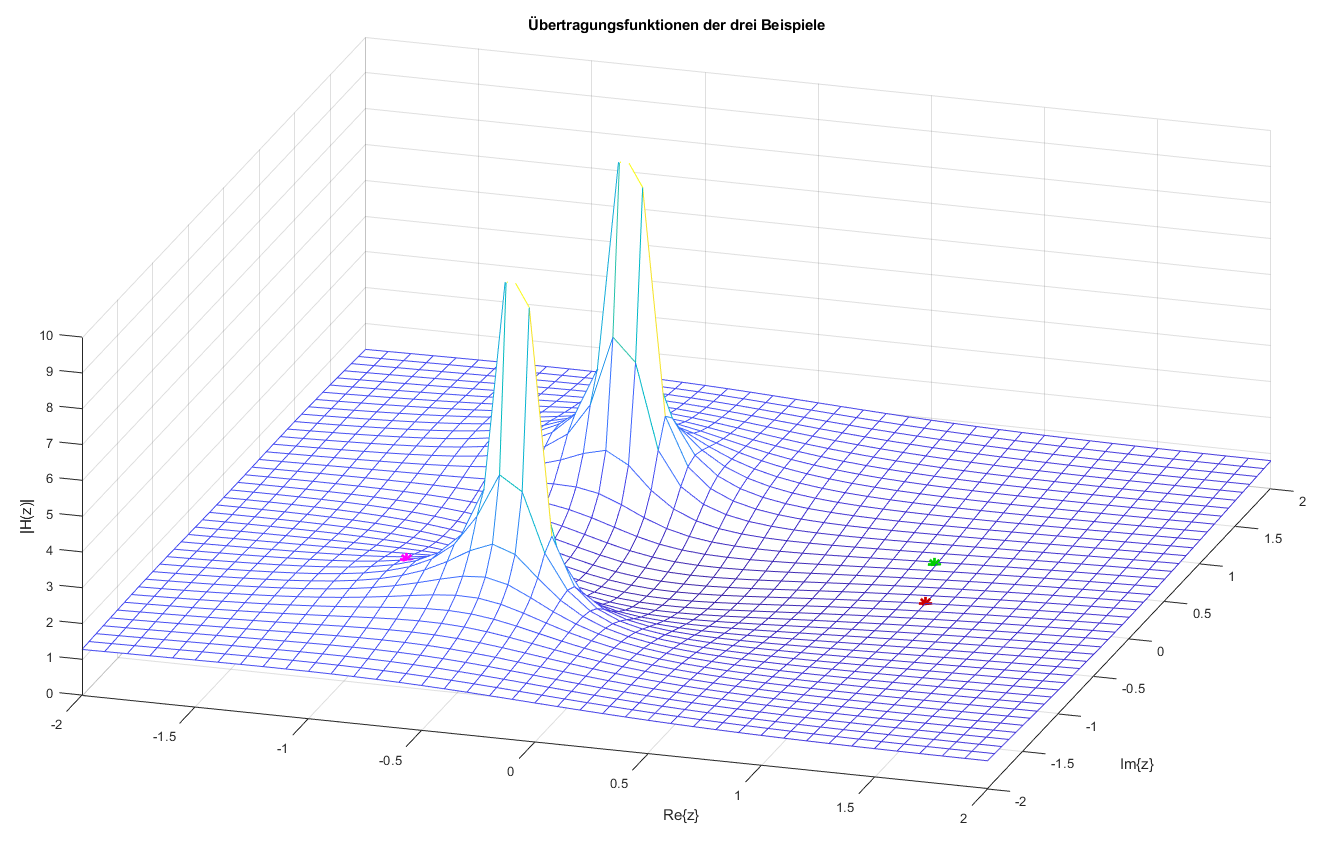


**Digitale Signalverarbeitung**

**MATLAB Rechenübung**

**Vorgelegt von:**Oskar Creutzer & Lennert Hahn

**Matrikelnummer:**  
S0569454 & S0568305



**Hochschule für Technik und Wirtschaft (HTW) Berlin**

**Studiengang:**Informations- und Kommunikationstechnik

**Semester:**Wintersemester 2020/21

**Stand:**21. Januar 2021

**Dozent:**Dipl. Ing. Felix Frey

Inhaltsverzeichnis

[1. Vorwort und Motivation 3](#_Toc62074929)

[2. Zeitdiskrete Signale 4](#_Toc62074930)

[2.2 Faltung 4](#_Toc62074931)

[2.3 Formatierung von Graphen 5](#_Toc62074932)

[2.4 Systemeigenschaften 6](#_Toc62074933)

[*Skalierer*  6](#_Toc62074934)

[*Offset (Gleichanteil)*  8](#_Toc62074935)

[*Zeitspiegelung*  9](#_Toc62074936)

[Differenzenbildung 9](#_Toc62074937)

[*Modulation*  11](#_Toc62074938)

[*Quantisierer*  12](#_Toc62074939)

[2.5 FIR (Finite Impulse Response) Systeme 13](#_Toc62074940)

[Testen für beliebige Eingangssignale mit Probe durch Faltung der Impulsantwort 15](#_Toc62074941)

[1. Zufallstestsignal 15](#_Toc62074942)

[2. Zufallstestsignal 16](#_Toc62074943)

[*IIR - Infinite Impulse Response* 18](#_Toc62074944)

[*Stabilitätsprobe der Übertragungsfunktion* 22](#_Toc62074945)

[**1.** Methode: Einheitskreis 22](#_Toc62074946)

[**2.**Methode: Darstellung im 3D-Plot (meshgrid) 23](#_Toc62074947)

[*FIR-System aus einem IIR-System* 25](#_Toc62074948)

[*Erregung mit nichtkausaler komplexer Exponentialfolge* 25](#_Toc62074949)

[25](#_Toc62074950)

[26](#_Toc62074951)

[27](#_Toc62074952)

[3. Zusammenfassung & Fazit 29](#_Toc62074953)

[4. Eigenständigkeitserklärung 30](#_Toc62074954)

# Vorwort und Motivation

Durch die aktuellen Bedingungen der Online-Lehre aufgrund von Covid-19 ist das Modul „Digitale Signalverarbeitung“ über ein Matlab-Labor-Protokoll zu einer Rechenübung der Lehrveranstaltung zu beenden. Das Ziel dieser Hausarbeit ist die eigenständige Erarbeitung von Grundlagen des „Computer-Algebra-Systems“ sowie das Erlangen neuer Kenntnisse, praxisnaher Fähigkeiten und Fertigkeiten anhand der Lehrveranstaltung sowie des Labor-Skripts von Herrn Dipl. Ing. Felix Frey.

In der vorliegenden Arbeit werden dafür zunächst zeitdiskrete Systeme charakterisiert und deren Systemeigenschaften betrachtet. Dazu zählen u.a. ihre Skalier-, Offset- und Zeitspiegelungseigenschaften. Daraufhin werden ein nichtrekursives System (Finite Impulse Response- bzw. FIR-System) und ein rekursives System (Infinite Impulse Response- bzw. IIR-System) implementiert.

Während der Durchführung wurde das IIR-System zunächst auf Stabilitäts-Eigenschaften untersucht und im Anschluss anhand nichtkausaler komplexer Exponentialfolgen erregt.

Die Ergebnisse werden abschließend noch zusammengefasst und es wird ein Fazit gezogen.

# Zeitdiskrete Signale

## Faltung

Innerhalb einer Funktion sollte anhand einer For-Schleife eine Faltungsoperation durchgeführt werden, und zwar mithilfe von zwei endlich langen, kausalen Signalen. Hierbei war darauf zu achten, dass der Vektorenindex in Matlab bei eins indiziert wird und zunächst beide Vektoren auf dieselbe Länge gebracht werden, ehe daraufhin die Länge des Ausgangsvektors bestimmt werden konnte. Die eigentliche Faltungsoperation wird durch eine if-Abfrage initiiert, wonach ein gegeneinander Laufen und berechnen der beiden Vektoren durchgeführt wird.  
Schlussendlich wurde anhand des Matlab-Befehls *conv(x,u)* dann überprüft, ob die Faltungsoperation korrekt war.

x = [5 4 3 2 1];

u = [1 1 1];

y = faltung(x,u);

function y = faltung (x,u)

%faltet x mit u

%beide Vektoren auf die selbe Länge bringen

f\_x = [x, zeros(1,length(u))];

f\_u = [u, zeros(1,length(x))];

%Länge des gefalteten Signals bestimmen

len\_y = length(x) + length(u)-1;

y = zeros(1,len\_y);

% ein Durchlauf pro Ausgabeelement in y

for i = 1:len\_y

% Suche nach der zu faltenden Stelle durch gegeneinander laufen beider Vektoren

for j = 1:length(f\_x)

if((i-j+1)>0)

y(i) = y(i) + f\_x(j) \* f\_u(i-j+1);

end

end

end

end

%Probe

%falt = conv(x,u)

## Formatierung von Graphen

In der Aufgabenstellung wurde gefordert, dass das Ergebnis der Faltung graphisch dargestellt wird. Dazu haben wir eine Aufteilung in *Subplots* gewählt, bei der die einzelnen Graphen in der unten erkennbaren Struktur in einem Bild (*figure*) angeordnet werden. Die Graphen wurden jeweils als *Stem*-plot umgesetzt, da dies einer zeitdiskreten Darstellung sehr nahe kommt. Eine Limitierung der X- und Y-Achse musste nicht vorgenommen werden, da alle Diagrammbereiche ausreichend erkennbar waren. Die selbstgeschriebene Funktion *stem\_properties* fasst dabei jedoch die Befehle für eine Achsenbeschriftung und den Diagrammtitel in einer Zeile zusammen.

figure (1);

subplot(2,2,1)

stem(x,'Color',[0 0.5 0])

stem\_properties('Faltung von X mit U --> X','Zeit / s','Amplitude',x);

subplot(2,2,2)

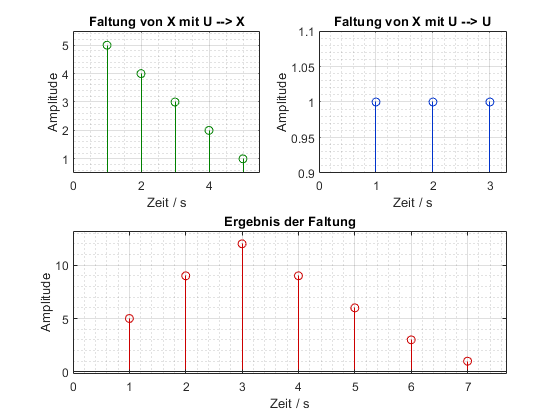
stem(u,'Color',[0 0.2 0.8])

stem\_properties('Faltung von X mit U --> U','Zeit / s','Amplitude',u);

subplot(2,2,[3,4])

stem(y,'Color',[0.8 0 0])

stem\_properties('Ergebnis der Faltung','Zeit / s','Amplitude',y);



## Systemeigenschaften

Für die folgenden Beobachtungen der diskreten Systemeigenschaften sollen verschieden Eingangssignale als Testsignale generiert werden. Dazu werden zwei Vektoren der Länge *nSymbols* mit zufälligen Zahlen gefüllt. Beide pseudzufälligen Operationen starten mit unterschiedlichem *Seed*, um durch die deterministische „Zufallsberechnung“ zwei unterschiedliche diskrete Vektoren (*data1 & 2*) zu erhalten.  
Der erzeugte Dirac Impuls, mit derselben Länge, wie der erste Datenvektor, ist mit Nullen gefüllt. Lediglich an der vorher bestimmten Position wird ein diskreter Wert auf eins gesetzt. Diese Position (*dirac\_1\_pos*) ist auf die Hälfte der gesamten Vektorlänge gesetzt, um später die Kausalität feststellen zu können.

nSymbols = 2^3;

%Erzeugung von 2 zufälligen Signalen

rng(1)

data1 = randn(1, nSymbols);

rng(13) % gleicher Seed bedeutet gleiche Zufallszahlenfolge

data2 = randn(1, nSymbols);

%Dirac zur späteren Kausalitätsprüfung

dirac\_1\_pos = floor(length(data1)/2);

dirac = zeros(1, length(data1));

dirac(dirac\_1\_pos) = 1;

Bei allen folgenden Systemen werden zuerst die entsprechenden Parameter der Funktion definiert, danach zwei Ausgangsvektoren aus den beiden zufälligen Eingangsvektoren und dem System berechnet und anschließend die Systeme auf die Eigenschaften Linearität, Zeitinvarianz und Kausalität geprüft. Diese Eigenschaften werden einmalig beim Skalierer vorgestellt und abschließend für alle Systeme am Ende zusammengefasst.

# *Skalierer*

Ein skalierendes System (*scaler*) haben wir als elementweise Multiplikation des Vektors mit einem einstellbaren Skalar *a* realisiert.

a = 2;

function y = scaler(x, a)

% skaliert den Eingangsvektor "x" mit "a"

y = a.\*x; %elementweise Operation

end

scaled\_y1 = scaler(data1, a);

scaled\_y2 = scaler(data2, a);

Ein lineares System ist prinzipiell eine Überlagerung zweier Eigenschaften: die der Addition und der Skalierung. Somit lässt sich ein Vergleich mit *isequal* anstellen, bei dem überprüft wird, ob zwei Signale identisch sind. Zum einen eine Addition der skalierten Vektoren und zum anderen eine Skalierung der Bereits addierten Vektoren. Somit ergibt sich eine *1* bei der Erfüllung dieser Eigenschaft.

% Linearitätsprüfung (linear = 1)

isequal(scaled\_y1 + scaled\_y2 , scaler((data1+data2), a))

ans = *logical*

1

Die Prüfung auf Zeitinvarianz gibt Auskunft darüber, ob das System auch bei einer Zeitlichen Verschiebung des Eingangssignals dieselbe Verschiebung des Ausgangssignals hervorruft. So haben wir eine Zeitverschiebung *(t0)* von 5 Werten vorgegeben und verschieben mittels *circshift* sowohl das Ausgangssignal, als auch das Eingangssignal, bei dem Danach die Operation des Systems noch ausgeführt wird. Nach einem Vergleich der beiden Signale mit *isequal* wird die Zeitinvarianz mit einer *1* bestätigt.

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(scaled\_y1, -t0);

eingang = scaler(circshift(data1, -t0), a);

isequal(ausgang, eingang)

ans = *logical*

1

Die Kausalität eines Systems sagt aus, ein Eingangssignal, welches zu einem definiertem Zeitpunkt t gesendet wird am Ausgang vor diesem Zeitpunkt keinen Pegel hervorruft. Dies testen wir mit einem verschobenen Diracimpuls, der am Anfang von Kap. 2.4 erklärt wurde. Mittels einer Schleife wird überprüft, ob sich Werte ungleich Null vor diesem festgelegten Dirac im Ausgangsvektor befinden. Sollte dies nicht der Fall sein, ist das System kausal (=1).

% Kausalitätsprüfung (kausal = 1)

dir\_sca = scaler(dirac, a);

function kausal = kausalitaet(x)

%prüft ein System auf Kausalität mit Bezug auf einem Dirac als Eingangssignal

for k = 1:length(x)

if x(k)~= 0 %1.Prüfbedingung = x(k) =/= 0

if k < floor(length(x)/2) %2.Prüfbedingung = k ist vor dem einzelnen Eingangsdiracimpuls

kausal = 0

return

else

kausal = 1

return

end

end

end

end

kausalitaet(dir\_sca);

kausal = 1

# *Offset (Gleichanteil)*

Ein Offset ist ein System, welches einen Gleichanteil auf jeden Wert des Signals hinzufügt. So haben wir einen variablen Offsetwert *o* durch eine simple Addition auf jedes Element des Vektors addiert.

o = 3;

function y = offset(x, o)

% addiert einen Offset "o" zum Eingangssignal "x"

y = o + x;

end

offset\_y1 = offset(data1, o);

offset\_y2 = offset(data2, o);

% Linearitätsprüfung (linear = 1)

isequal(offset\_y1 + offset\_y2 , offset((data1+data2), o))

ans = *logical*

0

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(offset\_y1, -t0);

eingang = offset(circshift(data1, -t0), o);

isequal(ausgang, eingang)

ans = *logical*

1

% Kausalitätsprüfung (kausal = 1)

dir\_off = offset(dirac, o);

kausalitaet(dir\_off);

kausal = 0

# *Zeitspiegelung*

Die Zeitspiegelung spiegelt das Signal an seiner Mittelachse. Dies kann durch die Matlab-Funktion *flip* bewerkstelligt werden.

mirror\_y1 = flip(data1);

mirror\_y2 = flip(data2);

% Linearitätsprüfung (linear = 1)

isequal(mirror\_y1 + mirror\_y2 , flip(data1+data2))

ans = *logical*

1

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(mirror\_y1, -t0);

eingang = flip(circshift(data1, -(-t0)));

isequal(ausgang, eingang)

ans = *logical*

1

% Kausalitätsprüfung (kausal = 1)

dir\_mir = flip(dirac);

kausalitaet(dir\_mir);

kausal = 1

# Differenzenbildung

Die Differenzbildung sieht in diesem Beispiel vor, dass ein Element vom Nachfolgenden subtrahiert wird. Dies haben wir mit Hilfe einer Schleife gelöst, die genau die beschriebene Operation in einen, mit Nullen initialisierten, Ausgangsvektor schreibt. Der Sonderfall an der Indexstelle 1 wird anhand der Annahme, dass das System kausal ist (x0 = 0) nicht im Wert verändert.

diff\_y1 = difference(data1);

function y = difference(x)

% y[k] = x[k]-x[k-1]

y = zeros(1,length(x)); %Initialisierung vom Ausgangsvektor

for k = 1:length(x)

if k == 1 %Annahme x(0)=0

y(1) = x(1);

else

y(k) = x(k)-x(k-1); %Differenzbildung mit dem vorherigen Element

end

end

diff\_y2 = difference(data2);

% Linearitätsprüfung (linear = 1)

a1 = diff\_y1 + diff\_y2;

a2 = difference(data1+data2);

isequal(round(a1,4) , round(a2,4)) % round wird wegen rundungsfehlern benutzt

ans = *logical*

1

%isequal(diff\_y1 + diff\_y2 , difference(data1+data2))

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(diff\_y1, -t0);

eingang = difference(circshift(data1, -t0));

isequal(ausgang, eingang)

ans = *logical*

0

% Kausalitätsprüfung (kausal = 1)

dir\_diff = difference(dirac);

kausalitaet(dir\_diff);

kausal = 1

# *Modulation*

Die Modulation ist vergleichbar mit einer Multiplikation mit einer Kosinus-Trägerschwingung. Diese Multiplikation haben wir auf Grund der Indizierung mit *k* im cos-Term durch eine Schleife gelöst, anstatt eine elementweise Operation durch den Punktoperator durchzuführen.

mod\_y1 = simple\_mod(data1);

mod\_y2 = simple\_mod(data2);

function y = simple\_mod(x)

% moduliert das Eingangssignal mit einer cos-Schwingung

y = zeros(1, length(x)); %Initialisierung vom Ausgangsvektor

for k = (1:length(x))

y(k) = x(k) \*cos((pi/4)\*k); %Modulation (Multiplikation)

end

end

% Linearitätsprüfung (linear = 1)

isequal(mod\_y1 + mod\_y2 , simple\_mod(data1+data2))

ans = *logical*

0

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(mod\_y1, -t0);

eingang = simple\_mod(circshift(data1, -t0));

isequal(ausgang, eingang)

ans = *logical*

0

% Kausalitätsprüfung (kausal = 1)

dir\_mod = simple\_mod(dirac);

kausalitaet(dir\_mod);

kausal = 1

# *Quantisierer*

Eine Quantisierung wird unter anderem bei der Umwandlung eines wertkontinuierlichen in ein wertdiskretes Signal angewendet. Da in unserer Matlab-Übung jedoch nur diskrete Signale zur Verfügung stehen ist die Quantisierung vereinfacht mit einer Rundung auf ganze Zahlen (bzw. Intger-Werte gleichzusetzen. Dies wird auch über die Matlab interne Funktion *round* gelöst.

quant\_y1 = round(data1);

quant\_y2 = round(data2);

% Linearitätsprüfung (linear = 1)

isequal(quant\_y1 + quant\_y2 , round(data1+data2))

ans = *logical*

0

% Zeitinvarianzprüfung (zeitinvariant = 1)

t0 = 5;

ausgang = circshift(quant\_y1, -t0);

eingang = round(circshift(data1, -t0));

isequal(ausgang, eingang)

ans = *logical*

1

% Kausalitätsprüfung (kausal = 1)

dir\_quan = round(dirac);

kausalitaet(dir\_quan);

kausal = 1

Die Auswertung der einzelnen Systeme auf die drei Eigenschaften lässt sich in folgender Tabelle zusammenfassen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Eigenschaft | | |
|  |  | Linearität | Zeitinvarianz | Kausalität |
| System | Skalierer | ✓ | ✓ | ✓ |
| Offset | 🞪 | ✓ | 🞪 |
| Zeitspiegelung | ✓ | ✓ | ✓ |
| Differenzbildung | ✓ | 🞪 | ✓ |
| Modulation | 🞪 | 🞪 | ✓ |
| Quantisierer | 🞪 | ✓ | ✓ |

## FIR (Finite Impulse Response) Systeme

Das Finite Impulse Response (FIR)-System aus der Klasse der diskreten Systeme, gehört den nichtrekursiven Systemen an und hat eine endlich lange Impulsantwort. Dies bedeutet, dass bei einem (kausalen) nichtrekursiven System der Ordnung , das Ausgangssignal zum Zeitpunkt nur vom aktuellen sowie dem letzten Eingangswert abhängt. Die zuvor produzierten Ausgangswerte werden hierbei nicht in Abhängigkeit gezogen.

Zum Testen des FIR-Systems eignet sich ein Dirac-Impuls, welcher durch das System folgende Impulsantwort lieferte

Bei diesen Systemen ist die Ein-/ Ausgangsbeziehung in Form der untenstehenden Gleichung gegeben, welche die Faltung des Eingangssignals mit der Impulsantwort darstellt.

Die Impulsantwort des Systems wird zunächst anhand mehrerer Anregungen des Systems mit dem diskreten Dirac-Impuls unter Verwendung des Vektors *coef = [1 2 3 2 1]* implementiert.

Die Implementierung des FIR wurde als "tapped-delay-line", also der Direktform, erstellt. Dabei wir das Eingangssignal um die Anzahl der Ordnung mit Nullen erweitert.

Im nachfolgendem Code wird der IIR-Filter als erste Direktform realisiert, wodurch zwei Register (*reg1 & 2*) benötigt werden, die jeweils in den beiden Zweigen die Elemente zur Verarbeitung laden. Das Zwischenergebnis vom Vorwärtszweig wird in *u[k]* gespeichert und dient dem Rückkoppelzweig als Eingangssignal, der daraus das Ausgangssignal in *y[k]* speichert. Mittels *circshift* werden die beiden Register weitergetaktet, um das nächste Element des Eingangsvektors einzulesen.

n\_Symbols = 2^5;

%Dirac-Impuls

dirac\_puls = zeros(1, n\_Symbols);

dirac\_puls(1) = 1;

coef = [1 2 3 2 1]; %Vorwärtszweig

fir\_system = FIR(dirac\_puls, 4, coef);

function y = FIR(x,ord,coef)

%FIR-Filter (finite impulse response)

% coef = [a0 a1 a2 a3 a4] bei n=4

n = ord;

a = coef;

len = (length(x)+n);

x\_long = zeros(len); %Verlängerung des Eingangssignals mit n Nullen

x\_long(1:length(x)) = x;

y = zeros(1,len); %Initialisierung des Ausgangs

reg = zeros(1,n+1); %Register mit allen Delayelementen auf 0

%Realisierung des FIR\_Systems als "tapped-delay-line" (Direktform)

for k = 1:len

reg(1) = x\_long(k); %1. Registerstelle füllen

y(k) = sum(a .\* reg); %Registerwerte mit Koeffizienten multiplizieren und aufsummieren für aktuellen Ausgang

reg = circshift(reg, 1); %Register um eine Stelle verschieben

end

end

%zum Probe ([1 0 0 0 0] = Rückwärtszweig)

%fir\_system = filter(coef, [1 0 0 0 0], dirac\_puls);

figure (2)

subplot(2,1,1)

stem(dirac\_puls)

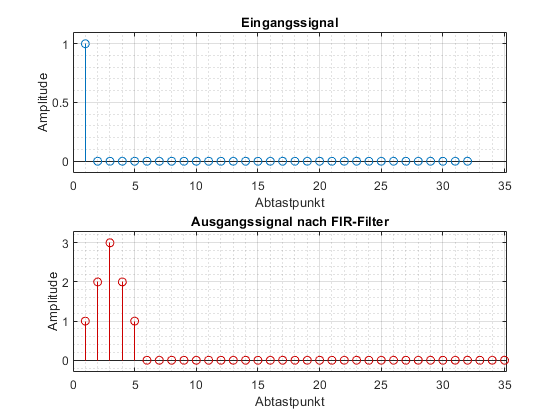
stem\_properties('Eingangssignal', 'Abtastpunkt', 'Amplitude', dirac\_puls)

subplot(2,1,2)

stem(fir\_system, 'Color',[0.8 0 0])

stem\_properties('Ausgangssignal nach FIR-Filter', 'Abtastpunkt', 'Amplitude', fir\_system)

xlim([0 length(dirac\_puls)+length(dirac\_puls)/10]); %zur Zentrierung mit Eingangssignal



#### Testen für beliebige Eingangssignale mit Probe durch Faltung der Impulsantwort

Das FIR-System wird anschließend anhand zwei weiterer Zufallstestsignale getestet und das Ergebnis durch die Faltung des Eingangssignals mit der Impulsantwort überprüft.

#### 1. Zufallstestsignal

rng(1)

data3 = randn(1, n\_Symbols); %generieren des Zufallssignals

fir\_system3 = FIR(data3, 4, coef); %Ausgangssignalbestimmung durch Filterfunktion

fir\_faltung3 = faltung(data3, coef); %Ausgangssignalbestimmung durch Faltung mit Impulsantwort

figure (3)

subplot(2,1,1)

stem(data3)

stem\_properties('zufälliges Eingangssignal 1', 'Abtastpunkt', 'Amplitude', data3)

subplot(2,1,2)

hold on

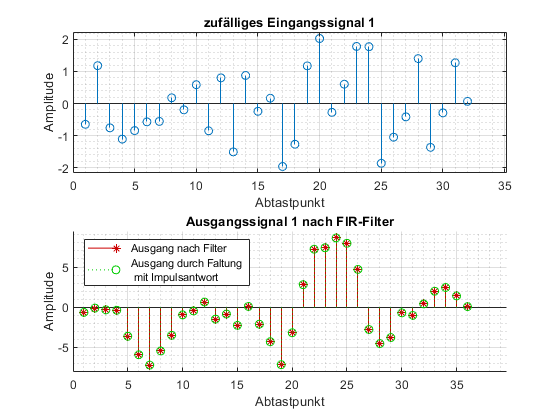
stem(fir\_system3, 'Color',[0.8 0 0],'Marker',"\*")

stem(fir\_faltung3, 'Color',[0 0.8 0],'LineStyle',":")

stem\_properties('Ausgangssignal 1 nach FIR-Filter', 'Abtastpunkt', 'Amplitude', fir\_faltung3)

hold off

legend('Ausgang nach Filter','Ausgang durch Faltung \newline mit Impulsantwort','Location','northwest')



#### 2. Zufallstestsignal

rng(13)

data4 = randn(1, n\_Symbols);

fir\_system4 = FIR(data4, 4, coef);

fir\_faltung4 = faltung(data4, coef);

figure (4)

subplot(2,1,1)

stem(data4)

stem\_properties('zufälliges Eingangssignal 2', 'Abtastpunkt', 'Amplitude', data4)

subplot(2,1,2)

hold on

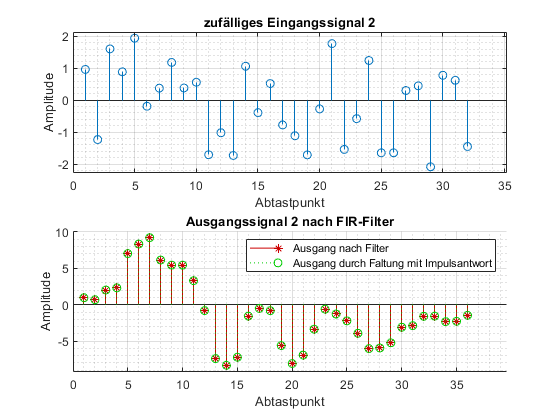
stem(fir\_system4, 'Color',[0.8 0 0],'Marker',"\*")

stem(fir\_faltung4, 'Color',[0 0.8 0],'LineStyle',":")

stem\_properties('Ausgangssignal 2 nach FIR-Filter', 'Abtastpunkt', 'Amplitude', fir\_faltung4)

hold off

legend('Ausgang nach Filter','Ausgang durch Faltung mit Impulsantwort')



# *IIR - Infinite Impulse Response*

IIR-Filter gehören wie die FIR-Filter auch zu den diskreten Systemen, unterscheiden sich aber durch einen zusätzlichen Rückkoppelzweig, wodurch sie rekursiv werden.  
Besonders deutlich lässt sich dies aus der Differenzengleichung ablesen:  
Die erste Summe entspricht dem Vorwärtszweig des FIR-Filters mit den Koeffizienten b (*coef1* in Matlab) und der zweite Summenterm beschreibt den Rückkoppelzweig des rekursiven Anteils, der durch die Koeffizienten c (*coef2* in Matlab) gewichtet wird. n bestimmt die Ordnung des Filters, wodurch sich bei beispielhaftem n=4 die Koeffizienten b0-b4 und entsprechend c0-c4 ergeben. Die Koeffizienten an nullter Stelle sind jedoch immer mit ein zu wählen.  
Im nachfolgendem Code wird der IIR-Filter als erste Direktform realisiert, wodurch zwei Register (*reg1 & 2*) benötigt werden, die jeweils in den beiden Zweigen die Elemente zur Verarbeitung laden. Das Zwischenergebnis vom Vorwärtszweig wird in *u[k]* gespeichert und dient dem Rückkoppelzweig als Eingangssignal, der daraus das Ausgangssignal in *y[k]* speichert. Mittels *circshift* werden die beiden Register weitergetaktet, um das nächste Element des Eingangsvektors einzulesen.

coef1 = [1 0 0]; %Vorwärtszweig

coef2 = [1 0.9 0.81]; %Rückkoppelzweig

iir\_system = IIR(dirac\_puls, 2, coef1, coef2);

function y = IIR(x,ord,coef1, coef2)

%IIR-Filter

% coef1 = [b0 b1 b2 b3 b4] bei n=4

% coef2 = [c0 c1 c2 c3 c4] bei n=4

n = ord;

b = coef1;

c = coef2;

len = (length(x));

u = zeros(1,len); %Initialisierung der beiden Ausgänge

y = zeros(1,len);

reg1 = zeros(1,n+1); %alle Delayelemente auf 0

reg2 = zeros(1,n+1); %alle Delayelemente auf 0

%Realisierung des IIR\_Systems als erste Direktform

%Vorwärtszweig (FIR-Teil)

for k = 1:len

reg1(1) = x(k); %Register füllen mit Eingangsvektor

u(k) = sum(b .\* reg1); %Registerwerte mit Koeffizienten multiplizieren und aufsummieren für aktuellen Ausgang

reg1 = circshift(reg1, 1); %Register um eine Stelle verschieben

end

%Rückgekoppelter Zweig

for m = 1:len

if m ==1

y(m) = u(m); %Sonderfall des ersten Elements

else

reg2(1) = y(m-1); %1. Registerstelle füllen mit rückgekoppeltem Ausgang

y(m) = u(m) - sum(reg2(1:length(c)-1) .\* c(2:end)); %"Rückkoplung"

end

reg2 = circshift(reg2, 1); %Register um eine Stelle verschieben

end

end

Nach der Aufgabenstellung soll ein IIR-System mit der Gleichung:  
implementiert werden. Die Koeffizienten sind bereits oben erkennbar und setzen sich aus   
 zusammen, wodurch sich ein System zweiter Ordnung ergibt. Zur Darstellung der Filterfunktion haben wir uns für eine Impulsantwort entschieden, bei der ein Diracimpuls (oberes Bild) als Eingangssignal genutzt wird, um im unteren Bild das Ausgangssignal des Filters abzubilden.

figure (5)

subplot(2,1,1)

stem(dirac\_puls)

stem\_properties('Eingangssignal', 'Abtastpunkt', 'Amplitude', dirac\_puls)

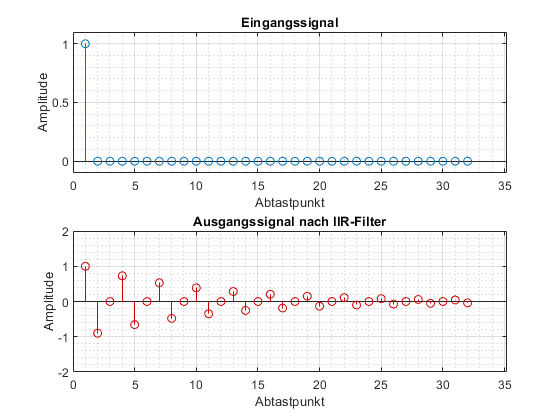
subplot(2,1,2)

stem(iir\_system, 'Color',[0.8 0 0])

stem\_properties('Ausgangssignal nach IIR-Filter', 'Abtastpunkt', 'Amplitude', fir\_system)

xlim([0 length(dirac\_puls)+(length(dirac\_puls)/10)]); %zur Zentrierung mit Eingangssignal

ylim([-2 2])



Zur Probe unserer Filterfunktion kann dasselbe Eingangssignal durch die Matlab-Funktion *filter* geschickt werden. Durch Eingabe derselben Koeffizienten ergibt sich auch eine identische Impulsantwort.

% %Zur Probe:

% iir\_system2 = filter(coef1, [1 0.9 0.81], dirac\_puls);

% figure (6)

% subplot(2,1,1)

% stem(dirac\_puls)

% stem\_properties('Eingangssignal', 'Abtastpunkt', 'Amplitude', dirac\_puls)

% subplot(2,1,2)

% stem(iir\_system2, 'Color',[0.8 0 0])

% stem\_properties('Ausgangssignal nach IIR-Filter', 'Abtastpunkt', 'Amplitude', fir\_system)

% ylim([-2 2])

Zusätzlich zur Programmierung soll die Übertragungsfunktion dieses Filter analytisch berechnet werden. Dazu muss eine Division des Ausgangs mit dem Eingang im z-Bereich erfolgen:

Ausgehend von der Differenzengleichung

und die z-Transformation der Gleichung ergibt sich:

Da folgt im Z-Bereich

Nach einer Umstellung nach Y(z) erhält man:

Und weiter umgestellt und mit erweitert ergibt sich:

Diese Gleichung muss nun in die obere Division eingesetzt werden, wodurch abschließend die Übertragungsfunktion gebildet wird:

(Man beachte, dass ist)

# *Stabilitätsprobe der Übertragungsfunktion*

In dieser Aufgabe sollten wird anhand zweier graphischer Methoden das System anhand von coef3 = [1 0.9 0.81] auf Stabilität testen.

Hierbei wird bei Methode eins im zweidimensionalen Raum und bei Methode zwei im dreidimensionalen Raum betrachtet, ob der Betrag der Polstelle der Übertragungsfunktion kleiner als eins sind und somit das System stabil ist oder größer als eins sind und das System damit unstabil wäre. Es stellte sich bei Betrachtung der Graphen heraus, dass sich beide Polstellen innerhalb des Einheitskreises befinden, die Beträge also kleiner als ein sind und das System somit stabil ist.

Dazu wurden die Nullstellen des Nennerpolynoms mittel der Matlab-Funktion *roots(coef3)* berechnet.

coef3 = [1 0.9 0.81]; %gegebene Koeffizienten

Pol = roots(coef3); %bestimmen der Nullstellen des Nennerpolynoms der Übertragungsfunktion --> Polstellen

abs(Pol) %Betrag der Polstelle in der komplexen Ebene

ans = 2×1

0.9000  
 0.9000

Kreisumfang = linspace (0, 2\*pi); %Erstellung des Einheitskreises

x = cos(Kreisumfang);

y = sin(Kreisumfang);

Die Darstellung im Einheitskreis plottet einen Kreis, der die cos-Funktion auf die sin-Funktion abträgt. Die Polstellen werden über die kartesischen Koordinaten dem Plot als *Marker* hinzugefügt.

#### 1. Methode: Einheitskreis

figure (6)

plot(x,y,'Color','r', "LineStyle",'-.'); %Einheitskreis

axis equal; %mit symmetrischen Achsen

hold on

% Sobald die Beträge < 1 und damit im Einheitkreis sind --> System ist stabil

plot(real(Pol(1)),imag(Pol(1)),'Marker','x','Color',[0 0 0.8],"MarkerSize",13,'LineStyle','none','linewidth',2)

plot(real(Pol(2)),imag(Pol(2)),'Marker','x','Color',[0 0.8 0],"MarkerSize",13,'LineStyle','none','linewidth',2)

xlabel('Real')

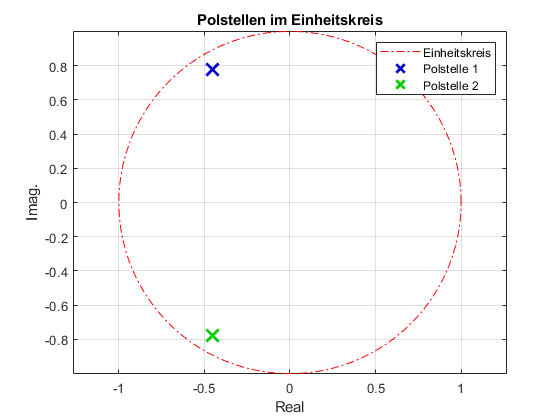
ylabel('Imag.')

grid on;

hold off;

title ('Polstellen im Einheitskreis')

legend ('Einheitskreis','Polstelle 1','Polstelle 2')



Die Abbildung im dreidimensionalem Raum verwendet wieder den Einheitskreis als grafische Darstellung der Stabilität. Zusätzlich wird die komplexe Ebene als *meshgrid* dargestellt. Auf das *meshgrid* wird der Betrag der Übertragungsfunktion als 3.Achse geplottet. Somit ergibt sich die untere Darstellung der Übertragungsfunktion als Fläche über die komplexe Ebene.

#### 2.Methode: Darstellung im 3D-Plot (meshgrid)

[x\_Real,y\_Imag] = meshgrid (-2:0.1:2); %quadratisches "Meshgrid" (Grenzen von -2 bis 2 in 0,1 Schritten) --> spannt die komplexe Ebene auf

z = x\_Real + 1j \* y\_Imag; %z soll auf der komplexen Ebene dargestellt werden

x2 = z.^2 ./ (z.^2 + 0.9\*z + 0.81); %Übertragungsfunktion (Herleitung im Protokoll)

figure (7);

mesh (x\_Real, y\_Imag, abs(x2)) %Plotten vom 3D-Raum

line(x, y , 3\*ones(size(Kreisumfang)),'linewidth',2,'Color','red','LineStyle','-.') %Einheitskreis (z-Koordinate besteht nur aus Einsen, die durch den Vorfaktor auf 3 (zur besseren Ansicht) hochgeschoben sind)

view([45 60]) %schräge Ansicht

xlim ([-2 2])

ylim ([-2 2])

zlim ([0 10]);

caxis ([0,10]); %caxis = Limit für die Colormap

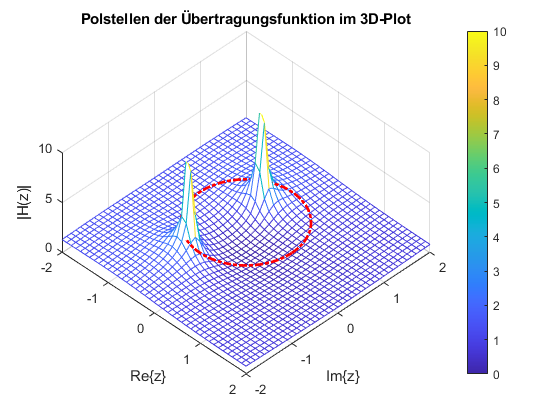
xlabel ('Re\{z\}') %(Achsenbeschriftung wie in Vorlesung)

ylabel ('Im\{z\}')

zlabel ('|H(z)|')

title ('Polstellen der Übertragungsfunktion im 3D-Plot')

colorbar



# *FIR-System aus einem IIR-System*

Das FIR-System lässt durch einfaches Ändern der Filterkoeffizienten aus der Implementierung des IIR-Systems erzeugen. So müssen die b-Koeffizienten (*coef1*) den Werten des FIR-Filters entsprechen und die c-Koeffizienten (*coef2*) müssen zu Null gesetzt werden:

coef4 = [1 2 3 2 1]; %Vorwärtszweig

coef5 = [1 0 0 0 0]; %Rückkoppelzweig

fir\_aus\_iir = IIR(dirac\_puls, 4, coef4, coef5);

Das Ergebnis ist dasselbe, wie beim anfänglichen FIR-Filter aus Kapitel 2.5.

# *Erregung mit nichtkausaler komplexer Exponentialfolge*

k = -50:1:100; %gegeben

z = 1.1; %gegeben

x= z.^k; %komplexe Exponentialfolge

y = IIR(x,2,coef1,coef2); %Filterung der Folge mit bekanntem IIR-Filter

figure(8)

subplot (2,1,1)

hold on

stem (k,x, 'Color',[0 0 0.8])

stem (k,y, 'Color',[0.8 0 0])

hold off

legend ('Eingangssignal', 'Ausgangssignal','Location','northwest')

xlabel ('Abtastpunkt')

ylabel ('Amplitude')

title ('Eingangssignal mit z\_{inf}= 1.1')

grid on

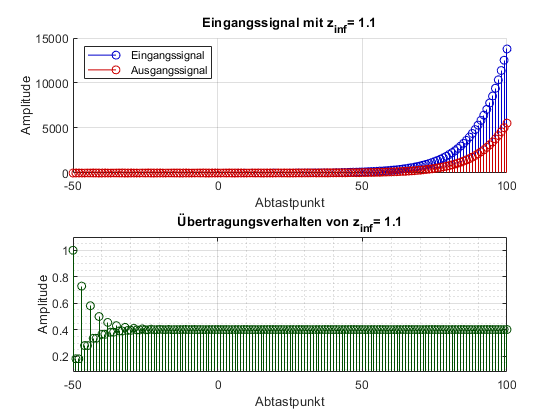
q = y ./ x; %Quotient aus Ausgangssignal zu Eingangssignal

subplot (2,1,2)

stem (k,q,'Color',[0 0.3 0])

stem\_properties('Übertragungsverhalten von z\_{inf}= 1.1', 'Abtastpunkt', 'Amplitude', q)

xlim([-50 100])



Y=polyval(coef1,z); %Auswertung des Zählerpolynoms

X=polyval(coef2,z); %Auswertung des Nennerpolynoms

H\_z = Y/X

H\_z = 0.4020

z1= 1.1 \* exp(1j\*pi/7);

losung0 = ((1.1 \* exp(1j\*pi/7))^2)/(((1.1 \* exp(1j\*pi/7))^2)+(0.9\*(1.1 \* exp(1j\*pi/7)))+0.81);

losung1 = ((z1)^2)/(((z1)^2)+(0.9\*(z1))+0.81);

Y1=polyval(coef1,z1); %Auswertung des Zählerpolynoms

X1=polyval(coef2,z1); %Auswertung des Nennerpolynoms

H\_z1 = Y1/X1

H\_z1 = 0.3980 + 0.1623i

abs(H\_z1)

ans = 0.4298

x= z1.^k; %komplexe Exponentialfolge

y = IIR(x,2,coef1,coef2); %Filterung der Folge mit bekanntem IIR-Filter

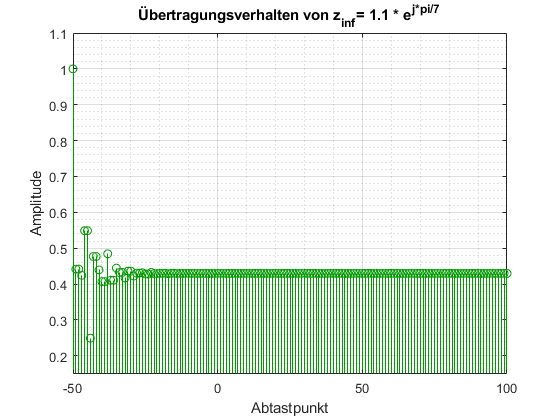
q = y ./ x; %Quotient aus Ausgangssignal zu Eingangssignal

figure(9)

stem (k,abs(q),'Color',[0 0.6 0])

stem\_properties('Übertragungsverhalten von z\_{inf}= 1.1 \* e^{j\*pi/7}', 'Abtastpunkt', 'Amplitude', abs(q))

xlim([-50 100])



z2= 1.2 \* exp(1j\*6\*pi/5);

losung2 = ((1.2 \* exp(1j\*6\*pi/5))^2)/(((1.2 \* exp(1j\*6\*pi/5))^2)+(0.9\*(1.2 \* exp(1j\*6\*pi/5)))+0.81);

losung3 = ((z2)^2)/(((z2)^2)+(0.9\*(z2))+0.81);

Y2=polyval(coef1,z2); %Auswertung des Zählerpolynoms

X2=polyval(coef2,z2); %Auswertung des Nennerpolynoms

H\_z2 = Y2/X2

H\_z2 = 1.7162 + 0.2849i

abs(H\_z1)

ans = 0.4298

x= z2.^k; %komplexe Exponentialfolge

y = IIR(x,2,coef1,coef2); %Filterung der Folge mit bekanntem IIR-Filter

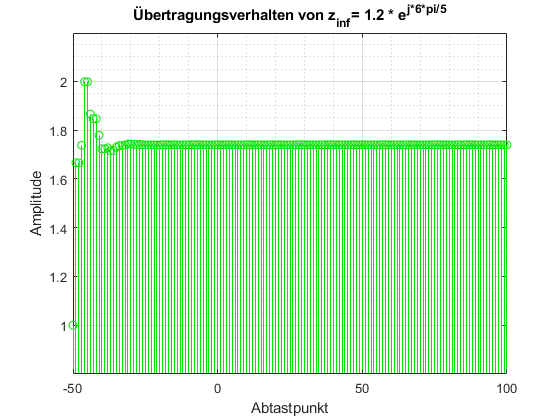
q = y ./ x; %Quotient aus Ausgangssignal zu Eingangssignal

figure(10)

stem (k,abs(q),'Color',[0 0.9 0])

stem\_properties('Übertragungsverhalten von z\_{inf}= 1.2 \* e^{j\*6\*pi/5}', 'Abtastpunkt', 'Amplitude', abs(q))

xlim([-50 100])



**Plot**

figure (11);

Kreisumfang = linspace (0, 2\*pi); %Erstellung des Einheitskreises

x = cos(Kreisumfang);

y = sin(Kreisumfang);

mesh (x\_Real, y\_Imag, abs(x2)) %Plotten vom 3D-Raum

line(x, y , 1.5\*ones(size(Kreisumfang)),'linewidth',2,'Color','red','LineStyle','-.')

line(real(z), imag(z) , abs(H\_z),'Marker','\*','Color',[0.8 0 0],"MarkerSize",10,'linewidth',2,'LineStyle','none')

line(real(z1), imag(z1) , abs(H\_z1),'Marker','\*','Color',[0 0.8 0],"MarkerSize",10,'linewidth',2,'LineStyle','none')

line(real(z2), imag(z2) , abs(H\_z2),'Marker','\*','Color','m',"MarkerSize",10,'linewidth',2,'LineStyle','none')

view([45 60]) %schräge Ansicht

xlim ([-2 2])

ylim ([-2 2])

zlim ([0 10]);

caxis ([0,10]); %caxis = Limit für die Colormap

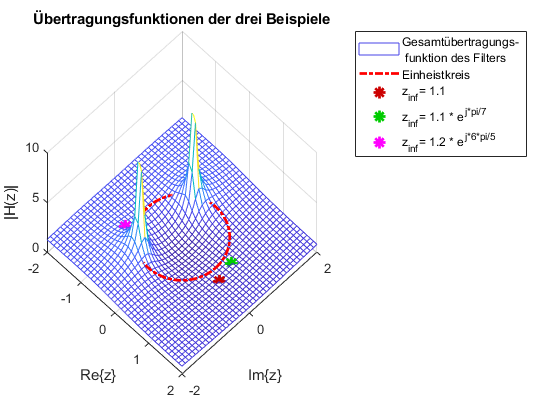
xlabel ('Re\{z\}') %(Achsenbeschriftung wie in Vorlesung)

ylabel ('Im\{z\}')

zlabel ('|H(z)|')

title ('Übertragungsfunktionen der drei Beispiele')

legend('Gesamtübertragungs- \newline funktion des Filters', 'Einheistkreis', 'z\_{inf}= 1.1', 'z\_{inf}= 1.1 \* e^{j\*pi/7}', 'z\_{inf}= 1.2 \* e^{j\*6\*pi/5}')



# Zusammenfassung & Fazit

# Eigenständigkeitserklärung

Hiermit erkläre ich, Oskar Creutzer und Lennert Hahn, dass es sich bei der von mir eingereichten schriftlichen wissenschaftliche Arbeit mit dem Titel *„Digitale Signalverarbeitung MATLAB Rechenübung“* um eine von mir erstmalig, selbstständig und ohne fremde Hilfe verfasste Arbeit handelt.

Ich erkläre ausdrücklich, dass ich sämtliche in der oben genannten Arbeit verwendete fremde Quellen, auch aus dem Internet (einschließlich Tabellen, Grafiken u.Ä.) als solche kenntlich gemacht habe und die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen habe. Insbesondere bestätige ich, dass ich ausnahmslos sowohl bei wörtlich übernommenen Aussagen bzw. unverändert übernommenen Tabellen, Grafiken u.Ä. (Zitaten) als auch bei in eigenen Worten wiedergegebenen Aussagen bzw. von mir abgewandelten Tabellen, Grafiken u.Ä. anderer Autorinnen und Autoren (Paraphrasen) die Quelle angegeben habe.

Berlin, 21. Januar 2021 Oskar Creutzer & Lennert Hahn