Lab 2: Matlab, Simulink, and a Communications Example

1 Introduction

Linear systems are very often used for signal and information processing. Thus, it is essential to be able to generate both deterministic and random waveforms, apply them to the system(s) under test, and then process and display the results. Deterministic waveforms include test signals such as sinusoids, rectangular waveforms, and unit impulses, steps, and ramps, etc. Examples of random waveforms are noise, speech and music signals, and text messages. Even though Simulink has many different source blocks available, it is generally more convenient and more versatile to generate signals and waveforms in Matlab and then to run a Simulink model with these waveforms as input(s) from Matlab. The results of the simulation can then be passed back to Matlab for further interpretation, e.g., in the form of plots or sound signals. An application of this is the simulation of a communication system that uses pulse amplitude modulation (PAM) to transmit text messages over a bandlimited channel, such as a first order RC circuit. For an introduction to the basics of Matlab, see Appendix A.

1.1 Signals and Waveforms in Matlab

Continuous time (CT) waveforms such as \( x(t) = \cos(2\pi f_0 t + \theta) \) or \( x(t) = u(t) \) (where \( u(t) \) denotes the unit step function) have to be represented as discrete time (DT) sequences in a digital computer. This is typically done by sampling the CT waveforms at time instants \( t = nT_s \) which are spaced \( T_s \) seconds apart. The sampling rate or frequency \( F_s \) in Hz is then \( F_s = 1/T_s \) and the DT signal \( x_n \) that is obtained from a CT signal \( x(t) \) is

\[
x_n = x(nT_s) = x(n/F_s) \quad \text{for } n = 0, \pm 1, \pm 2, \ldots
\]

Typical sampling rates that are used for audio signals are \( F_s = 8000 \) Hz, \( F_s = 44100 \) Hz, \( F_s = 22050 \) Hz, etc. It is quite easy to see that higher sampling rates will yield better DT representations of CT signals and an interesting question is how large of an \( F_s \) is large enough. In a nutshell, \( F_s \) has to be at least twice as high as the highest frequency in the CT signal \( x(t) \), but to obtain nice looking graphs, \( F_s \) may have to be chosen on the order of ten times as much. As an example, consider the sinusoidal CT signal \( x(t) = \cos(2\pi f_0 t + \theta) \), \( t \geq 0 \), which becomes

\[
x_n = x(n/F_s) = \cos(2\pi f_0 n/F_s + \theta), \quad n = 0, 1, 2, \ldots
\]

after sampling with rate \( F_s \). In Matlab \( x_n \) can be generated as follows:
Fs = %Enter sampling rate here
f0 = 100; %Frequency f0 in Hz
theta = 30; %Phase in degrees
tlen = 0.1; %Signal duration in sec

\[
\text{tt} = \left[0:\text{round}(\text{tlen}\times\text{Fs})\right]/\text{Fs}; \quad \text{Time axis in sec}
\]

\[
\text{xt} = \cos(2\pi f0 \times \text{tt} + (\pi/180)\times\text{theta});
\]

%Sinusoidal waveform x(t),
%frequency f0, phase theta

The graphs below show what \(x(t)\) with \(f_0 = 100\) Hz and \(\theta = 30^\circ\) looks like after sampling at rate \(F_{s1} = 300\) Hz and at rate \(F_{s2} = 2000\) Hz.

The first graph clearly does not look like a sinusoid, despite the fact that \(F_{s1} = 300\) Hz is more than twice the frequency \(f_0 = 100\) Hz of the sinusoid. The problem is that the plot function in Matlab uses straight lines between sample points, whereas a soundcard, for example, uses a lowpass filter to interpolate between sample points. Thus, when plotting DT versions of CT signals in Matlab, it often helps to explicitly plot the sample points (shown by red dots in the graphs above) and to choose \(F_s\) large enough such that the straight line interpolation does not cause large errors.

The following examples show how to generate and plot DT representations of some of the most common CT signals in Matlab. How the sampling rate \(F_s\) is chosen depends on the signal duration of interest and on the highest relevant frequency component of the signal.
Example: Unit step of duration \( t_{len} \) at time \( t_{step} \). The following Matlab script can be used to generate \( u(t) \) at \( t = n/F_s \) for a given sampling rate \( F_s \).

```matlab
Fs = 100; %Sampling rate
tlen = 1; %Signal duration
tstep = 0.1; %Step time
tt = [0:round(tlen*Fs)]/Fs; %Time axis
ut = zeros(size(tt)); %Prepare \( u(t) \)
ix = find(tt>tstep); %Indexes where \( u(t)=1 \)
utt = ones(size(ix)); %Unit step signal \( u(t) \)
subplot(211) %Upper half of graph
plot(tt,ut,'-b',tt,utt,'.r') %Plot \( u(t) \) and samples \( u_n \)
ylim([-0.5,1.5]),grid %Adjust ylim, add grid
str = ['Unit Step u(t-t_0), t_0=' num2str(tstep) 'sec'];
str = [str ', F_s=' int2str(Fs) ' Hz'];
title(str)
legend('Straight line interpolation','Actual samples',4)
figure(gcf) %Show current figure
```

The strategy to generate \( ut \) is to start from an all-zero vector of the same size as the time axis \( tt \). Then the find command is used to find those indexes \( ix \) for which \( ut \) has to be 1. The graph of \( ut \) is shown in the figure below.

![Graph of Unit Step](image)

Note that for the sampling rate used here the actual step is visibly slanted. Choosing a higher value for \( F_s \) would make it more vertical.

Example: Unit ramp \( r(t) \) at \( t_{ramp} \) with duration \( t_{len} \). Here is a Matlab script that can be used:

```matlab
3
```
Fs = 2; %Sampling rate
tlen = 20; %Signal duration
tramp = 4; %Ramp time
tt = [0:round(tlen*Fs)]/Fs; %Time axis
rt = zeros(size(tt)); %Prepare r(t)
ix = find(tt>tramp); %Indexes where r(t)>0
rt(ix) = (tt(ix)-tramp).*ones(size(ix)); %Unit ramp signal r(t)

Note the use of component-wise multiplication (*.) in the last statement. The graph of r(t) at t = n/F_s is shown next.

In this case the sampling rate F_s can be chosen quite low for the purpose of making a plot since the function is very well approximated with straight line interpolation between samples.

Example: Real exponential. The CT signal

\[ x(t) = \begin{cases} 
 Ae^{-t/T_c}, & t > 0, \\
 0, & t < 0, 
\end{cases} \]

for \(-T_c \leq t \leq 3T_c\) can be generated in Matlab as follows:

Fs = 1000; %Sampling rate
A = 2; %Amplitude
Tc = 0.05; %Time constant
ta = -Tc; %Start time
tb = 3*Tc; %End time
tt = [round(Fs*ta):round(Fs*tb)]/Fs;
%Time axis
xt = zeros(size(tt)); %Prepare x(t)
ix = find(tt>0); %Indexes where x(t)>0
xt(ix) = A*exp(-tt(ix)/Tc); %Real exponential x(t)

The resulting graph is:
To read $T_c$ off of a graph like this, make use of the fact that $x(T_c) = A e^{-1} \approx 0.368 A$.

**Example: Damped sinusoid.** The CT signal

$$x(t) = \begin{cases} A e^{-t/T_c} \cos(2\pi f_0 t + \theta), & t > 0, \\ 0, & t < 0, \end{cases}$$

for $-T_c \leq t \leq 3T_c$ can be generated using the Matlab statements:

```matlab
Fs = 4000; %Sampling rate
A = 1; %Amplitude
Tc = 0.01; %Time constant
f0 = 200; %Frequency of sinusoid
theta = 0; %Phase of sinusoid
ta = -Tc; %Start time
tb = 3*Tc; %End time
tt = [round(Fs*ta):round(Fs*tb)]/Fs; %Time axis
at = zeros(size(tt)); %Prepare a(t)
ix = find(tt>0); %Indexes where a(t)>0
at(ix) = A*exp(-tt(ix)/Tc); %Envelope a(t) of signal x(t)
x = at.*cos(2*pi*f0*tt+(pi/180)*theta); %Damped sinusoid x(t)
```

A graph of $x(t)$ is shown below.
In this case the real exponential \( a(t) = A e^{-t/T_c} u(t) \) forms the envelope of the signal \( x(t) \).

1.2 Communication of Text Messages Using PAM

Consider the problem of transmitting a text message, such as

\[
\text{Is there life after Linear Systems?}
\]

over a waveform channel such as a twisted pair cable or a wireless RF (radio frequency) link. The design of a system that can accomplish this task requires the following ingredients:

- **Encoding** of the letters of the alphabet, the numbers, punctuation, etc. For example A could be encoded as 0, B as 1, C as 2, etc.
- Conversion of the encoded message into a **serial data stream**, e.g., of 0’s and 1’s in the case of a binary transmission system.
- **Modulation** by the serial data stream of a CT waveform that can be transmitted through the waveform channel.
- **Demodulation** of the received waveform at the output of the waveform channel to obtain the received serial data stream.
- **Conversion** of the received serial data stream to a sequence of character codes.
- **Decoding** of the received character codes to the received message.

**ASCII Code.** The task of encoding letters, numbers, punctuation, etc., can be accomplished using the ASCII (American Standard for Information Interchange) code which is a 7-bit code. The following table shows all \( 2^7 = 128 \) possible codes.
7-Bit ASCII (American Standard Code for Information Interchange)

<table>
<thead>
<tr>
<th></th>
<th>000...</th>
<th>001...</th>
<th>010...</th>
<th>011...</th>
<th>100...</th>
<th>101...</th>
<th>110...</th>
<th>111...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0000</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>'</td>
<td>p</td>
</tr>
<tr>
<td>.0001</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>.0010</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>.0011</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>.0100</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>.0101</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>.0110</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>.0111</td>
<td>BEL</td>
<td>ETB</td>
<td>’</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>.1000</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>.1001</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>.1010</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>.1011</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>.1100</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>.1101</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>.1110</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>~</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>.1111</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>

The first two columns consist of non-printable control characters, e.g., CR stands for carriage return and LF stands for line feed. The space character (ASCII code 0100000) is shown as SP in the table. If the binary codes are converted to decimal then

\[
A = 1000001 = 65, \quad B = 1000010 = 66, \quad C = 1000011 = 67, \quad \text{etc.}
\]

In Matlab the same decimal codes can be obtained from

```matlab
>> double('ABC') \%ASCII -> decimal conversion
ans =
65 66 67
```

**Decimal to Serial Binary Conversion.** To transmit a message over a communication channel with one input and one output, the bits of each ASCII encoded character need to be sent serially, one after another. One convention that needs to be established is whether the LSB (least significant bit) or MSB (most significant bit) of each code is sent first. Another consideration is how many bits should be sent per character. Even though the ASCII code has seven bits, it is customary to send 8 bits per character and set the MSB to zero. Thus,
if the convention of sending the LSB first is used, the word Red (decimal 82,101,100) is converted to the binary data sequence (commas are only shown for clarity, they are not part of the data sequence)

\[0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0, \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0, \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0\]

In Matlab the following function, called asc2bin can be used to convert directly from a textstring txt to a binary data sequence dn:

```
function dn = asc2bin(txt)
%asc2bin Text to ASCII to serial binary conversion
% 8 bits per character, LSB first
% >>>>> dn = asc2bin(txt) <<<<
% where txt is the input text string
% dn is binary output sequence

dec = double(txt);  % Text to ASCII (decimal)
p2 = 2.^(0:2^7-1);  % 2^0,2^1,...,2^7
B = mod(1000*dec,2); % Decimal to binary conversion
%Cols of B are bits of chars
dn = reshape(B,1,numel(B));  %Bytes to serial conversion
```

Running this with Red in the Matlab workspace results in

```
>> dn = asc2bin('Red') %ASCII -> serial binary conversion
dn =
  Columns 1 through 14
       0     1     0     0     1     0     1     0     1     0     0     1
  Columns 15 through 24
                      1     0     0     0     1     0     0     1     1     0
```

**Conversion from Serial Binary to Text.** Assuming that the sequence dn can be successfully transmitted and received at the channel output, the next step is to convert it back to a textstring. This requires that the binary data string is decomposed into 8-bit segments (chopping off any extraneous bits at the end) which are then converted back to decimal ASCII codes that can then be displayed as text using the char function, which works as follows:

```
>> char([82 101 100])
ans =
   Red
```
A complete function, called bin2asc that converts a binary data string dn back to a textstring txt is shown below.

```matlab
function txt = bin2asc(dn)
%bin2asc Serial binary to ASCII to text conversion
% 8 bits per character, LRB first
% >>>> txt = bin2asc(dn) <<<<
% where dn is binary input sequence
% txt is output text string

L = length(dn); %Length of input string
L8 = 8*floor(L/8); %Multiple of 8 length
B = reshape(dn(1:L8),6,L8/8); %Cols of B are bits of char
p2 = 2.^0:7; %Powers of 2
dec = p2*8'; %Binary to decimal conversion
txt = char(dec); %ASCII (decimal) to text
```

Thus, using bin2asc with asc2bin(txt) as input should return the text in txt as demonstrated below:

```matlab
>> bin2asc(asc2bin('This is a test!'))
ans =
This is a test!
```

Modulation of CT Signal. A communication channel such as a twisted pair cable or a coaxial cable needs a CT waveform as input. Thus, the binary sequence dn either needs to be converted into a suitable CT signal or needs to modify one or more parameters of a CT waveform depending on the dn values. This process is called modulation. If the CT signal is a pulse whose amplitude is controlled by the values of dn, then the modulation method is called pulse amplitude modulation (PAM). Pulse shapes that are used in practice are rectangular, triangular, and sinusoidal shaped pulses. By far the simplest of these to use are rectangular pulses. Consider the following Matlab script to generate a rectangular binary PAM signal from the textstring Red:

```matlab
Fs = 100; %Sampling rate
M = 10; %Multiplier for pulse width
txt = 'Red';
dn = asc2bin(txt); %Serial binary string
S = ones(M,1)*dn; %Signal matrix, M copies of dn
st = reshape(S,1,numel(S)); %Rectangular PAM signal
tt = [0:length(st)-1]/Fs; %Time axis
```
A plot of $s(t)$ versus $t$ looks as follows (blue lines are straight line interpolations, red dots are actual sample points)

What the Matlab script file does is to create $M$ copies of the binary sequence $d_n$ as rows of the matrix $S$. The content of $S$ is then read out columnwise, thereby creating a rectangular pulse of width $M$ samples for each bit in $d_n$. With the (arbitrary) choice of $F_s = 100$ Hz, this yields a pulsewidth of 0.1 sec for each of the rectangular pulses.

**PAM Signal Receiver.** At the receiver, the PAM signal will in general be attenuated, the pulse shape will be altered by the bandlimitation of the channel, and noise may have been picked up during transmission. Here is an example of what the received PAM signal might look like.

One strategy to demodulate this signal is to use every $M$-th sample at times where the signal has its positive and negative peaks, and then decide for each whether it is above or below a threshold in the middle of the amplitude range between the positive and negative peaks. The optimal sampling points are indicated with purple circles in the graph below and the threshold above which a received sample corresponds to $d_n = 1$ and below which a sample corresponds to $d_n = 0$ is the green dash-dotted line.
Thus, despite the impairments picked up during transmission, the received signal $r(t)$ can be converted back to the transmitted message Red.

### 1.3 General First Order CT Systems

The differential equation of a general first order CT system is

$$y^{(1)}(t) + a_1 y(t) = b_0 x^{(1)}(t) + b_1 x(t).$$

To determine $y(t)$ as a function of $x(t)$ and to draw a block diagram of this system, it is advantageous to use the (one-sided) Laplace transform.

**Definition:** The one-sided or unilateral Laplace transform $X(s)$ (if it exists) of a CT signal $x(t)$ is defined as

$$\mathcal{L}[x(t)] = X(s) = \int_{0^-}^\infty x(t) e^{-st} dt,$$

where $t = 0^-$ is a time instant just to the left of $t = 0$.

**Differentiation Property:** Let $X(s)$ be the one-sided Laplace transform of $x(t)$. Then the one-sided Laplace transform of the first derivative $x^{(1)}(t)$ is

$$\mathcal{L}[x^{(1)}(t)] = s X(s) - x(0^-),$$

where $x(0^-)$ is the initial value of $x(t)$ at time $t$ just before zero.

Assuming zero initial conditions, the Laplace transform of the differential equation of a general first order system becomes

$$s Y(s) + a_1 Y(s) = b_0 s X(s) + b_1 X(s) \quad \implies \quad (s + a_1) Y(s) = (b_0 s + b_1) X(s).$$

Solving for $Y(s)$ yields

$$Y(s) = \frac{b_0 s + b_1}{s + a_1} X(s) = H(s) X(s), \quad \text{where} \quad H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1},$$
is called the system function of the CT system. To obtain a block diagram representation of this system, write
\[ Y(s) = \frac{(b_0 s + b_1) X(s)}{s + a_1} = \frac{(b_0 s + b_1) Q(s)}{s + a_1}, \]
where \( Q(s) = \frac{X(s)}{s + a_1} \).

The key idea is now to first make a block diagram that computes \( Q(s) \) from \( X(s) \) and then, in a second step, to compute \( Y(s) \) from \( Q(s) \). Multiplying the equation for \( Q(s) \) by \( s + a_1 \) yields
\[ s Q(s) = a_1 Q(s) = X(s) \quad \implies \quad s Q(s) = X(s) - a_1 Q(s). \]
This is readily implemented by the block diagram shown below.

Next, using
\[ Y(s) = (b_0 s + b_1) Q(s) = b_0 s Q(s) + b_1 Q(s), \]
the computation of the output \( Y(s) \) is added as shown in blue in the following block diagram.

A physical interpretation for this is that \( q(t) \leftrightarrow Q(s) \) is the state variable of the system, and in a first step the new state is computed from the old state (through \( a_1 \) and the feedback loop) and from the current input \( x(t) \leftrightarrow X(s) \). In a second step, \( y(t) \leftrightarrow Y(s) \) is computed from the new state and the current input input.

**Inverse System.** If the original system has system function
\[ H(s) = \frac{b_0 s + b_1}{s + a_1}, \]
then the inverse system has system function
\[ H_I(s) = H^{-1}(s) = \frac{s + a_1}{b_0 s + b_1} = \frac{1}{b_0} \frac{s + a_1}{s + b_1/b_0} , \]
where for the last equality \(b_0 \neq 0\) is required. The importance of an inverse system \(H_I(s)\) is that

\[
\text{if } Y(s) = H(s) X(s), \quad \text{then } H_I(s) Y(s) = H_I(s) H(s) X(s) = X(s),
\]

i.e., \(X(s)\) can be recovered from \(Y(s)\). Note, however, that even if \(H(s)\) is the system function of a stable system, \(H_I(s)\) may be the system function of an unstable system and thus it may not be possible to recover \(X(s)\) exactly from \(Y(s)\).

1.4 Running Simulink Models from Matlab

Suppose you would like to obtain the response \(y(t)\) of the following first order CT Simulink model to an arbitrary input signal \(x(t)\), e.g., such as the one shown below

for different values of the parameters \(a_1, b_0,\) and \(b_1\) in the system function

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1}.
\]
Note that in the Simulink model \( a_0 = a(1) = 1, \ a_1 = a(2), \ b_0 = b(1), \) and \( b_1 = b(2) \) because indexing of the vectors \( a = [a(1) \ a(2)] \) and \( b = [b(1) \ b(2)] \) in Matlab starts at 1 rather than at 0.

The Simulink model CT1sys001 shown above can be run from the Matlab workspace and from Matlab script files and functions using the command

\[
\text{sim('CT1sys001', [t(1) \ t(end)])}
\]

where \( t(1) \) is the start time for the simulation and \( t(end) \) is the stop time. All other settings will be taken from the simulation configuration settings (under “Simulation” select “Configuration Parameters...” to open the dialog window) of the model. To use the fixed-step ode3 solver with step size \( 1/F_s \) (so that the model is run with a sampling rate of \( F_s \)), change the “Solver” page of the “Simulation Configuration Parameters” for your model to the settings shown below.

To import the time axis \( t \) and the input signal(s) \( u \) from the Matlab workspace and return all data points of the results (in \( t_{out}, \ y_{out} \)) to the Matlab workspace, use the settings shown below on the “Data Import/Export” page of the “Simulation Configuration Parameters” for your model. In particular, make sure that the box next to “Limit data points to last:” is unchecked.
Example: Find $y(t)$ for the input signal $x(t)$ given earlier when

$$H(s) = \frac{Y(s)}{X(s)} = \frac{s/100 + 1}{s + 1}.$$

Assuming that $x(t)$ is already available in the row vector $xt$, the commands to setup the simulation and run it are:

```matlab
Fs = 100; %Sampling rate
tt = [0:length(xt)-1]/Fs; %Time axis
a = [1 1]; %Denominator of H(s)
b = [1/100 1]; %Numerator of H(s)
t = tt'; %t for Simulink, must be column vector
u = xt'; %Input u for Simulink, column vector
sim('CT1sys001',[t(1) t(end)]); %Run Simulink model
yt = yout'; %Output signal y(t)
plot(tt,yt),grid %Plot result of Simulation
```

Note that the input ($t$, $u$) and output ($tout$, $yout$) vectors associated with Simulink are column vectors. The result of the simulation is shown in the following annotated graph.
As can be seen, the response of the system is not very fast. If a faster response is desired, the same simulation can be run again with $a$ and $b$ changed to

\[
a = [1 \ 10]; \quad %\text{Denominator of } H(s)
\]
\[
b = [1/100 \ 10]; \quad %\text{Numerator of } H(s)
\]

which corresponds to

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{s/100 + 10}{s + 10}.
\]

Now the output reacts more quickly as can be seen from the graph below.

Thus, running a Simulink model from a Matlab script file with parameter values that can be set in the Matlab workspace, is a powerful and flexible way to try different settings and answer what-if questions.

**Inverse Filter for Channel Equalization.** Consider again the received PAM signal shown below.
From the graph the time constant of the channel is estimated to be about 33 ms (time from 0 to \((1 - e^{-1})A\) where \(A \approx 0.6\)). Thus, the system function of the (first order) channel is estimated as (for a gain of 1)

\[
H(s) \approx \frac{1/0.033}{s + 1/0.033} = \frac{30}{s + 30} .
\]

The inverse system is then

\[
H_I(s) = \frac{1}{30} \frac{s + 30}{1} \approx \frac{1}{30} \frac{s + 30}{s/100 + 1} = \frac{1}{30} \frac{100s + 3000}{s + 100} ,
\]

where \(s/100 (s\) multiplied by a number much smaller than 1 but not smaller than \(1/F_s\)) has been added to the denominator of \(H_I(s)\) to make it implementable using an integrator rather than a differentiator block. To apply the inverse filter to the received signal \(r(t)\) in Simulink, use the Matlab script

```matlab
a = [1 100]; %Denominator of HI(s)
b = (1/30)*[100 3000]; %Numerator of HI(s)
t = tt'; %Time axis for Simulink model
u = rt'; %Input r(t) for HI(s) system
sim('CT1sys001', [t(1) t(end)]) %Run Simulink model
rIt = yout'; %Output is rI(t)
```

A plot of \(r(t)\) and the resulting \(rI(t)\) is shown below.
The shape of the received PAM signal is closer to rectangular after the inverse of the (estimated) channel filter. But, as a side effect, the noise in the received signal has been amplified leading to a smaller signal-to-noise ratio. In general a compromise needs to be made between compensating for the bandlimitation of the channel and amplification of the noise in the received signal. A more intelligent approach which looks at the whole sequence of received data samples yields more optimal results but is beyond the scope of this lab.

1.5 Unit Step and Unit Impulse Response in Simulink

Assume that the following general first order model is used in Simulink:

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1},
\]

in the Matlab workspace. Note that because Matlab indexes the vector components starting with index 1, \( b_0 = b(1) \) and \( b_1 = b(2) \). Similarly, \( a = [a(1) \ a(2)] \) is used to set the denominator coefficients of \( H(s) \) in the Matlab workspace with \( a_0 = a(1) = 1 \) and \( a_1 = a(2) \).

To obtain the unit step and impulse responses for this model with a fixed sampling rate \( F_s \) from a Matlab script file, a time axis and a unit step need to be generated and the coefficients \( b \) and \( a \) for the numerator and the denominator of \( H(s) \) need to be set. If

\[
H(s) = \frac{5s + 50}{s + 50} \quad \iff \quad h(t) = 5 \delta(t) - 200 e^{-50t} u(t),
\]

and thus

\[
G(s) = \frac{5s + 50}{s(s + 50)} \quad \iff \quad g(t) = (1 + 4 e^{-50t}) u(t),
\]

the first few lines of the script file look as follows:
Fs = 8000; % Sampling rate in Hz
tlen = 0.1; % Time duration
tstep = 0.01; % Step time
b = [5 50]; % Numerator of H(s)
a = [1 50]; % Denominator of H(s)

% Time axis
tt = [0:round(tlen*Fs)]/Fs; % Time axis
% Prepare unit step
ut = zeros(size(tt)); % Prepare unit step
ix = find(tt>=tstep); % Indexes for tt>=tstep
ut(ix) = ones(size(ix)); % Unit step

This creates a time axis tt from 0 to tlen and a unit step signal ut that goes from 0 to 1 at time tstep. A sampling rate of Fs = 8000 Hz was chosen to obtain a good graphical representation for the unit step u(t). To run the Simulink model CT1sys001 with this input data, use

```
t = tt'; % t vector for Simulink
u = ut'; % u vector for Simulink
sim('CT1sys001',[t(1) t(end)]); % Run Simulink model
% with ode3 fixed-step solver
% and step size 1/Fs
```

The vectors t and u must be column vectors with the same length. Under “Simulation”, “Configuration Parameters...” the following settings should be used:

- On the solver page under “Solver options” select the “Fixed-step” solver type and the ode3 (Bogacki-Shampine) solver. Set the “Fixed step size:” to 1/Fs.
- On the data import/export page check the box next to “Input: [t u]” so that the input data for the simulation is loaded from the Matlab workspace. Make sure that the boxes next to “Time: tout” and “Output: yout” are checked as well. Uncheck the box next to “Limit data points to last: 1000” so that all the output data from the simulation is returned to the Matlab workspace.

After running the simulation the command plot(tout,yout) can be used to plot the step response of the system. To obtain both g(t) and h(t) for plotting use

```
gt = yout; % Step response
ht = Fs*diff([0;gt]); % Impulse response, equal to % derivative of step response
```

right after running the Simulink model. An example of how to display both g(t) and h(t) is shown in the plots below.
If $b_0 \neq 0$ in $H(s)$ then $h(t)$ will contain an impulse at the time where the step occurs. To see the size of the impulse as well as the details of $h(t)$ right after the impulse, two plots may be needed, one which uses the autoscale function and one which uses manual scaling with the `ylim` command.

2 Prelab Questions

P1. Matlab Practice. (a) Which Matlab commands were used to obtain the answers shown (replace `command` or `????` by an actual Matlab command or statement):

(i) $x=(????);$, $x^2 \Rightarrow \text{ans} = -5.0000 +12.0000i$
   $\text{abs}(x)^2 \Rightarrow \text{ans} = 13.0000$

(ii) $u=(????);$, $v=\text{cumsum(ones(1,4))};$, $u*v \Rightarrow \text{ans} = \begin{array}{cccc} 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 5 & 10 & 15 & 20 \end{array}$

(iii) $n=[2 3:2:30];$, $ix=(????);$, $n(ix) \Rightarrow \text{ans} = 2 3 5 7 11 13 17 19 23 29$

(b) If the statement for $p2$ in the function `asc2bin` given in the introduction is changed to

\[ p2 = 2 \cdot (-7:0); \]
how would this affect the ASCII text to binary conversion? How would the \texttt{bin2asc} function have to be changed so that

\[
\text{\texttt{bin2asc(asc2bin('any text'))}} = \text{'any text'}
\]

**P2. Step Response of General First Order System.** Consider a first order system with system function

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1}.
\]

Determine the unit step response \( g(t) \) in terms of \( a_1, b_0, b_1 \).

**P3. First Order System Parameter Estimation.** The following graph shows the unit step response of a first order system with system function

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1}.
\]

Determine \( a_1, b_0, b_1 \).
3 Lab Experiments

E1. CT Signals in Matlab. (a) The goal of this experiment is to gain some intuitive experience for the question of how to choose the sampling rate \( F_s \) for the sampled representation of a CT signal \( x(t) \) in Matlab. Make a script file, e.g., called \texttt{sine100.m}, using the following Matlab commands:

\begin{verbatim}
Fs = input('Enter sampling rate Fs in Hz: ');
f0 = 100; % Frequency of sine
tlen = 5e-2; % Signal duration in sec
tt = [0:round(tlen*Fs)]/Fs; % Time axis
xt = sin(2*pi*f0*tt); % Sinewave, frequency f0
\end{verbatim}

This creates a discrete time axis \( tt \) of duration \( tlen \) with time instants spaced \( 1/Fs \) seconds apart. The last statement generates the signal \( xt \) which is a 100 Hz sine, sampled with rate \( Fs \). Plot \( xt \) versus \( tt \) for \( Fs \) equal to 200, 400, 800, 1600, 3200, etc. What is the smallest \( Fs \) that yields a “nice” and representative graph of a 100 Hz CT sinusoid?

After generating the sine in \( xt \), use the command

\begin{verbatim}
rt = sign(xt); % Sine -> rectangular
\end{verbatim}

to generate a rectangular signal \( rt \) with frequency 100 Hz and sampling rate \( Fs \). Note that \texttt{sign} stands for the \texttt{signum} function which is defined as

\[
\text{sgn}(x) = \begin{cases} 
+1, & x > 0, \\
0, & x = 0, \\
-1, & x < 0. 
\end{cases}
\]

Plot \( rt \) versus \( tt \) for \( Fs \) equal to 200, 400, 800, 1600, 3200, etc. What is the smallest \( Fs \) that yields a “nice” and representative graph of a 100 Hz rectangular CT signal?

(b) Implement the Matlab functions \texttt{asc2bin} and \texttt{bin2asc} that were given in the introduction. Test that you can generate a binary sequence \( dn \) from a textstring using \texttt{asc2bin} and that you can obtain the correct textstring again by using \texttt{bin2asc} with input \( dn \). Recreate the PAM signal \( s(t) \) for the text \texttt{Red} that was given in the introduction.

(c) The \texttt{wav} file \texttt{PAMsig003.wav} contains a received PAM signal \( r(t) \) with the same parameters \( M = 10 \) and \( F_s = 100 \) Hz as the PAM signals discussed in the introduction. Write a Matlab script file that reads \( r(t) \) and outputs the transmitted text message. You will have to optimize the time instants at which you sample \( r(t) \) and you will need to choose a threshold above which \( dn \) is set to 1 and below which \( dn \) is set to 0. To read the \texttt{wav} file in Matlab use the command

\begin{verbatim}
[r Fs] = wavread('PAMsig003.wav')
\end{verbatim}

Note that \( r \) will be a column vector which can be converted to a row vector using \( r = r' \); if necessary.

E2. General First Order System Simulation. Build the following general first order system in Simulink.
Note that the gain parameters were set to $a(2)$, $b(1)$ and $b(2)$ so that the specifics of the model can be controlled from $a=[a(1) a(2)]$ and $b=[b(1) b(2)]$ in Matlab. The corresponding system function is

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 s + b_1}{s + a_1} = \frac{b(1)s + b(2)}{s + a(2)},$$

where the last expression is written in terms of the elements of vectors $a$ and $b$ in Matlab, taking into account that indexing in Matlab starts at 1 rather than 0. But note that $a_1$, $b_0$, $b_1$ refer to the gains in the block diagram:

(a) Use Simulink and Matlab script files to make labeled plots of the unit step responses $g(t)$ and the unit impulse responses $h(t)$, similar to the ones shown in the introduction, of the following first order CT systems

(i) $H(s) = \frac{0.1s + 10}{s + 10}$,

(ii) $H(s) = \frac{10s + 1}{s + 10}$,

(iii) $H(s) = \frac{0.1s + 100}{s + 10}$,

(iv) $H(s) = \frac{10s + 10}{s + 10}$.
Make sure to choose the length of the time axis and the plot parameters in each case such that the essential features of $g(t)$ and $h(t)$ are clearly visible. Which of these systems is closest to an ideal differentiator, which is closest to an ideal integrator. Of the remaining two systems, which behaves more like an integrator? Explain your reasoning.

(b) Determine the system function $H_I(s)$ of the inverse system of

$$H(s) = \frac{0.1s + 10}{s + 10}.$$ 

Use Simulink and a Matlab script to plot and label the unit step response $g(t)$ and the unit impulse response $h(t)$ of the inverse system $H_I(s)$. Adjust the time axis and the plot parameters such that the essential features of $g(t)$ and $h(t)$ are clearly visible. Simulate the cascade of $H(s)$ and $H_I(s)$ in Simulink by using $\text{yout}$ from the output of $H(s)$ as input of $H_I(s)$. What should the overall step and impulse responses look like? Does your simulation give the desired result?

(c) Verify that, if you use the parameters which you found in prelab problem P3, you indeed get the given unit step response from a simulation in Simulink. Make a labeled plot of the unit impulse response as well. What is the most characteristic difference of this system compared to the systems in part (a)?

E3. Data Transmission Using Pulse Amplitude Modulation and First Order System as Channel. The goal of this experiment is to simulate the transmission of a PAM signal that contains a text message over a bandlimited channel. If the bandlimitation is not too severe then the text message can be recovered by choosing the right threshold between binary 0 and 1 and by selecting the right time instants for sampling the PAM signal. But if the bandlimitation is strong enough to create intersymbol interference (ISI), then an inverse filter may have to be used to equalize the channel before the text message can be recovered.

(a) Write a Matlab script file that converts an ASCII text message to serial binary and then to a rectangular PAM signal $s(t)$ with an expansion factor of $M$. Use $M = 100$ and $F_s = 1000$ Hz for the sampling rate. An example of $s(t)$ for the text “Quick” is shown below.
(b) Write a Matlab script file that receives a PAM signal $r(t)$ with expansion factor $M$ and converts it back to a text message. Your script file must have input parameters $\text{dly}$ (delay) and $\text{thr}$ (threshold) so that you can adjust by how much the sampling time is delayed from the beginning of the PAM pulses and at which level the threshold is set above which the received symbol is a binary 1 and below which it is a binary 0. An example that shows the threshold (green dashed line) and the sampling times (purple circles) is shown in the following graph.

![Received PAM Signal](received_pam_signal.png)

Test your receiver script file together with transmitter script file from part (a). Use $M = 100$ and $F_s = 1000$ Hz. A PAM test signal with the same parameters is also available in the wav file PAMsig012.wav.

(c) Use your script from part (a) to generate a PAM signal $s(t)$ with $M = 100$ and $F_s = 1000$ Hz. Use the text “Quick” or any other short text message. Use Simulink to pass the signal $s(t)$ through a channel with system function

$$H(s) = \frac{0.1s + \omega_1}{s + \omega_1}, \quad \text{with } \omega_1 = 30.$$  

The output from the Simulink channel model is the received PAM signal $r(t)$. It should look similar to the following example.

![Received PAM Signal](received_pam_signal.png)
Use your PAM receiver script to recover the message from $r(t)$. Adjust $\text{dly}$ (delay) and $\text{thr}$ (threshold) as necessary. Next, change $\omega_1$ in $H(s)$ from $\omega_1 = 30$ to $\omega_1 = 10$. What do you observe? Can you still recover the message signal? If not, try passing $r(t)$ through an inverse $H^{-1}(s) = H^{-1}(s)$ of the channel filter $H(s)$.

(d) Apply your insights and results from part (c) to recover the text message from the noisy, scaled and bandlimited PAM signal $r(t)$ in the wav file `PAMsig025.wav`. It uses $M = 100$ and $F_s = 1000$ Hz.

Appendix A: Introduction to MATLAB

A.1 Background

MATLAB (the name stands for MATrix LABoratory) is an interactive system for solving scientific and technical computing problems. The basic data elements are arrays in one or more dimensions. The original purpose was to create a user interface that provides easy access to free matrix software (EISPACK and LINPACK) that was developed in 1972–1973 in FORTRAN. Over time MATLAB has evolved into a standard educational tool as well as a high-productivity tool for industrial research and implementations. The MATLAB system now consists of a desktop tool and workspace, a mathematical function library, a programming language, and an extensive set of graphics facilities.

A.2 MATLAB Basics

MATLAB can be used directly in interactive mode by typing commands one line at a time, or m-files (either scripts or functions) which contain code in the MATLAB programming language can be run.

Some general purpose commands to get started with Matlab are shown in the following table:

<table>
<thead>
<tr>
<th>General Purpose Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
<tr>
<td>help topic</td>
</tr>
<tr>
<td>helpwin</td>
</tr>
<tr>
<td>lookfor string</td>
</tr>
<tr>
<td>demo</td>
</tr>
<tr>
<td>who, whos</td>
</tr>
<tr>
<td>^C (control-C)</td>
</tr>
<tr>
<td>quit, exit</td>
</tr>
</tbody>
</table>

Workspace not saved automatically |

Matlab also provides a number of file system functions and commands that allow you to list file names, view and delete m-files (the script and function files of Matlab), show and
change the current directory or folder, etc. Here is a table of the most important file system functions:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd, pwd</td>
<td>Show present working directory or folder</td>
</tr>
<tr>
<td>cd path</td>
<td>Change to directory or folder given by path</td>
</tr>
<tr>
<td>dir</td>
<td>List all files in the current directory or folder</td>
</tr>
<tr>
<td>what</td>
<td>List Matlab-specific files in the current directory or folder</td>
</tr>
<tr>
<td>type filename</td>
<td>List m-file filename.m in command window</td>
</tr>
<tr>
<td>edit</td>
<td>Opens a new editor window for editing m-files</td>
</tr>
</tbody>
</table>

Certain characters and variable names have predefined uses in Matlab. Here is a table with the most important variable names:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ans</td>
<td>Default variable used for results</td>
</tr>
<tr>
<td>eps</td>
<td>Floating point relative accuracy (“machine epsilon”)</td>
</tr>
<tr>
<td>namelengthmax</td>
<td>Maximum length of variable names used</td>
</tr>
<tr>
<td>i and j</td>
<td>Stands for square root of -1</td>
</tr>
<tr>
<td>Inf</td>
<td>Stands for infinity</td>
</tr>
<tr>
<td>NaN</td>
<td>Stands for Not a Number</td>
</tr>
<tr>
<td>pi</td>
<td>3.14159265358979</td>
</tr>
</tbody>
</table>

In general, variable names in MATLAB consist of a letter, followed by any number of letters, digits, or underscores. Spaces are not allowed and the letters are treated as case sensitive in variable names. Special characters and operators are shown in the next table.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>∗</td>
<td>Matrix and scalar multiplication</td>
</tr>
<tr>
<td>.*</td>
<td>Array (elementwise) multiplication, note the dot!</td>
</tr>
<tr>
<td>/</td>
<td>Matrix and scalar divide</td>
</tr>
<tr>
<td>./</td>
<td>Array (elementwise) divide, note the dot!</td>
</tr>
<tr>
<td>∧</td>
<td>Matrix and scalar power</td>
</tr>
<tr>
<td>.∧</td>
<td>Array (elementwise) power, note the dot!</td>
</tr>
<tr>
<td>[]</td>
<td>Square brackets, used to enter matrices and vectors</td>
</tr>
<tr>
<td>'</td>
<td>Complex conjugate vector/matrix transpose or quotes to enclose text</td>
</tr>
<tr>
<td>.'</td>
<td>Vector/matrix transpose, note the dot!</td>
</tr>
<tr>
<td>:</td>
<td>Colon, used to generate vectors and pick out rows/columns from matrices</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon, used as row delimiter and to suppress echoing</td>
</tr>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>≈</td>
<td>Not equal</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
<tr>
<td>...</td>
<td>Continuation (on next line)</td>
</tr>
<tr>
<td>%</td>
<td>Comment</td>
</tr>
</tbody>
</table>

The colon (:) symbol in particular is used quite extensively and in several different ways. To learn more about it, type `help colon` at the Matlab command prompt.

A few simple examples of using the MATLAB desktop environment interactively are shown below. Note that everything after the `%` sign is interpreted as a comment.

```
>> 2^8-1 %Standard arithmetic
ans =
 255

>> exp(-1) %Exponential function
ans =
 0.3679

>> log(10) %Natural logarithm
ans =
 2.3026

>> log10(10) %Base 10 logarithm
ans =
 1
```
```matlab
>> 1/sqrt(2)  %Square root
ans =
    0.7071

>> num2str(1/sqrt(2),10)  %10 digits of result displayed
ans =
    0.7071067812

>> sin(pi/4)  %sine of 90 degrees
ans =
    0.7071

>> factor(81472369)  %Integer factorization
ans =
    11   7406579

>> pi,floor(pi),ceil(pi)  %Floor and ceiling function
ans =
    3.1416
ans =
    3
ans =
    4

>> mod(13,5)  %13 modulo 5
ans =
    3

>> (1+j)^2  %Complex number squared
ans =
    0 + 2.0000i

>> abs(1+j),angle(1+j),conj(1+j)  %Operations on complex number
ans =
    1.4142
ans =
    0.7854
ans =
    1.0000 - 1.0000i

>> real(1+2j),imag(1+2j)  %Real and imaginary parts
ans =
    1
ans =
    2
```
A.3 Vectors, Matrices, and Polynomials in MATLAB

MATLAB is a matrix and vector oriented computing environment. Most commands and functions such as \( \exp \), or \( \log \), or \( \sin \), etc, can be used directly on vectors and matrices.

To enter a row vector \( \mathbf{u} = [0 \ 1 \ 2 \ 3 \ 5 \ 4] \), type

\[
\mathbf{u} = [0 \ 1 \ 2 \ 3 \ 5 \ 4]
\]

at the command prompt. MATLAB will immediately echo the result in the workspace. To suppress this, enter \( \mathbf{u} \) as

\[
\mathbf{u} = [0 \ 1 \ 2 \ 3 \ 5 \ 4];
\]

i.e., with a trailing semicolon. To enter the same vector, but as a column vector \( \mathbf{v} \), type

\[
\mathbf{v} = [0;1;2;3;5;4];
\]

Here the first five semicolons act as row delimiters, and the last semicolon suppresses the echo printout of \( \mathbf{v} \). Alternatively, one could enter \( \mathbf{v} \) as a row vector and then transpose it using a single quote as follows

\[
\mathbf{v} = [0 \ 1 \ 2 \ 3 \ 5 \ 4]';
\]

Once \( \mathbf{u} \) and \( \mathbf{v} \) are entered, matrix multiplication can be used to compute \( \mathbf{u} \cdot \mathbf{v} \) (the inner product between the two vectors) and \( \mathbf{v} \cdot \mathbf{u} \) (the outer product between the two vectors). To compute the componentwise (or array) product between the vectors \( \mathbf{u} \) and \( \mathbf{v} \), use the command \( \mathbf{u} \cdot \mathbf{v}' \), “dot” in front of \( * \). Note that the componentwise product can only be taken between two vectors (or matrices) if their dimensions are the same, which is the reason why \( \mathbf{v} \) is transposed in \( \mathbf{u} \cdot \mathbf{v}' \). Here’s a summary of these commands:

\[
\begin{align*}
\mathbf{u} &= [0 \ 1 \ 2 \ 3 \ 4 \ 5]; & \text{Row vector, 6 components} \\
\mathbf{v} &= [0;1;2;3;4;5]; & \text{Column vector, 6 components} \\
\mathbf{u} \cdot \mathbf{v}; & \text{Inner product, result is scalar} \\
\mathbf{v} \cdot \mathbf{u}; & \text{Outer product, result is 6x6 matrix} \\
\mathbf{u} \cdot \mathbf{v}'; & \text{Componentwise (or array) product, result is row vector, 6 components}
\end{align*}
\]

Caution: If \( \mathbf{v} \) contains complex-valued components, then the elements of \( \mathbf{v}' \) are the complex conjugates of the elements of \( \mathbf{v} \). To obtain the transpose of \( \mathbf{v} \) without complex conjugation use the command \( \mathbf{v}.' \), with a “dot” in front of \( ' \).

A \( 2 \times 3 \) matrix \( \mathbf{A} = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 5 & 4 \end{bmatrix} \) is entered in Matlab as follows:

\[
\mathbf{A} = [0 \ 1 \ 2; 3 \ 5 \ 4];
\]

To pick out one or more elements from \( \mathbf{A} \) use the commands:
A(1,2) %Picks out 2'nd element in 1'st row
A(1,:) %Picks out whole 1'st row
A(:,2) %Picks out whole 2'nd column
A(2,end) %Picks out last element in 2'nd row

Note that Matlab indexes all vectors and matrices beginning with 1, and not with 0 as is often done in mathematical statements and formulas. To determine the size of an existing matrix A or vector u, use the commands

\begin{verbatim}
size(A) %Returns size of matrix A: rows and columns
length(u) %Returns length of vector u: max{rows,columns}
\end{verbatim}

To generate a $m \times n$ matrix and fill it with all zeros or ones use the commands $\text{zeros}(m,n)$ and $\text{ones}(m,n)$. Here is an example that creates two $3 \times 5$ matrices:

\begin{verbatim}
A = zeros(3,5); %A is 3x5 matrix of all zeros
B = ones(size(A)); %B is matrix of all ones, same size as A
\end{verbatim}

Often it is necessary to fill a vector with a consecutive set of numbers, e.g., to generate a time axis as in $t = [0 0.1 0.2 0.3 \ldots 1.0]$, or a set of indexes as in $n = [0 1 2 3 \ldots 10]$ or $k = [16 18 20 22 \ldots 32]$. Rather than typing in all elements individually in such cases, the following commands can be used in Matlab:

\begin{verbatim}
t = 0:0.1:1; %Generates t=[0 0.1 0.2 ... 1.0]
n = 0:10; %Generates n=[0 1 2 3 4 ... 10]
k = 16:2:32; %Generates k=[16 18 20 22 ... 32]
\end{verbatim}

The general form of the command is $v_1:inc:v_2$ where $v_1$ is the initial value, $v_2$ is the final value, and $inc$ is the increment (which can be positive or negative). For example, to reverse the vector $t$ given above, the command

\begin{verbatim}
trev = t(end:-1:1);
\end{verbatim}

can be used. In this case $\text{end}:-1:1$ first creates a vector with all indexes of $t$ in decreasing order. This vector of indexes is then used in $t(\text{end}:-1:1)$ to pick out the elements of $t$ in reverse order.

Two vectors or matrices A and B with the same number of rows can be concatenated using

\begin{verbatim}
C = [A B]; %Matrix C is concatenation of matrices A and B
\end{verbatim}

Similarly, two vectors or matrices A and B with the same number of columns can be stacked on top of each other using

\begin{verbatim}
C = [A;B]; %Matrix C is matrix A stacked on top of matrix B
\end{verbatim}

In the first case the number of columns of $C$ is the sum of the number of columns of $A$ and $B$. In the second case the number of rows of $C$ is the sum of the number of rows of $A$ and $B$. 

31
To check if two matrices $A$ and $B$ with the same dimensions are equal use

```matlab
isequal(A,B); %returns 1 if A and B are equal, 0 otherwise
```

A polynomial $p_n x^n + p_{n-1} x^{n-1} + \ldots + p_1 x + p_0$ is represented in MATLAB as a vector $p = [p_n, p_{n-1}, \ldots, p_1, p_0]$ of polynomial coefficients, starting with the highest degree on the left. The command $r=roots(p)$ returns the roots (or zeros) $r$ of the polynomial $p$. Conversely, $p=poly(r)$ converts the set of roots $r$ to a polynomial $p$. Examples:

```matlab
p1 = [1 0 -2 -4]; %polynomial p1=x^3-2*x-4
r1 = roots(p1); %returns roots r1=[2 -1+j -1-j]
r2 = [-2 -1+j -1-j] %roots r2=[-2 -1+j -1-j]
p2 = poly(r2); %returns polynomial p2=x^3+4*x^2+6*x+4
polyval(p2,[-2 -1 0 1 2]); %evaluates p2 at x=[-2 -1 0 1 2]
```

### A.4 Flow Control, Scripts, and Functions in Matlab

MATLAB has flow control commands to implement `if-else-end` constructions, `for` loops, `while` loops, and `switch-case` constructions.

**if-else-end** examples. To test if a vector (or set of vectors) $x$ is a column vector and to transpose it if necessary, the following statements can be used:

```matlab
[m n] = size(x); %Determine size of (vector) array x
if m<n
    x = x.'; %Transpose if x is array of row vectors
[end]
```

After the execution of these statements, $m$ is the length of each vector in the array $x$ and $n$ is the number of vectors in the array $x$.

The assignment

$$
d = \begin{cases} 
1, & \text{if } x > 0.5, \\
-1, & \text{if } x \leq -0.5, \\
0, & \text{otherwise}.
\end{cases}
$$

can be programmed in MATLAB as follows

```matlab
if x > 0.5
    d = 1;
elseif x <= -0.5
    d = -1;
else
    d = 0;
end
```
**for loop** example. To compute the integral

\[ y(t) = \int_0^t x(\tau) d\tau, \]

numerically, for \( t = T_s, 2T_s, 3T_s, \ldots, nT_s \) use the sum approximation

\[ y(kT_s) \approx T_s \sum_{i=1}^{k} x(iT_s), \quad k = 1, 2, 3, \ldots, n. \]

Noting that \( y(kT_s) = y((k-1)T_s) + T_s x(kT_s) \) and assuming \( y(0) = 0 \), this can be programmed in MATLAB as

```matlab
y(1) = x(1);
for k=2:n
    y(k) = y(k-1) + x(k);
end
y = Ts*y;
```

**for** loops are slow in MATLAB and you should only use them if there is no other good way to implement a computation. For the integration example above there is an internal MATLAB command, called `cumsum`, that achieves the same result with the following statement

```matlab
y = Ts*cumsum(x);
```

Example of a **while** loop. The following statements generate a random integer with 9 digits or less that is a prime number.

```matlab
p = round(1e9*rand); %Random integer, <=9 digits
if p==2*floor(p/2) %Check if even
    p = p-1;
end
while length(factor(p))>1 %Check if prime
    p = p-2; %Decrement by 2 if not prime
end
disp(int2str(p)) %Display prime number p
```

**while** loops are also slow in Matlab and should only be used when truly necessary.

**M-files.** Files that contain code in the MATLAB language are called M-files. There are two kinds of M-files, *script* files and *function* files. M-files are created and modified using either the MATLAB M-file editor (type `edit` at the command prompt to invoke it) or any other text editor.

A *script* file consists of a sequence of MATLAB commands that are executed in the workspace when the file is invoked. Scripts can generate new data and/or operate on data that is already in the workspace. They do not return any output arguments but the data they create remain
in the workspace. In addition, scripts can produce graphs, e.g., using the `plot` command. Any of the examples, especially the ones that use several lines of code, that were given earlier can be put in an M-file (a file with extension `.m`) and then run by invoking this M-file. An example of a script file is the following (saved as `sine1000.m`):

```matlab
% sine1000 Generates plot of 1000 Hz sinewave for different sampling rates Fs.
% Note: The first lines of comment are shown when you type `help sine1000` in the Matlab workspace.

Fs = input('Enter sampling rate Fs in Hz: ');
f0 = 1000;  % Frequency of sine in Hz
tlen = 1e-2;  % Signal duration in sec
tt = [0:round(tlen*Fs)]/Fs;  % Time axis in sec
st = sin(2*pi*f0*tt);  % Sinewave, frequency f0
subplot(211)  % Select subplot
plot(1000*tt,st,'-b',1000*tt,st,'.r')
grid  % Plot st, red: sample points
xlabel('t [ms]'),ylabel('s(t)')
str = 'Graph of s(t)=sin(2\pi f_0 t)';
str = [str ', f_0=' num2str(f0) ' Hz'];
str = [str ', F_s=' int2str(Fs) ' Hz'];
title(str)
legend('Straight line interpolation','Actual samples')
figure(gcf)  % Show current figure
```

If 11025 is entered for `Fs`, then this script file generates the graph shown below.

![Graph of s(t)=sin(2\pi f_0 t), f_0=1000 Hz, F_s=11025 Hz](image)

Functions are M-files that accept input data and return output data. When a function is invoked it operates on variables in its own workspace, separate from the main MATLAB workspace. Here is an example of a function M-file:

```matlab
% Functions are M-files that accept input data and return output data.
% When a function is invoked it operates on variables in its own workspace, separate from the main MATLAB workspace.
```
function y = reverse(x)
%reverse Returns the time reversal of vector x in y

if min(size(x))>1
    error('input must be vector')
end
y = x(end:-1:1); %Time reversed vector

The first line must start with the keyword function, followed by the function name and
order of arguments. The next few lines, up to the first blank or executable line, are comment
lines which are displayed when you type help reverse in the MATLAB workspace. The
first three lines of the executable code check that the input x is indeed a vector and not a
matrix and an error message is displayed if the smallest dimension of x is larger than 1. The
actual process of reversing x is then performed in the last statement of the function.

Note: If the name of an M-file that you create is the same as that of a function that already
exists in MATLAB, then your M-file may supersede the original MATLAB function. To
avoid ambiguities it is best to use names for your M-files that are different from existing
names (if in doubt about a name, type help name at the MATLAB prompt to see if it
already exists).