Technische Hochschule Köln

Fakultät IME - NT
Bereich Regelungstechnik
Prof. Dr.-Ing. R. Bartz

SIG: Signalverarbeitung

Teampartner (Name):	**Praktikum**	**Laborplatz:**
Fatima Al Khttabi 11145964	**Versuch 1**	ZO6-10

Name: Al Housseini	**Studiengang /-richtung** ☑ **Bachelor TIN (IE)** ☐ **Sonstige:**
Vorname: Ahmad	**Versuchs-Datum:** 7/12/2021
Matr.-Nr.: 11145964	**Abgabe-Datum:** 11/12/2021

Task-Nr: T17

Signalverarbeitung mit MATLAB

VORTESTAT: (erfolgreiche Versuchsdurchführung)	
☐ **Bitte den Versuchsbericht erneut vorlegen.**	
ENDTESTAT: (Vortestat & anerkannter Versuchsbericht)	

Anmerkung: Erteilung des Endtestates bedeutet nicht, dass der Versuchsbericht fehlerfrei ist.

1 Einleitung und Vorbereitung

1.1 Einleitung

Dieser Versuch dient dazu, die Funktionalität von DT-Systemen mit Hilfe von MATLAB-Programmen zu implementieren.
Diese Versuchsanleitung stellt dazu in Abschnitt 2 die wesentlichen für die Implementierung wichtigen MATLAB-Eigenschaften dar. Lesen Sie diesen Abschnitt bitte sehr gründlich durch; er stellt in knapper Form alle wichtigen Zusammenhänge dar.
In Abschnitt 3 bis 5 werden dann die Teilschritte dieses Versuchs beschrieben.

Da Ihnen im Praktikum PC-basierte Tools zur Verfügung stehen (u.a. Word, Excel, MATLAB) sollten Sie **einen eigenen USB-Stick** mitbringen. Auf diesen können Sie Ihre Ergebnisse speichern und ggf. zuhause weiter bearbeiten.

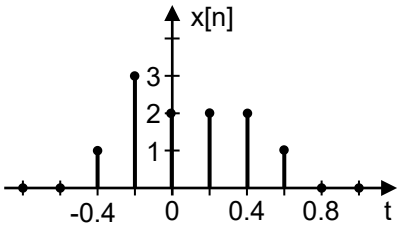
Es hat sich als zweckmäßig erwiesen, die **Unterlagen** (Vorlesung, Übung, Lehrbuch, Hilfsblätter, ... soweit vorhanden) zu "SIG: Signalverarbeitung" **mitzubringen**, da das notwendige Wissen in den meisten Fällen sonst nicht verfügbar ist.

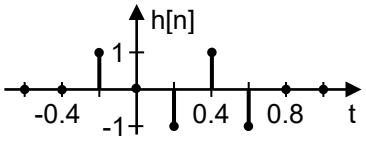
Jedem Team wird am Versuchstermin eine konkrete Aufgabenstellung ausgehändigt (sog. **Task-Definition**), die die Details der zu implementierenden Funktionalität definiert. Diese Task-Definition wird an verschiedenen Stellen dieser Versuchsanleitung referenziert; sie trägt eine Task-Nummer.

Tragen Sie diese Task-Nummer auf der Titelseite ein!

1.2 Vorbereitung

Führen Sie die Faltung y[n]=x[n]*h[n] der beiden folgenden Signale durch, und füllen Sie die leeren Felder in der Tabelle in Abschnitt 4.1 (im dort mit '*' markierten Feld muss der linke Rand von y[n] eingetragen sein).





Diese Vorbereitung ist **vor dem Versuchstermin** zu bearbeiten und zum Versuchstermin mitzubringen. Die Durchführung der Faltung muss **handschriftlich** erfolgt sein, und Sie müssen auf Nachfrage in der Lage sein, jeden einzelnen Schritt der Faltung zu erläutern.
Die handschriftliche Durchführung der Faltung muss mit dem Versuchsbericht abgegeben werden.
Fehlende Vorbereitung oder mangelnde Kenntnis der Faltung zeitdiskreter Signale kann zum Ausschluss von der Versuchsteilnahme führen!

2 (Sehr kurze) Einführung in wichtige Aspekte von MATLAB

MATLAB (**Mat**rix **lab**oratory) ist eine professionelle Entwicklungsumgebung für wissenschaftliche Modellbildung, Analyse und Design. Es besteht aus dem Basispaket MATLAB/Simulink und einer großen Zahl an Zusatzkomponenten (Toolboxen) für eine Vielzahl spezieller Aufgabenstellungen. Für die Durchführung dieses Versuchs ist nur das Basispaket erforderlich.

Zu MATLAB gibt es eine große Anzahl frei verfügbarer Beschreibungen und Tutorials im Internet, die Sie gerne für die Vorbereitung hinzuziehen können; eine (nicht repräsentative) Auswahl ist:

- www.informatik.uni-ulm.de/ni/Lehre/WS07/DMM/MatlabIntro.pdf
- www.mathworks.de/academia/student_center/tutorials/launchpad.html
  Das Original; behandelt (natürlich) die neueste Version von MATLAB; im Labor ist jedoch ein früheres Release installiert; hier können sich also kleine Unterschiede im Aussehen und Verhalten ergeben.

Die folgenden Erläuterungen fassen die wichtigsten Aspekte kurz zusammen; mit den hier beschriebenen Aspekten von MATLAB sollten Sie in der Lage sein die Praktikums-Aufgaben erfolgreich zu implementieren. Im Labor steht Ihnen dann die umfangreiche Hilfe-Funktion aus MATLAB heraus zur Verfügung, in der Sie Details nachschlagen können.
MATLAB speichert und lädt Dateien immer per default aus dem aktuellen Verzeichnis, das im oberen Bereich der GUI einstellbar ist. Bitte achten Sie darauf, dass dies **immer 'D:\User\Matlab'** ist!

MATLAB ist ein kommando-orientiertes System. In einem 'Command-Window' kann der Benutzer MATLAB-eigene oder benutzer-erstellte Kommandos eingeben; MATLAB führt das Kommando aus und gibt bei Bedarf das Ergebnis im Command-Window aus. Wird dem Kommando ein '`;`' nachgestellt, wird MATLAB das Kommando ausführen, das Ergebnis aber nicht ausgeben.
Eine Sequenz von Kommandos kann in einen Text-File gespeichert werden; der Filename sollte mit einem Buchstaben beginnen, nur aus Buchstaben, Ziffern oder '_' bestehen und die Extension '.m' tragen; sein Name kann dann als neues Kommando verwendet werden. Wird dieser Name (ohne die Extension) im Command-Window eingegeben, werden die im Text-File gespeicherten Kommandos nacheinander abgearbeitet. Eine solche Kommando-Sequenz wird als MATLAB-Programm betrachtet.
Neben der rein sequentiellen Abarbeitung bietet MATLAB die auch in anderen Programmiersprachen üblichen Schleifen-Konstrukte `for..`, `while..` sowie Verzweigungsmöglichkeiten über `if..else..` und `switch..case...`
In MATLAB können Variablen zur Zwischenspeicherung von Werten verwendet werden; sie müssen nicht deklariert werden und werden von MATLAB dann standardmäßig als double-precision floating-point Variable angelegt.
Variablen werden von MATLAB grundsätzlich als Matrix interpretiert - im einfachsten Fall also als 1x1 Matrix.
Namen sind in MATLAB immer case-sensitiv; "n" und "N" sind Namen zweier verschiedener Variablen.
Variablen werden innerhalb von MATLAB in einem 'Workspace' gespeichert. Dieser Workspace ist von der Entwicklungs-Umgebung aus erreichbar (falls nicht sichtbar: HOME→Layout→SHOW→Workspace). Für einfache Variablen wird der aktuelle Wert dort angezeigt; Arrays und Strukturen werden nach einem Doppelklick auf den Variablennamen in einem separaten Bereich angezeigt.
Array-Indizes starten immer mit 1 (Achtung, dies ist also anders als bei C oder Java !!!). Arrays werden durch runde Klammern indiziert: x(3) adressiert das dritte Element im Array x.
Variablen können in eine binäre Datei gespeichert und wieder aus ihr gelesen werden (Extension '.mat').

Hinweise zur MATLAB-Programmiersprache:
MATLAB kennt arithmetische Operatoren '`+`' '`-`' '`*`' '`/`' '`^`'; sie führen die bekannten math. Operationen durch; wenn die Operanden Arrays sind, werden die zugehörigen Matrixoperationen durchgeführt. Wenn zwei Arrays elementweise multipliziert werden sollen, muss der Operator '`.*`' verwendet werden; ebenso bei elementweiser Division ('`./`') und Potenzierung ('`.^`'); beide Operanden müssen dabei jeweils Arrays gleicher Größe sein, und das Ergebnis ist dann auch ein Array derselben Größe.
MATLAB kennt Vergleichsoperatoren: '`>`' '`<`' '`==`' ...; sie liefern einen logischen Wert (true (1), false (0)).
MATLAB kennt logische Operatoren '`&&`' '`||`' (z.B. '`n>1&&y<7`'); sie liefern wieder einen logischen Wert.

Die folgenden von MATLAB bereitgestellten Syntax-Elemente und Funktionen sind wichtig:

`%`	markiert den Beginn eines Kommentars; er endet mit dem Zeilenende.
`arr=a:b:c`	erzeugt ein Array mit dem Namen 'arr' und einer Reihe von Werten entsprechend der drei Parameter a (Startwert), b (Inkrement), und c (obere Grenze). (Beispiel: `n=-3:2:8` erzeugt das Array n mit den 6 Werten [-3 -1 1 3 5 7])
`arr=a:b`	erzeugt ein Array mit dem Namen 'arr' und einer Reihe von Werten entsprechend der Parameter a (Startwert) und b (obere Grenze). (Beispiel: `n=-3:2` erzeugt das Array n mit den 6 Werten [-3 -2 -1 0 1 2])
`arr=[a b c ..]`	erzeugt ein Array mit dem Namen 'arr' und den (durch Leerzeichen getrennten) angegebenen Werten.
`length(n)`	liefert die Länge des Arrays n; hat n mehrere Spalten/Zeilen wird die längste Ausdehnung geliefert.
`arr2=arr(n:m)`	erzeugt ein Array mit dem Namen 'arr2' und weist ihm die Elemente arr(n)…arr(m) zu.
`[z,fs,bits]=wavread('file.wav')`	liest die Datei mit dem Namen <file.wav>, sofern es eine .wav Datei ist und liefert folgende Informationen: bits: die Auflösung der Samples (Anzahl Bit; typ. 8 oder 16) fs: die Abtastfrequenz f_s (=1/T) z: die Audio-Samples; bei Stereo-Aufnahmen stecken darin beide Kanäle; der erste Kanal kann durch die Funktion `x=z(:,1)'` in eine Array-Variable x extrahiert werden.
`pl=audioplayer(x,fs); playblocking(pl);`	spielt die samples im Array x als Sound auf einem PC-Lautsprecher ab. Die Werte von x sollten im Bereich [-1,1] liegen; fs gibt die Abtastfrequenz an (default:11025Hz)
`save('fn','x')`	speichert die im Workspace liegende Variable mit dem Namen 'x' in eine Datei mit dem Namen 'fn.mat'.
`load fn x`	lädt die Variable mit dem Namen 'x' aus einer Datei mit dem Namen 'fn.mat' in den Workspace.
`xmin=min(x)`	liefert den kleinsten der Werte im Array x an die Variable 'xmin' zurück.
`xmax=max(x)`	liefert den größten der Werte im Array x an die Variable 'xmax' zurück.
`x=cos(t)`	berechnet cos von t und liefert das Ergebnis an x zurück. Wenn t ein Array ist, erfolgt die Berechnung für jedes Array-Element, und x wird zu einem Array gleicher Größe.
`x=exp(t)`	berechnet e^t und liefert das Ergebnis an x zurück. Wenn t ein Array ist, erfolgt die Berechnung für jedes Array-Element, und x wird zu einem Array gleicher Größe.
`pi`	die Zahl π.
`j`	die imaginäre Zahl ($j^2 = -1$).Kann auch als '`i`', '`1j`' oder '`1i`' dargestellt werden.
`tic`	startet einen Zeitgeber.
`toc`	stoppt den mit tic gestarteten Zeitgeber und gibt die Zeitdifferenz zurück.
`clear`	löscht alle Variablen im Workspace; sollte gelegentlich aufgerufen werden.
`figure`	erzeugt ein neues Diagramm-Fenster. Es kann über File→SaveAs als .jpg oder .png Datei gespeichert und dann in den Versuchsbericht integriert werden.
`subplot(a,b,c)`	erzeugt einen Zeichenbereich in einem Diagramm-Fenster. Dazu wird das Diagramm-Fenster in a·b Teile aufgeteilt (angeordnet in a Zeilen und b Spalten). Das Element c ist das ab jetzt aktive Subplot (dabei wird zeilenweise von links oben nach rechts unten gezählt).
`stem(t,y)`	zeichnet die Array-Werte y(..) (vertikale Achse) über den Array-Werten t(..) (horizontale Achse) als DT-Signal. Beide Arrays müssen dieselbe Länge haben. Gezeichnet wird in das zuletzt aktive Diagramm-Fenster und (falls mehrere Subplots dort angelegt sind) in das zuletzt aktive Subplot. Eine Vielzahl weiterer Übergabeparameter erlaubt eine Anpassung der Darstellung (optional; hier nicht weiter erläutert).
`axis([a,b,c,d])`	(optional) skaliert das aktuelle 2D-Koordinatensystem auf die Darstellungsbereiche [a,b] in horizontaler und [c,d] in vertikaler Richtung. Dieses Koordinatensystem wird in dem zuletzt aktiven Diagramm-Fenster und (falls mehrere Subplots dort angelegt sind) in dem zuletzt aktiven Subplot eingerichtet.
`grid on`	schaltet ein Gitternetz ein.
`grid off`	schaltet das Gitternetz aus.

`hold on` sorgt dafür, dass bei nachfolgenden Zeichenbefehlen (`stem()`) die bisherigen Signale in der Grafik nicht ersetzt werden sondern erhalten bleiben.
`hold off` sorgt dafür, dass bei nachfolgenden Zeichenbefehlen (`stem()`) die bisherigen Signale in der Grafik durch das jeweils neue ersetzt werden.

Wichtige Schleifenkonstrukte von MATLAB sind:

```
if (a)      a bestimmt, ob block1 durchgeführt wird (a==true; a==1) oder nicht.
   block1   ein oder mehrere MATLAB Befehle.
else        (optional) zeigt den Beginn des Bereichs an, der durchgeführt wird wenn a==false.
   block2   (optional) ein oder mehrere MATLAB-Befehle.
end         zeigt das Ende des if-Konstrukts an.

for i=a:b:c i wird initial auf den Wert a gesetzt; block wird dann ausgeführt solange i≤c.
   block    ein oder mehrere MATLAB Befehle; bei jedem Durchlauf wird i um b inkrementiert.
end         zeigt das Ende des for-Konstrukts an.

while expr  block wird ausgeführt solange expr true ist; expr wird vor jedem Durchlauf geprüft.
   block    ein oder mehrere MATLAB Befehle.
end         zeigt das Ende des while-Konstrukts an.
```

Weitere ggf. interessante Funktionen sind:

```
xlabel()              erlaubt die Beschriftung der horizontalen Achse
ylabel()              erlaubt die Beschriftung der vertikalen Achse
set(gca,'xtick',..)   ersetzt die horizontale default-Achsenskalierung durch eine eigene Skalierung
set(gca,'ytick',..)   ersetzt die vertikale default-Achsenskalierung durch eine eigene Skalierung
```

Sie werden hier nicht näher beschrieben, können aber in der MATLAB Hilfefunktion nachgeschlagen werden

3 Teil 1: Ton- und Klanggenerator

Töne sind periodische Signale mit Frequenzen im hörbaren Bereich (~ 20Hz .. 20kHz); Klänge sind Überlagerungen aus solchen Tönen.
Mit Hilfe von MATLAB-Programmen sollen in diesem Versuchsteil zeitdiskrete Ton- und Klangsignale erzeugt werden. Durch die Ausgabe auf einem Lautsprecher wird aus einem zeitdiskreten Signal ein zeitkontinuierliches Schall-Signal.

Ein einzelner Ton lässt sich als sinusförmiges zeitdiskretes Signal darstellen: $x[nT]=A\cdot\cos(2\pi f_0 nT+\theta)$; darin ist A die Amplitude, f_0 die Frequenz und θ die Phase.
Ein Klang entsteht durch Überlagerung mehrerer sinusförmiger Signale mit Frequenzen, die jeweils ein ganzzahliges Vielfaches einer gemeinsamen Grundfrequenz sind (Harmonische).
Ein Mischton (auch Tongemisch) ist eine Überlagerung mehrerer sinusförmiger Signale, deren Frequenzen nicht ein ganzzahliges Vielfaches einer Grundfrequenz sein müssen.

3.1 Einzel-Ton

a) Schreiben Sie ein MATLAB-Programm das einen einzelnen Ton erzeugt;
   Parameter: Amplitude: 0.3; Frequenz 400Hz; Phase:0; Dauer: 3s; Abtastfrequenz f_s: 22050Hz
b) Stellen Sie einen Ausschnitt der Länge 10ms des Signals über der **Zeitachse** (t) grafisch dar (`stem()`).
c) Spielen Sie das Signal auf dem Lautsprecher ab.

3.2 Klang

a) Schreiben Sie ein MATLAB-Programm das einen Klang erzeugt. Die Spezifikation des zu erzeugenden Klangs finden Sie auf Ihrer speziellen Task-Definition.
b) Stellen Sie einen Ausschnitt der Länge T_{graph} (siehe Task-Definition) des Signals über der **Zeitachse** (t) grafisch dar (`stem()`).
c) Spielen Sie das Signal auf dem Lautsprecher ab.

4 Teil 2: Faltung kurzer Signale

In diesem Versuchsteil soll ein MATLAB-Programm erstellt und angewendet werden, das die zeitdiskrete Faltung implementiert. Es kann davon ausgegangen werden, dass die zu faltenden Signale eine endliche Breite haben. Linker und rechter Rand können jedoch auch negativ werden.

Folgende Notation soll eingehalten werden:
x[n] und h[n] seien die zu faltenden Signale, y[n] das Ergebnissignal. x[n] kann als Eingangssignal eines DT-Systems betrachtet werden und h[n] als Einheitspuls-Antwort; dann ist y[n] das Ausgangssignal des Systems; es berechnet sich aus:

$$y[n] = \sum_{m=-\infty}^{\infty} x[m] \cdot h[n-m] = \sum_{m=-\infty}^{\infty} h[m] \cdot x[n-m]$$

Die Signale sollen jeweils in einem MATLAB-Array gespeichert werden; Variablen-Namen: `x`, `h`, `y`.

Die Ränder der Signale sollen in folgenden Variablen abgelegt werden:

linker Rand von x: `NXL`	rechter Rand von x: `NXR`	Breite von x: `IX`
linker Rand von h: `NHL`	rechter Rand von h: `NHR`	Breite von h: `IH`
linker Rand von y: `NYL`	rechter Rand von y: `NYR`	Breite von y: `IY`

Der Abtastabstand $T=1/f_s$ soll in der MATLAB-Variable `T` abgelegt werden.

Jedes Signal soll über der Zeitachse (t) und über der Indexachse (n) dargestellt werden können. Da MATLAB nur mit positiven Array-Indizes arbeiten kann, muss zwischen MATLAB Array-Index und Indexachse (n) unterschieden werden. Dies soll wie folgt umgesetzt werden:

- Zur vollständigen Beschreibung des Signals x[n] werden drei MATLAB-Arrays verwendet:

`x:`	`x(1)`	`x(2)`	`...`	`x(ix)`	`...`	`x(IX)`	enthält die Samples von x[n]
`nx:`	`nx(1)`	`nx(2)`	`...`	`nx(ix)`	`...`	`nx(IX)`	enthält die Werte der Indexachse für x[n]
`tx:`	`tx(1)`	`tx(2)`	`...`	`tx(ix)`	`...`	`tx(IX)`	enthält die Werte der Zeitachse für x[n]

- Zur vollständigen Beschreibung des Signals h[n] werden drei MATLAB-Arrays verwendet:

`h:`	`h(1)`	`h(2)`	`...`	`h(ih)`	`...`	`h(IH)`	enthält die Samples von h[n]
`nh:`	`nh(1)`	`nh(2)`	`...`	`nh(ih)`	`...`	`nh(IH)`	enthält die Werte der Indexachse für h[n]
`th:`	`th(1)`	`th(2)`	`...`	`th(ih)`	`...`	`th(IH)`	enthält die Werte der Zeitachse für h[n]

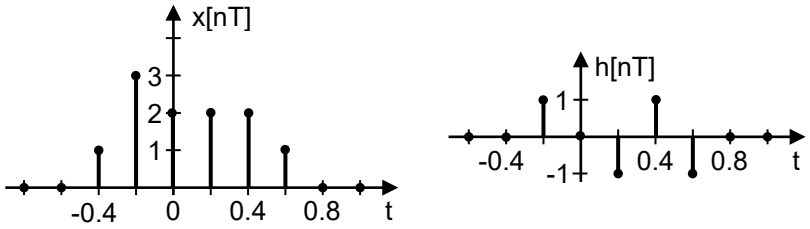
- Zur vollständigen Beschreibung des Signals y[n] werden drei MATLAB-Arrays verwendet:

`y:`	`y(1)`	`y(2)`	`...`	`y(iy)`	`...`	`y(IY)`	enthält die Samples von y[n]
`ny:`	`ny(1)`	`ny(2)`	`...`	`ny(iy)`	`...`	`ny(IY)`	enthält die Werte der Indexachse für y[n]
`ty:`	`ty(1)`	`ty(2)`	`...`	`ty(iy)`	`...`	`ty(IY)`	enthält die Werte der Zeitachse für y[n]

Die Arrays sind gerade so groß, dass sie die Signalausschnitte vom linken bis zum rechten Rand (einschließlich) speichern können; es ist jeweils `nx(1)=NXL`, `nh(1)=NHL` und `ny(1)=NYL`.

4.1 Implementierung und Test

a) Schreiben Sie ein MATLAB-Programm das auf dieser Basis die Faltung implementiert; das Hilfsblatt im Anhang kann dazu ggf. ebenfalls nützlich sein.
b) Verifizieren Sie das Programm, indem Sie es auf folgende Signale anwenden (s. 1.2 Vorbereitung):



Es sollten sich folgende Array-Inhalte ergeben haben
(füllen Sie die leeren Felder mit den Ergebnissen aus Ihrer Vorbereitung; Abschnitt 1.2):

x:	1	3	2	2	2	1
nx:			0			
tx:	-0.4		0.0			

h:	1	0	-1	1	-1
nh:		0			
th:		0.0			0.6

y:										
ny:	*									
ty:										

*: linker Rand von y[nT]

c) Stellen Sie die Signale x, h und y übereinander in jeweils einzelnen Subplots innerhalb eines Diagramm-Fensters grafisch dar (`stem(), subplot()`); je ein separates Diagramm über der **Zeitachse** (t) und der **Indexachse** (n) ist erforderlich.
(Es ergeben sich also 2 Diagramm-Fenster mit jeweils drei übereinander angeordneten Subplots.)

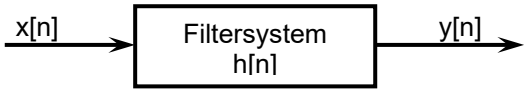
4.2 Faltung spezieller Signale

a) Passen Sie das Programm auf die in Ihrer Task-Definition gegebenen Signale x[nT] und h[nT] an und erzeugen Sie daraus das Ergebnissignal y[nT].
b) Stellen Sie die Signale x, h und y übereinander in jeweils einzelnen Subplots innerhalb eines Diagramm-Fensters grafisch dar (`stem(), subplot()`); je ein separates Diagramm über der **Zeitachse** (t) und der **Indexachse** (n) ist erforderlich.
(Es ergeben sich also 2 Diagramm-Fenster mit jeweils drei übereinander angeordneten Subplots.)

5 Teil 3: Filterung eines .wav-Signals

5.1 Audio-Filterung

In diesem Versuchsteil soll das zur Faltung erstellte MATLAB-Programm angewendet werden, um ein in einer .wav-Datei enthaltenes Audio-Signal zu filtern. Das in der .wav-Datei enthaltene Signal x[n] ist somit Eingangssignal eines DT-Filtersystems. Die Filterung soll durch eine Faltung des Eingangssignals mit der Einheitspuls-Antwort h[n] des Systems erfolgen.



a) Lesen Sie die Ihnen von Ihrem Betreuer zur Verfügung gestellte .wav-Datei in MATLAB ein und spielen Sie den Audio-Stream des Eingangssignals auf dem Lautsprecher ab.
   Tragen Sie den Namen der .wav-Datei in das Blatt mit Ihrer Task-Definition ein.
b) Falten Sie das Signal mit der Einheitspuls-Antwort h[n] des Systems, die in Ihrer Task-Definition spezifiziert ist. Passen Sie dazu Ihr Programm entsprechend an.
c) Stellen Sie die Signale x, h und y übereinander in jeweils einzelnen Subplots innerhalb eines Diagramm-Fensters über der **Zeitachse** (t) grafisch dar (`stem(), subplot()`):
   - als Gesamt-Signal, und
   - als Ausschnitt der ersten 500 Samples.
   (Es ergeben sich also 2 Diagramm-Fenster mit jeweils drei übereinander angeordneten Subplots.)
d) Spielen Sie den Audio-Stream des Eingangs- und Ausgangssignals im Wechsel hintereinander auf dem Lautsprecher ab.

6 Versuchs-Bericht

Jeder Teilnehmer hat einen eigenen Versuchs-Bericht in elektronischer Form per email an den Betreuer abzugeben. Zippen Sie alle unten genannten Dateien in **12345678.zip** (darin steht 12345678 für Ihre Matr.-Nr.) - achten Sie darauf dass es tatsächlich ein **zip** Format ist (nicht Formate wie rar, 7z, …).

Zum Versuchs-Bericht gehören (mit den hier angegebenen Dateinamen und -typen!):
1. das Deckblatt dieser **Versuchsanleitung** (vollständig ausgefüllt, gescannt) [v1_Deck.pdf]
2. Seite 7 dieser **Versuchsanleitung** (vollständig ausgefüllt, gescannt) [v1_S7.pdf]
3. Ihre **Task-Definition** (vollständig ausgefüllt, gescannt) [v1_TaskDef.pdf]

sowie die **Dokumentation Ihrer Versuchsdurchführung**:

4. Vorbereitungs-Schritt zu 1.2: eigene handschriftliche Faltung (gescannt) [v1_Faltung.pdf]
5. Programmcode zu 3.1a [v1_code_31.m]
6. Diagramm aus 3.1b (Screenshot mit gut lesbarer Auflösung) [v1_graph_31.png]
7. Programmcode zu 3.2a [v1_code_32.m]
8. Diagramm aus 3.2b (Screenshot mit gut lesbarer Auflösung) [v1_graph_32.png]
9. Programmcode zu 4.1a [v1_code_41.m]
10. 2 Diagramme aus 4.1c (Screenshots mit gut lesbarer Auflösung) [v1_graph_41.png]
11. Programmcode zu 4.2a [v1_code_42.m]
12. 2 Diagramme aus 4.2b (Screenshots mit gut lesbarer Auflösung) [v1_graph_42.png]
13. Programmcode zu 5.1b [v1_code_51.m]
14. Diagramm mit Gesamtsignalen aus 5.1c (Screenshot mit gut lesbarer Auflösung) [v1_graph_51.png]
15. Diagramm mit Ausschnitt von 500 Samples aus 5.1c (Screenshot mit gut lesbarer Auflösung) [v1_graph_51part.png]

Bitte prüfen Sie unbedingt vor Abgabe, ob sich im zip-File 15 Dateien mit den gegebenen Namen befinden!

7 Hinweise

Denken Sie daran, Ihre Ergebnisse stets auf **einem eigenen USB-Stick** zu sichern. Es kann nicht davon ausgegangen werden, dass die zuletzt von Ihnen erarbeiteten Ergebnisse nach Ende Ihres Labortermins noch auf dem Laborrechner vorliegen.

Hilfsblatt

Hinweise zur korrekten Zuordnung und Verarbeitung der verschiedenen Indizes bei der numerischen Implementierung der zeitdiskreten Faltung mit MATLAB

Die Faltung ist nur dann vollständig numerisch implementierbar wenn beide beteiligten Signale eine endliche Breite aufweisen. Die linken (rechten) Ränder der Signale x[n], h[n], y[n] seien mit NXL, NHL, NYL (NXR, NHR, NYR) bezeichnet.
Wegen der Kommutativität der Faltung ist zunächst zu entscheiden welche der alternativen Faltungsdefinitionen für die Implementierung verwendet werden soll:

$$y[n] = \sum_{m=NXL}^{NXR} x[m] \cdot h[n-m] \qquad \text{oder} \qquad y[n] = \sum_{m=NHL}^{NHR} h[m] \cdot x[n-m] \qquad \text{jeweils für n=NYL...NYR}$$

Beides ist mathematisch gleichwertig. Da aber in der Praxis die Breite von h[n] deutlich kleiner als die von x[n] ist, bietet sich die rechte Form mit der dann kleineren Anzahl Summanden an.

Sei nun h[n] in einem MATLAB-Array `h`, x[n] in einem MATLAB-Array `x` abgelegt, und für das Ergebnis y[n] der Faltung soll ein MATLAB-Array `y` verwendet werden.

Folgende Zuordnung gilt dann für das Signal x (mit IX=NXR-NXL+1):

Signal-Sample:	x[NXL]	x[NXL+1]	...	x[n]	...	x[NXR]
Array `x(ix)`:	`x(1)`	`x(2)`	...	`x(n-NXL+1)`	...	`x(IX)`

Folgende Zuordnung gilt dann für das Signal h (mit IH=NHR-NHL+1):

Signal-Sample:	h[NHL]	h[NHL+1]	...	h[n]	...	h[NHR]
Array `h(ih)`:	`h(1)`	`h(2)`	...	`h(n-NHL+1)`	...	`h(IH)`

Folgende Zuordnung gilt dann für das Signal y (mit IY=NYR-NYL+1):

Signal-Sample:	y[NYL]	y[NYL+1]	...	y[n]	...	y[NYR]
Array `y(iy)`:	`y(1)`	`y(2)`	...	`y(n-NYL+1)`	...	`y(IY)`

(1)

Nun ist in der Faltungsdefinition nicht h[nT] sondern h[mT] verwendet, was aber die Zuordnung nur unerheblich verändert:

Signal-Sample:	h[NHL]	h[NHL+1]	...	h[m]	...	h[NHR]
Array `h(ih)`:	`h(1)`	`h(2)`	...	`h(m-NHL+1)`	...	`h(IH)`

(2)

Etwas komplizierter wird die Betrachtung von x[n-m]; allerdings ist es nicht schwer, sich klarzumachen dass der oben grau hinterlegte Eintrag sich konsequenterweise wie folgt ändert:

x[n-m]
`x(n-m-NXL+1)`

(3)

Damit gilt:

1. Soll der Wert y[n] in das MATLAB-Array `y` eingetragen werden, muss das Array-Element `y(n-NYL+1)` angesprochen werden; die Beziehung zwischen n und iy ist also:
   iy=n-NYL+1 ⇔ n=NYL+iy-1
2. Soll der Wert h[m] aus dem MATLAB-Array `h` ausgelesen werden, muss das Array-Element `h(m-NHL+1)` angesprochen werden; die Beziehung zwischen m und ih ist also:
   ih=m-NHL+1 ⇔ m=NHL+ih-1
3. Soll der Wert x[n-m] aus dem MATLAB-Array `x` ausgelesen werden, muss das Array-Element `x(n-m-NXL+1)` angesprochen werden; die Beziehung zwischen n, m und ix ist also:
   ix=n-m-NXL+1. Mit Hilfe von 1. und 2. lässt sich nun eine Beziehung zwischen ix, iy und ih herstellen.

Die Summe muss nun nacheinander für jedes y[n] mit n∈NYL..NYR (entspricht iy∈1..IY) gebildet werden.
Die Summe selber benötigt Werte von h[m] im Bereich m∈NHL..NHR (entspricht ih∈1..IH).
Bei dem Zugriff auf die Werte x[n-m] führt nicht jede Kombination von n und m zu einem zulässigen (d.h. tatsächlich vorhandenen) Wert von `x`; nur n-m∈NXL..NXR ist erlaubt (entspricht ix∈1..IX); dies muss separat abgefragt werden. Nicht im Array vorhandene Werte können wegen der Voraussetzung endlicher Breite als 0 angenommen werden.
Alternativ lassen sich auch die Summengrenzen in Abhängigkeit von iy einschränken.