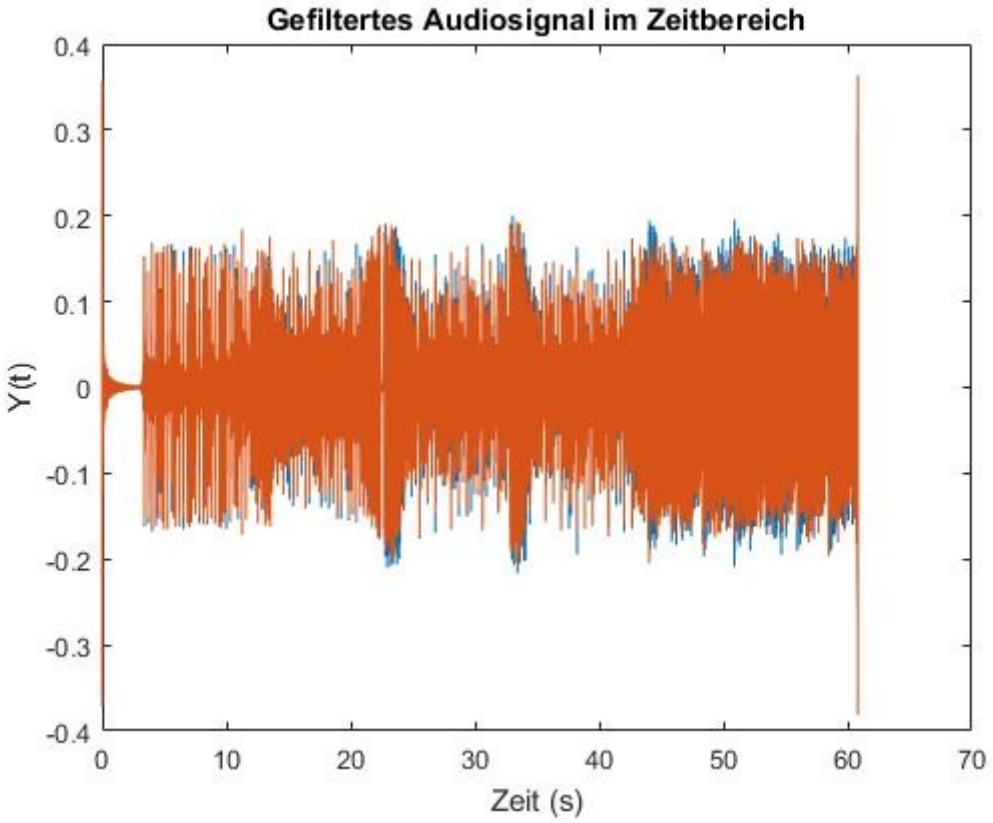
Danach wird das Spektrum wieder wie oben im Frequenzbereich dargestellt, diesmal aber sind die Störfrequenzen herausgefiltert.



Jetzt wird mittels *ifft()*, einer inversen Fourier Transformation, ein gefiltertes Audiosignal erzeugt.

```
%IFFT
tonNeu=real(ifft(spektrumZsm,fftRate)*fftRate);
```

Das neue Audiofile wird dann im Zeitbereich dargestellt, mit *sound(tonNeu,abtastrate);* abgespielt und mit *audiowrite('Aufgabe2Entstoert.wav',tonNeu,abtastrate);* gespeichert.



Aufgabe 3

In Aufgabe 3 wir eine Helix 3 Dimensional und in der Draufsicht geplottet.

Dateien: Aufgabe3.m
Helix_3D.jpg
Helix_Draufsicht.jpg

Eine Helix ist eine Überlagerung zweier Funktionen. Auf einer Ebene überlagert sich die X-Achse (Sinus) mit der Y-Achse (Cosinus). Doch zuerst müssen die grundeinstellungen für die Helix vorgenommen werden.

```
%Grundeinstellungen
laenge=50;
windungen=10;
```

Die Länge definiert die Zeit-Achse (Z) und die Windungen wie viele Windungen am ende der Länge abgeschlossen sein sollen.

```
%Zeitachse
t=[0:0.01:laenge];
```

Durch die Überlagerung von Sinus (x=sin(wt)) und Cosinus (y=cos(wt)) erhält man einen Kreis. Um jetzt eine Spirale zu erzeugen muss man die Z-Achse mit den beiden Funktionen als Vorfaktor verknüpfen. Jedoch soll bei einer Länge von 50 ein Radius von 30 erreicht sein (Durchmesser = 60 = Amplitude). Deshalb muss man erst einen Vorfaktor berechnen: a=Enddurchmesser/Länge → a=60/50=1.2.

```
%Amplitude
a=1.2*t;

%X-Funktion
x=a.*sin(w*t);

%Y-Funktion
y=a.*cos(w*t);
```

Der Punkt vor dem Multiplikationszeichen ist für eine elementweise Multiplikation.

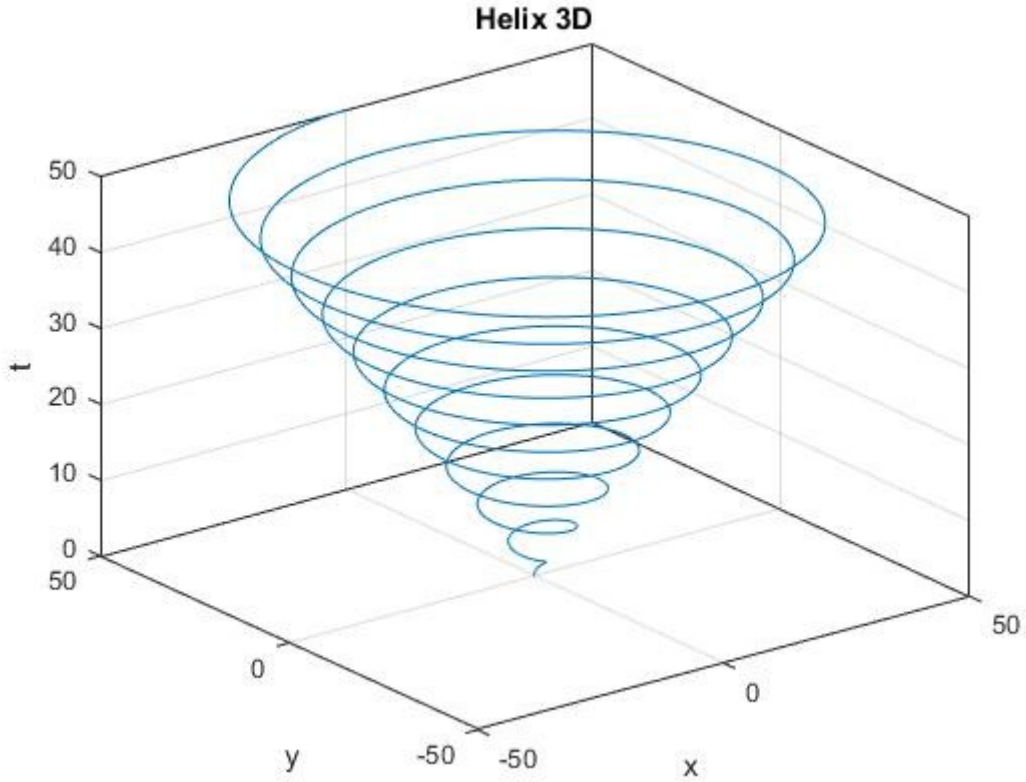
Jetzt fehlt nur noch w. Anhand der Grundeinstellungen sollen nach der Länge 50 (Z) 10 Windungen vorliegen. Eine Windung sind im Bogenmaß 2pi, dies folgt dass nach 10 Windungen (Perioden) 20pi zurückgelegt worden sein. Verbindet man dies jetzt mit der Länge und der Kreisfrequenz kommt man zu folgender Formel: Länge*w=Windungen*2pi. Demnach ist die Kreisfrequenz:

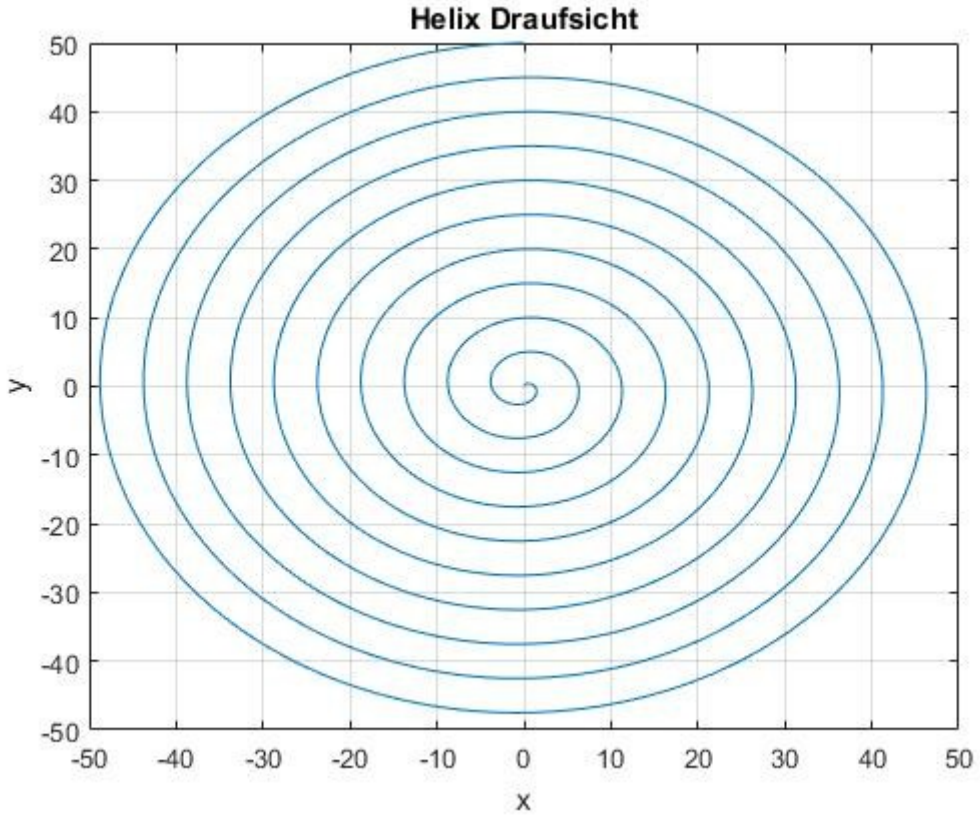
```
%Kreisfrequenz berechnen
w=2*(windungen/laenge)*pi;
```

Nun kann die Funktion geplottet werden. Mit `plot3(x,y,t);` kann eine 3-Dimensionale Funktion geplottet werden.

```
%Plot Helix 3D
figure(1);
plot3(x,y,t);
box on;
grid on;
title('Helix 3D');
xlabel('x');
ylabel('y');
zlabel('t');
```

Für die Draufsicht muss man das ganze noch mit `view(2);` erweitern.


Helix 3D
50
40
30
20
10
0
t
50
0
y
-50
-50
0
x
50


Helix Draufsicht
50
40
30
20
10
0
-10
-20
-30
-40
-50
y
-50
-40
-30
-20
-10
0
10
20
30
40
50
x

Aufgabe 4

Hier wird ein binäres Bild, welches einen verschlüsselten Text enthält entschlüsselt. Der Text wurde binär mit einem von einem PN-Generator erstellten Pseudo Rauschen verknüpft und rückwärts eingelesen.

Dateien: Aufgabe4.m
Aufgabe4.png

Erst wird das Bild eingelesen und die *pixelanzahl* bestimmt.

```
%Bild einlesen
bild = imread('Aufgabe4.png');

%Daten auswerten
pixelanzahl=size(bild,1)*size(bild,2);
```

Da der PN-Generator einen Vektor und keine Matrix ausgibt wird die eingelesene Bild-Matrix auch in ein Vektor gewandelt. Damit der Umwandlungsvorgang richtig funktioniert, muss die Bildmatrix zu ihrer Transponierenden geändert werden. Dies übernimmt das anhängen des Apostrophs mit dem Punkt für eine elementweise Umwandlung (.').

```
%Transponierte Erzeugen, damit im reshape vorgang die Matrix richtig
%abgearbeitet wird
bild=bild.';
```

Zur Verdeutlichung:

```
a=[1 2 ; 3 4; 5 6]
x=size(a,1)*size(a,2);
b=reshape(a,x,[])
c=reshape(a.',x,[])
```

Die Matrix a hat 3 Zeilen und 2 Spalten. Die Funktion *reshape()* wird verwendet mit den angegeben werten um aus der Matrix einen Vektor zu machen. Der Vektor b wurde ohne die Transponierende und der Vektor c mit der Transponierenden erzeugt.
Inhalt b=135246 Inhalt c =123456. Da die Bits der Matrix wie in der deutschen Sprache von links nach rechts und von oben nach unten gelesen werden soll ist der Weg für Vektor c der Richtige.

Jetzt wird die Bildmatrix in einen Vektor umgewandelt.

```
%Binäre Matrix in einen Vektor umwandeln
binDaten=reshape(bild,pixelanzahl,[]);
```

Da man jetzt einen Vektor mit den binären Daten hat kann die Entschlüsselung mit dem PN Generator beginnen. Der PN Generator benötigt zuerst ein paar grundlegende Parameter ein Generatorpolynom für die xor Rückführung des Schieberegisters und der Initialwert des Registers.

```
%Parameter
generatorpolynom=[12 8 2 0];
initialwert=[1 0 0 0 1 0 0 0 0 0 1 0];
```

Jetzt wird die Klasse des PN Generators verwendet. Mit *'Polynomial'* wird das Generatorpolynom angegeben, mit *'VariableSizeOutput', false,* wird angegeben falls der Wert ‚false' ist dass die Länge des Outputs durch *'SamplesPerFrame', pixelanzahl,* definiert werden kann. Der Initialwert gibt man hinter dem Parameter *'InitialConditions'* an.

```
%PN-Klasse
pnFunktion = comm.PNSequence(   'Polynomial', generatorpolynom,...
                                'VariableSizeOutput', false,...
                                'SamplesPerFrame', pixelanzahl,...
                                'InitialConditions', initialwert);
```

Die *pnFunktion* wird dazu verwendet das binäre Rauschen zu erzeugen. Mit *step()* erzeugt man die Antwort eines Systems, hier einer Klasse die sich in den Klammern befindet. In diesem Fall erzeugt der Befehl dass in der Variable *pnReihe* ein Vektor erzeugt wird mit so vielen Werten wie die Binärmatrix des Bildes Pixel (Werte) hat.

```
%PN-Reihe erzeugen
pnReihe = step(pnFunktion);
```

Nun können die binären Daten mit dem Pseudo-Rauschen verknüpft und somit entschlüsselt werden. Die Verknüpfung ist eine Exklusiv-Oder-Verknüpfung (*xor()*).

```
%Binäre Bilddaten mit der PN reihe entschlüsseln
binErgebnis=xor(binDaten,pnReihe);
```

Jetzt müssen die entschlüsselten Daten nur noch in das richtige Format gebracht werden. Dazu wird wieder die *reshape()* Funktion verwendet und die Matrix wird dann gedreht um 8 Spalten zu bekommen.

```
%Binärdaten in einen Matrix mit der Breite des ASCII Formats (8 Bit) umwandeln
binText=reshape(binErgebnis,8,[]).';
```

Die Matrix wird dann in das Char Format gewandelt, damit die Zeilen der Matrix zusammengefasst werden.

```
%Double Format in char Format ändern
binText=num2str(binText);
```

Nun werden die char-binären Zeilen in Dezimalzahlen umgewandelt.

```
%Binärblöcke der Matrix in Dezimalzahlen umwandeln
decText=bin2dec(binText);
```

Dezimalzahlen mit der Funktion *char()* in Buchstaben umwandeln und von einem Zeilenvektor in einen Spaltenvektor umwandeln, damit der Text besser gelesen werden kann.

```
%Dezimalzahlen in Buchstaben wandeln und zu einem Spaltenvector formatieren
text=char(decText).';
```

Jetzt muss der Text nur noch einmal in der Mitte geflippt werden, da er ja beim verschlüsseln falsch herum eingelesen wurde.

```
%Text umdrehen
text=flip(text);
```

Die Entschlüsselung ist nun fertig und der Text wird ausgegeben.

```
%Text ausgeben
disp(text);
```

Text:

Theodor Fontane 1819-1898
John Maynard

Wer ist John Maynard?
John Maynard war unser Steuermann,
Aus hielt er, bis er das Ufer gewann,
Er hat uns gerettet, er trägt die Kron,
Er starb für uns, unsre Liebe sein Lohn.
John Maynard.
*
Die "Schwalbe" fliegt über den Eriesee,
Gischt schäumt um den Bug wie Flocken von Schnee;
Von Detroit fliegt sie nach Buffalo
Die Herzen aber sind frei und froh,
Und die Passagiere mit Kindern und Fraun
Im Dämmerlicht schon das Ufer schaun,
Und plaudernd an John Maynard heranTritt alles: "Wie weit noch, Steuermann?"
Der schaut nach vorn und schaut in die Rund:
"Noch dreißig Minuten ... halbe Stund."
Alle Herzen sind froh, alle Herzen sind frei
Da klingt´s aus dem Schiffsraum her wie Schrei,
"Feuer!" war es, was da klang,
Ein Qualm aus Kajüt und Luke drang,
Ein Qualm, dann Flammen lichterloh,
Und noch zwanzig Minuten bis Buffalo.
Und die Passagiere, buntgemengt,
Am Bugspriet stehn sie zusammengedrängt,
Am Bugspriet vorn ist noch Luft und Licht,
Am Steuer aber lagert sich´s dicht,

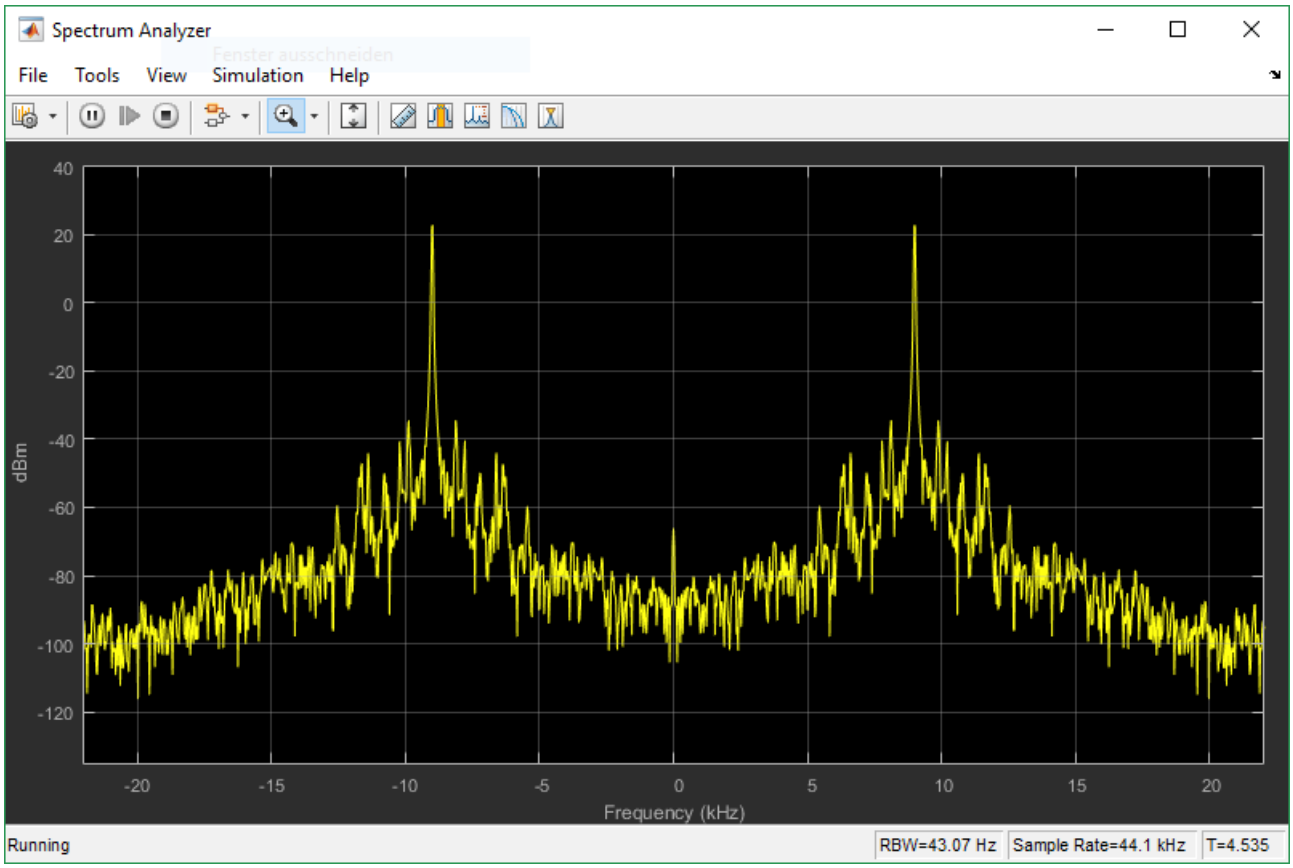
Und ein Jammern wird laut: "Wo sind wir? Wo?"
Und noch fünfzehn Minuten bis Buffalo.
Der Zugwind wächst, doch die Qualmwolke steht,Der Kapitän nach dem Steuer späht,
Er sieht nicht mehr seinen Steuermann,
Aber durchs Sprachrohr fragt er an:
"Noch da, John Maynard?"
"Ja, Herr. Ich bin."
"Auf den Strand! In die Brandung!"
"Ich halte drauf hin."
Und das Schiffvolk jubelt: "Halt aus! Hallo!"
Und noch zehn Minuten bis Buffalo.
"Noch da, John Maynard?" Und Antwort schallt´s
Mit ersterbender Stimme: "Ja, Herr, ich halt´s!"
Und in die Brandung, was Klippe, was Stein,
Jagt er die "Schwalbe" mitten hinein.
Soll Rettung kommen, so kommt sie nur so.
Rettung: der Strand von Buffalo!
*Das Schiff geborsten. Das Feuer verschwelt.
Gerettet alle. Nur einer fehlt.
*
Himmelan aus Kirchen und Kapell´n,
Ein Klingen und Läuten, sonst schweigt die Stadt,
Ein Dienst nur, den sie heute hat:
Zehntausend folgen oder mehr,
Und kein Aug im Zuge, das tränenleer.
Sie lassen den Sarg in Blumen hinab,
Mit Blumen schließen sie das Grab,
Und mit goldner Schrift in den Mamorstein
Schreibt die Stadt ihren Dankspruch ein:
"Hier ruht John Maynard! In Qualm und Brand
Hielt er das Steuer fest in der Hand,;
Er hat uns gerettet, er trägt die Kron´,
Er starb für uns, unsre Liebe sein Lohn."
John Maynard!

Aufgabe 5

Ein Frequenzmoduliertes Audiosignal wird mittels einer PLL demoduliert und gespeichert.

Dateien: Aufgabe5.M
PLL.slx
output.wav
Spektrum.png

Zuerst wird mit dem Spektrumanalysator die Modulationsfrequenz bestimmt.
Modulationsfrequenz = 9kHz.



Für die spätere Diskretisierung des Audiosignals wird das mit 44100Hz abgetastete Signal auch mit 44100 Samples ausgegeben. Anschließend wird ein Unbuffer für die Spektrenanalyse gebraucht. Der Filter Muss entsprechend eingestellt werden, dass störende Signalanteile abgeschnitten werden. Die Eckfrequenz des Filters ist 2*pi*f.

```
% RF Parameter
f=9000; % Trägerfrequenz
```

Die Parameter des Loop-Filters müssen auf das Signal angepasst werden dabei sind zwei Werte besonders wichtig.

```
% Loop Parameter
Kd=0.5; % Definition des Phasendetektorgewinns
K0=2*pi*400; % Definition der VCO Konstante in rad/Hz
w0=2*pi*190; % Definition der PLL Eigenfrequenz
zeta=0.7; % Definition der Dämpfung der PLL Regelschleife
```

Die Multiplikatoren von K0 und w0 müssen sehr fein abgestimmt werden damit ein sinniges Audiosignal aus der Demodulierung heraus kommt.

Die Form des Filters ist (1 + tau2*s)/(1 + tau1*s), mit folgender Berechnung für tau.

```
tau1=K0*Kd/w0^2;
tau2=2*zeta/w0-1/(K0*Kd);
```

Wenn jetzt noch der Spannungsgesteuerte Oszillator richtig Parametriert ist mit der Frequenz f und der Eingangssensitivität $K0/(2*pi)$. Wird das Audiosignal richtig Demoduliert.

Da nur diskrete Signale gespeichert werden können und das Signal in der PLL kontinuierlich ist muss dies mit einem Zero-Order-Hold Baustein diskretisiert werden. Dazu muss die Sampletime angegeben werden. Wie oben beschrieben wurden beim Auslesen des modulierten Audiosignals 44100 Samples ausgegeben also ist die Sampletime 1/44100.

Nun kann das demodulierte, diskrete Audiosignal Gespeichert werden.