Signals and Systems 2024/2025 (LEEC & LEBIOM)

Lab Guide

November 2024

Introduction

This document describes the laboratory component of the Signals and Systems course. Its goal is to provide students with practical experience and a deeper understanding of the theoretical concepts through experimentation and visualisation. Working with both synthetic and real-world signals, you will gain insights into the behavior of systems in time and frequency, develop a hands-on grasp of the effects of operations like convolution, filtering, or Fourier transformation, and hone your problem-solving skills. When answering the questions in this guide and commenting on the results, you are expected to exercise critical thinking; rather than simply describing what was obtained in each experiment, it is more important to interpret the results and discuss them (Were they expected? Do they make sense? What do they mean?)
Students shall work in groups of two and submit via Fénix, by the end of the course, a concise report of the lab work containing the answers to questions marked with **Q*** in this guide, supporting code and results, and any additional comments deemed appropriate (note that questions in this guide may span several paragraphs and end with a horizontal line when followed by regular text). The guide includes other activities and questions, not marked with Q, which you are strongly encouraged to address but should not cover in your report. The report consists of a mixed notebook with executable code and text (Matlab live script, Jupyter notebook, Colab notebook. . .), as well as a pdf generated from it. The pdf should not exceed the equivalent of 10 pages. Groups of Portuguese-speaking students only should write the report in Portuguese, others in English. Grading will be individual, based on the submitted materials and on your performance in class.
This guide provides example code in Matlab. Supporting files are provided in Matlab and Python, and you may use either programming language in your report. If using Python (with SciPy), however, it is up to you to adapt the code. Throughout the work, students will work mainly with discrete-time signals using signal-processing software. However, several of the concepts and tools pertain to continuous-time signals, and we will handle those rather informally until the section on sampling. For example, we will generate sampled versions of continuous-time signals using a discretized time axis, but when looking at their spectra we will tweak things to make it look as though these are "native" continuous-time signals and spectra (which is *not* the case), hiding their true discrete-time nature. Or we might play these signals on the speakers and listen to the output without worrying much about the underlying processing needed for generating true continuous-time acoustic signals from a computer-based representation.

1 Signals

Sinusoidal signals

In this section you will generate sinusoidal signals and listen to them. We will define the granularity of the discretized time axis (i.e., how many "time ticks" it contains per second) with the variable `samplerate`. Start by generating a 2 s time axis as

```
samplerate = 16e3;
t = (0:1/samplerate:2)';
```

Then, generate variable `x` as a sinusoid with 10 Hz frequency

```
x = cos(2*pi*10*t);
```

Visualise the sinusoid with

```
plot(t,x); grid on
```

Locate the figure window, which may be hidden behind other windows. Become familiar with the controls to pan, zoom, turn data tips on and off, etc, Check that the sinusoid indeed has 10 periods per second. Note that the next plot command that you issue will overwrite the current plot unless you use `hold on` or create a new figure window with the `figure` command.
Now change the time axis duration to 1 s and the frequency of the sinusoid to 1000 Hz. Recreate the signal in `x` and visualise it. Initially, the plot will look like a solid rectangle, and you will have to zoom in to see its rapid oscillations.
Listen to the sinusoidal signal using the command

```
soundsc(x,samplerate)
```

Caution! When using earphones make sure that the volume is not too loud before inserting them in your ears.

Test your hearing limits

For the next test, set

```
samplerate = 48e3;
```

Generate several sinusoids with different frequencies and check the limits of your hearing. For reasons that will become clear later on, do not test frequencies above `samplerate/2`, i.e., 24 kHz. For this type of test it may be more convenient to generate a sinusoidal-like signal with linearly increasing or decreasing frequency (this is often known as a *linear chirp*) and measure how much time elapses before you stop hearing the sound. Knowing the start and stop frequencies, the total duration of the signal, and the elapsed time when you started/stopped hearing, it is a simple matter to derive the corresponding instantaneous frequency and thus establish the upper and lower limits of your hearing (regular-quality headphones and computer sound cards will influence the results, so don't take them too literally).
For example, to generate a linear chirp with total duration 5 s that starts at 0 Hz and attains a final frequency of 20 kHz use

```
t = (0:1/samplerate:5)';
x = chirp(t,0,5,20e3);
```

Listen to the signal and visualise it. Try Matlab's Signal Analyzer App to interactively view the signal in time and frequency. To test your lower hearing limit it may be a good idea to use a decreasing linear chirp. To continue, set `samplerate` back to its original value

```
samplerate = 16e3;
```

Periodic signals

Generate, over an interval of 2 s, the sum of sinusoids $x(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t)$ with $f_1 = 32$ Hz and $f_2 = 40$ Hz, which is a periodic signal.

Q1.1 Determine the period of $x(t)$ theoretically and check that it agrees with the period measured directly in the plot of `x`.

The function `ginput` is very useful for this, as it allows you to interactively click points on a plot using a hairline cursor and then obtain their horizontal and vertical coordinates as workspace variables. Typically, simply invoke it as

```
[xi,yi] = ginput;
```

to get the horizontal and vertical coordinates in vectors `xi` and `yi`, respectively, and hit `<enter>` when you're done marking points

Q1.2 As discussed above, this "continuous-time" signal `x` in Matlab is actually a discrete-time signal. Determine its theoretical period in samples and check against the direct measurement taken from `plot(x)`.

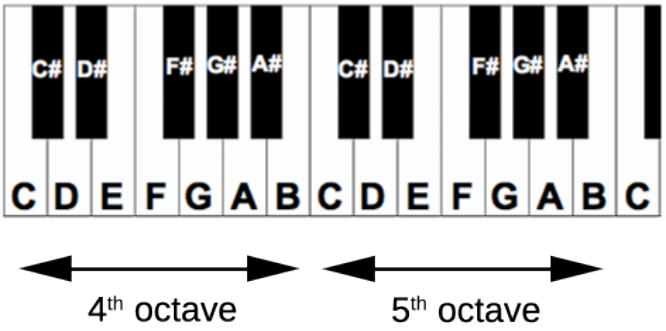
If you now downsample it by a factor of 3

```
y = downsample(x,3);
```

i.e., keeping only one sample for every three and discarding the other two, what will be the period of `y`, in samples? Why does this only seemingly disagree with direct measurement from `plot(y)`?

Musical notes

Each musical note corresponds to a well-defined frequency. The following figure shows two octaves of a piano keyboard, the 4th being the "middle C" octave that is a reference for musicians.



Keep in mind the following facts, which allow you to compute the absolute frequency of any musical note:

- The frequency of A4 is 440 Hz
- The musical scale is logarithmic. A full octave always corresponds to a 2:1 ratio. E.g., the frequency of C5 is double that of C4
- The ratio of frequencies between consecutive notes, including *sharp* ones labeled as # in the figure, is constant. Since there are 12 such notes in an octave, that ratio has to be $\sqrt[12]{2}$ to preserve the 2:1 ratio between octaves
- The ordering of notes in the "middle C" octave by ascending frequency is C4, C#4, D4, D#4, E4, F4, F#4, G4, G#4, A4, A#4, B4, and similarly for any other octave (in Portuguese notation, Dó4, Dó#4, Ré4, Ré#4, Mi4, Fá4, Fá#4, Sol4, Sol#4, Lá, Lá#4, Si4)

Q1.3 Determine the frequencies of C4, E4, F#4, G4, B4.

Use the `seqsin` function to generate a sequence of sinusoids and silence gaps. For example, the following

```
x = seqsin(samplerate, 440, 0.3, 0, 0.5, 466.164, 0.15);
```

will assign to `x` the samples of a sinusoid with frequency 440 Hz lasting for 0.3 s, followed by a silent gap (or *rest*) of 0.5 s and a sinusoid of frequency 466.164 Hz lasting for 0.15 s. In shorthand, we will represent this as "A4(0.3), R(0.5), A4#(0.15)".

Now create `x` as the following sequence and listen to the resulting signal:

R(0.7), E4(0.25), E4(0.25), F4(0.25), F#4(0.5), A4(0.25),

F#4(0.25), A4(0.25), F#4(0.25), F#4(0.25), F4(0.25),

E4(0.25), B4(0.25), E4(0.25), E4(0.5), R(0.7)

You just accomplished a very basic technique for assembling musical signals. Most electronic instruments use more elaborate ways of generating sounds, but at their core there is usually something akin to the above.

Notes:

The sound of a sinusoid has a relatively "poor" timbre. You can obtain something a bit more interesting with `sound(1.5*x)`, as this scales the samples to numeric values outside the range $[-1, 1]$, but `sound` will clip (saturate) those values back to $[-1, 1]$ before playing. This means that what you hear will effectively be generated by a non-sinusoidal signal, whose timbre is more complex than that of a sinusoid. We will look at periodic non-sinusoidal signals in a later section of this guide.

You may use `plot` or the Signal Analyzer to look at the music signal. Notice how each sinusoid starts/ends smoothly when transitioning to silence or to another sinusoid. If transitions were abrupt you would hear annoying "clicks" between notes. Consecutive notes with the same frequency would also sound as a single note with longer duration, rather than two distinct notes.

The initial and final silence gaps are inserted to separate the "popping" sound made by some computer sound cards when switching on or off from the actual melody.

You can create a polyphonic signal by generating signals for individual voices as outlined above, and then playing their weighted sum using the `sound` command.

Unit impulse and unit step

You may define the unit step, or Heaviside function, in a very simple way as an inline function in Matlab

```
u = @(t) (t >= 0);
```

Generate a time variable spanning an interval of 4 s. Use `plot(t,u(t-t0))` to visualize the (shifted) rectangle for different choices of the offset `t0`.

Q1.4 Function `u` can handle any affine transformation of the time variable. Yet, $u(at + b)$ with arbitrary a and b can simply be written as $u(\pm t - t_0)$ with suitable t_0. Justify this and find the apropriate values of factor $\pm 1$and t_0 given a and b. Check your results experimentally for different choices of a and b.

The unit impulse $\delta(t)$ is formally the derivative of the step function $u(t)$, which is infinite at $t = 0$ and zero elsewhere. Because directly working with such a function in practice is problematic, it is common to define a differentiable approximation to the step function $u_\Delta(t)$ and the approximate impulse $\delta_\Delta(t)$ as its derivative.

```
s = @(t) (t >= 0).*t;                    % Unit ramp
uD = @(t,D) (1/D)*(s(t+D/2)-s(t-D/2));  % Approximate unit step
duD = @(t,D,h) (uD(t+h/2,D)-uD(t-h/2,D))/h; % Numeric derivative
deltaD = @(t,D) duD(t,D,1e-6);           % Approximate unit impulse
```

The parameter Δ defines the width of the region around $t = 0$ where u_Δ transitions from 0 to 1, and where δ_Δ has nonzero support. Generate a time variable spanning a symmetric interval around zero and check `plot(t,deltaD(t-t0,D))` for different choices of `D` and `t0`.
Visualise the approximate impulse `deltaD(a*t,D)` for different values of the scaling parameter `a`. Explain why these results are consistent with the following known property of the ideal impulse

$$\delta(at) = \frac{1}{|a|}\delta(t)$$

2 Systems

Consider the continuous-time system whose input-output mapping is defined by

$$y(t) = x(t) + 0.7x(t - 0.4)$$

where x and y denote the input and output, respectively. This is implemented by the following function

```
system1 = @(x) filter([1 zeros(1,round(0.4*samplerate) - 1) 0.7],1,x);
```

Obtain the output of the system for $x(t) = u(t) - u(t - 1.5)$ with a symmetric time variable spanning 8 s as

```
t = (-4:1/samplerate:4)';
x = u(t) - u(t-1.5);
y = system1(x1);
plot(t,y); grid on
```

Check that the input and output are as expected. Similarly, check the response of this system to the approximate unit impulse $\delta_\Delta(t)$.

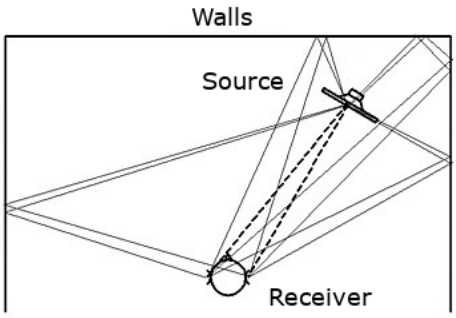
Q2.1 Read the following clip of a speech signal

```
[x,samplerate] = audioread('echoin.wav');
```

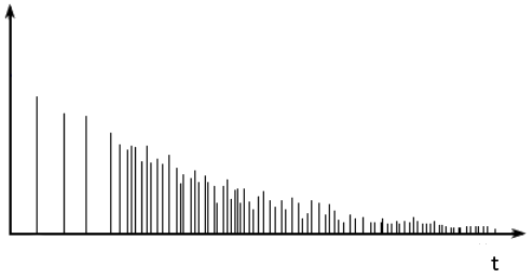
Apply it to `system1` and listen to the output. Explain what you heard.

Application: Acoustic fingerprinting

For linear and time-invariant (LTI) systems, the impulse response fully characterises the input-output mapping. Specifically, the output of the system to any input is obtained by convolving it with the impulse response. This section illustrates the process with a simple application where a room imparts an "acoustic fingerprint" on voice signals that propagate between a source and a receiver.
Consider the figure that shows a room inside which an acoustic source and a receiver are positioned. Sound propagates over multiple reflected paths on the walls, floor and ceiling, creating a pattern of echos whose attenuations and delays increase as the path length and number of reflections increase.



This translates into an impulse response between the source and receiver that is formed by a set of scaled impulses, each having a particular delay and magnitude that is associated with a specific propagation path. The following figure illustrates such an impulse response for a strongly reverberating room.



Q2.2 Explain the general features of the figure. Why are there fewer and stronger impulses for small delay? Why do amplitudes show a decreasing trend as the delay increases?

The supplied function `rir.m` computes the impulse response of a room given the coordinates of a source and receiver. Start by generating and visualizing a sample room impulse response as follows:

```
mic = [1 2 1];
n = 12;
r = 0.7;
rm = [10 10 3];
src = [5 5 1];
h = rir(samplerate, mic, n, r, rm, src);
```

Vectors `rm`, `src` and `mic` define the size of the room and the 3D positions of the source and receiver, respectively. Variable `r` defines the reflectivity of the walls. Repeat for different positions of the source, receiver, and reflectivity, and see how the impulse response changes.
Load the previously used voice clip, take it as the source signal, create the received signal, and listen to it

```
[x,samplerate] = audioread('echoin.wav');
y = conv(h,x);
soundsc(y,samplerate)
```

Repeat the test for various positions and reflectivities and check if you can perceive any differences.

Q2.3 Now load another sound clip containing the input signal `x` and the output `y` after propagation over a large hall simulated by `rir`

```
load fingerprint.mat
```

The receiver is still located at `mic = [1 2 1]`, but now the room dimensions are `rm = [20 15 10]`. The walls are either made from wood (`r = 0.6`) or concrete (`r = 0.9`). The source is located at the same height of the receiver (1 m), but its horizontal coordinates are unknown. Start by playing `x` and `y`. Based on your own simulations of acoustic propagation in this room driven by `x`, and taking the parameter `n = 12` as before, is it more likely that (i) the walls are made of wood or concrete? (ii) the source is at a distance of 3 m or 20 m from the receiver?

Q2.4 Now generate a sinusoidal signal $x_1(t)$ according to the the specifications given in the lab class and calculate the output $y_1(t)$ from `system1`. Using the decomposition of a sinusoid as a sum of complex exponentials, show that $y_1(t)$ is also a sinusoid of the form $y_1(t) = A\cos(\omega t + \phi)$ and determine the values of $A \in [0, +\infty[$, $\omega \in \mathbb{R}$, and $\phi \in]-\pi, \pi]$. Keep in mind that the arguments of standard trigonometric functions are given in **radians**.

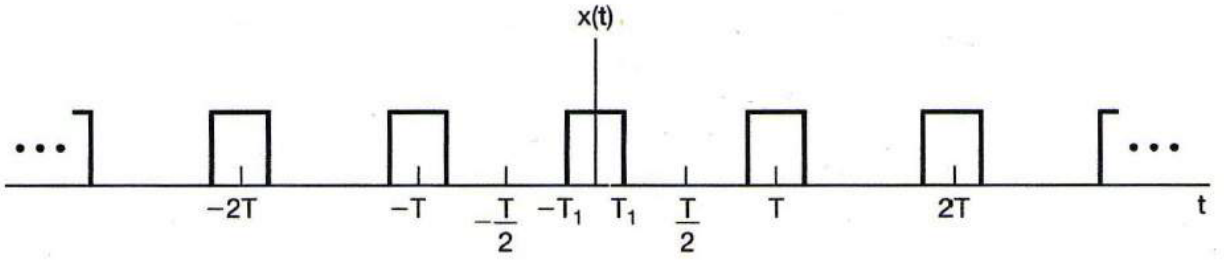
Invoke `system1` to obtain the numeric response to `x1`. Plot it jointly with the sinusoid whose parameters you determined above and check that they match. Keep in mind that, because you are working with a limited interval of the (infinite) time axis, the generated input is effectively taken as zero outside that interval. At the edges of the time axis you will see transient effects in the output of `system1` due to the temporal misalignment of the input signal and its delayed and scaled replica internally produced by the system.

Q2.5 Now you should characterize a "mystery" *discrete-time* system `system2` whose internal details are not available to you. Viewing it as a black box, you should design (multiple) inputs `x`, defined over suitable dicrete-time spans `n` (akin to `t` above for continuous-time signals and systems), observe the outputs `y = system2(x,n)`, and from those classify the system according to linearity, time invariance, memory, causality, stability, and invertibility. The function assumes that the input signal is zero outside the given discrete-time range.

Describe the experiments, the input signals, the results, and your conclusions. For each property, were you able to reach a definitive yes/no conclusion? Document only the experiments that were relevant to draw conclusions.

3 Fourier Series

Consider the periodic signal (partially) represented in the figure



This signal may be represented as the Fourier series

$$\tilde{x}(t) = \sum_{k=-\infty}^{+\infty} a_k e^{jk\omega_0 t}$$

where

$$a_k = 2\frac{T_1}{T}\frac{\sin(k\omega_0 T_1)}{k\omega_0 T_1},\ \omega_0 = \frac{2\pi}{T}$$

and $T = 1$ and $T_1 = 1/4$.
We will consider approximations to $\tilde{x}$ given by the truncated series

$$\tilde{x}_N(t) = \sum_{k=-N}^{N} a_k e^{jk\omega_0 t}$$

for different values of N.
Using ω_0 and a_k obtained previously build $\tilde{x}_N(t)$ in a symmetric time interval spanning 10 s for, at least, $N = 1$, 2, 3, 4, and 5. Since $\tilde{x}(t)$ is real, you may alternatively express $\tilde{x}_N(t)$ as a sum of sinusoidal terms with real coefficients

$$\tilde{x}_N(t) = A_0 + \sum_{k=1}^{N} A_k \cos(k\omega_0 t + \varphi_k)$$

where $A_0 = a_0$ and, for $k \geq 1$, $A_k = 2|a_k|$, $\varphi_k = \arg a_k$.

Q3.1 Create a superimposed plot of all the $\tilde{x}_N(t)$. Zoom in and observe the details of the waveforms. Do they behave as expected? Comment on the results. The following code sample demonstrates how the approximation could be compactly computed for a 100 Hz unipolar square wave

```
t = (-320:320)'/samplerate;
w0 = 2*pi*100;
T = 2*pi/w0;
ak = @(k) sin(k*w0*T/4)./(k*pi).*exp(-1i*k*w0*T/4);
a0 = 1/2;   % Segregate a0, which often has a different expression

N = 100;    % Order of the Fourier expansion
xN = exp(1i*w0*t*(-N:N))*[ak(-N:-1) a0 ak(1:N)].';
plot(t,real(xN)); grid
```

Now visualize the spectra of these signals using the Fourier transform. You may invoke `freqz`, which computes the Fourier Transform in discrete time, and scale the magnitude by `1/samplerate` to match the continuous-time Fourier transform. A wrapper function `cftransform` is provided for convenience

```
[XN,f] = cftransform(xN,find(t == 0),1e4,samplerate);
plot(f,abs(XN))
```

Note that, in general, `X` will be complex. When plotting it, you may also want to examine `angle`, `real`, or `imag`. There are more efficient ways of obtaining a Fourier transform than using `freqz`, at the cost of some additional technicalities that are not warranted here.

Q3.2 Visualize the spectra of the signals $\tilde{x}_N(t)$ created above, zooming in as appropriate. Do these have the structure that you expected?

Consider now the periodic signal provided to you in the lab session, which can once again be represented by the Fourier Series

$$\tilde{x}(t) = \sum_{k=-\infty}^{+\infty} a_k e^{jk\omega_0 t}$$

Q3.3 Find ω_0 and the general expression for the Fourier coefficients a_k for this new signal. You may obtain a_k directly from its definition, or using known signal-transform pairs and properties of the Fourier series. As before, create a superimposed plot of truncated series $\tilde{x}_N(t)$ with increasing N, as a means to check your computation of the Fourier coefficients.

4 Frequency response

The frequency response of an LTI system, $H(j\omega)$, may be interpreted as the gain that the system provides for a complex exponential $e^{j\omega t}$ at its input, yielding an output $H(j\omega)e^{j\omega t}$.
Conceptually, one could measure $H(j\omega)$ for a set of frequencies $\omega_1, \omega_2, \ldots$ by applying complex exponentials for each of these frequencies and measuring the gain. While this could be feasible in simulation, it may not be done for systems in the physical world, which act on real inputs to produce real outputs. Fortunately, LTI physical systems, including those described by linear constant-coefficient ordinary differential equations, have conjugate symmetric frequency responses that enable an alternative approach. When $H(j\omega) = H^*(-j\omega)$ for all ω, the response to a sinusoidal input $x(t) = \cos(\omega t)$ will be $y(t) = A(\omega)\cos(\omega t + \varphi(\omega))$, where $A(\omega) = |H(j\omega)|$ and $\varphi(\omega) = \angle H(j\omega)$.

Q4.1 If you measure the response $y(t)$ of a real-world LTI system to a sinusoidal input $x(t) = \cos(\omega t)$, explain how you can obtain $A(\omega)$ and $\varphi(\omega)$.

The function `system3` implements a system whose frequency response magnitude you should determine experimentally. Generate a time variable spanning 20 s and repeatedly go through the following steps:

- Generate a sinusoidal signal `x` with a chosen frequency ω
- Obtain the output `y = system3(x,samplerate)` and plot it. Because the function simulates processing of a continuous-time signal but operates on a discrete-time sampled version, it requires the sampling rate as one of its input parameters
- Obtain whatever measurements you need from `y`. Gridlines, zoom and data tips, available in the figure window, as well as the `ginput` function, are useful for this.
- Compute $|H(j\omega)|$ from those measurements

Keep in mind that there might be transient effects in the response at the edges of the time variable because you are applying time-truncated sinusoids to the system. Make sure that you take measurements inside a subinterval that does not contain such transients.

Q4.2 Compute $|H(j\omega)|$ of `system3` for $\omega \in \{1, 10, 50, 100, 200, 1000\}$ rad/s using the approach outlined above. Sketch $|H(j\omega)|$ and comment on the type of filtering (lowpass, highpass, bandpass) and whether the filtering response is ideal or not.

Alternative methods of obtaining the frequency response may first measure the impulse response and then compute its Fourier transform. This may be done by directly applying an impulse at the input, which is perfectly viable in simulation, but hardly feasible in practice as it might damage the system. More realistically, the impulse response could be evaluated (and often is) by computing the crosscorrelation between the input and output to a white noise stimulus, or by differentiating the response to a unit step.

Q4.3 Apply the approximate impulse `deltaD` to the input of `system3` to obtain (a good approximation of) the impulse response. This requires the sampling step (`1/samplerate`) to be at least 10 times smaller than the width of the impulse, otherwise the approximation will be too coarse. For example,

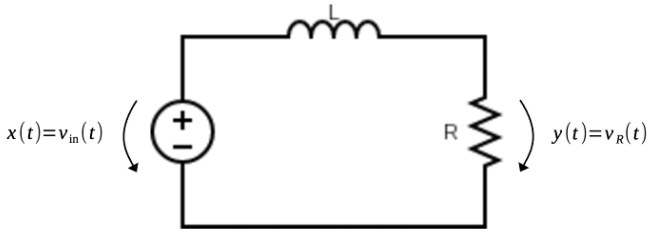
```
t = (-0.2:1/samplerate:1)';
x = deltaD(t,10/samplerate);
y = system3(x,samplerate);
```

Compute its Fourier transform as outlined above, zoom into the region $\omega \in [-0, 1000]$ rad/s, and confirm the previously obtained values of $|H(j\omega)|$. Also compute the time constant of this (first-order) system in *two different ways* directly from the impulse response.

Recheck your impulse response in an alternative way by taking the approximate derivative of the response to the unit step

```
ys = system3(u(t),samplerate);
% Append 0 to "fix" length because diff knocks off one element
y = [diff(ys); 0]*samplerate;
```

The given function `system3` implements the input-output mapping for the circuit depicted in the figure, where $x(t)$ and $y(t)$ denote the input and output voltages, respectively.



Q4.4 Obtain the frequency response of this circuit, $H(j\omega)$, as a function of R and L. You can do this by writing the differential equation, specifying the input and output as $x(t) = e^{j\omega t}$, $y(t) = H(j\omega)e^{j\omega t}$, and then solving for $H(j\omega)$. Alternatively, directly apply the Fourier transform to the differential equation, and use its known properties to factor $X(j\omega)$ and $Y(j\omega)$, such that $H(j\omega) = Y(j\omega)/X(j\omega)$.

Q4.5 For $R = 2\,\Omega$ determine the value of L based on you experimental results. How did you check that this agrees with the measurements of both $H(j\omega)$ and $h(t)$?

★ Go back here

5 Filtering and Sampling I

The following functions implement different types of filtering:

- `system4` is a lowpass filter with cutoff frequency of 500 Hz
- `system5` is a highpass filter with cutoff frequency of 400 Hz
- `system6` is a bandpass filter with passband from 300 Hz to 900 Hz, approximately

```
system4 = @(x) lowpass(x,5e2,samplerate,Steepness=0.99);
system5 = @(x) highpass(x,4e2,samplerate,Steepness=0.75);
system6 = @(x) bandpass(x,[3e2 9e2],samplerate,Steepness=[0.67 0.98]);
```

Q5.1 Similarly to what you did previously, create a linear chirp signal that will be used as an input to the filters

```
t = (0:1/samplerate:2)';
x = chirp(t,0,2,2e3);
```

Then, filter `x` through the three different filters to obtain `y4`, `y5`, `y6`, plot them and explain the results

```
y4 = system4(x); y5 = system5(x); y6 = system6(x);

subplot(3,1,1); plot(t,y4); grid
subplot(3,1,2); plot(t,y5); grid
subplot(3,1,3); plot(t,y6); grid
```

Implement an alternative to bandpass `filter6` as a cascade (series connection) of lowpass/highpass filters or as a parallel connection of lowpass/highpass filters.

Q5.2 Load the file containing the signal x_{noisy}

```
load noisy_signal.mat
```

According to the spectral content of x_{noisy}, determine the passband values for a bandpass filter `filter7` that can be applied to x_{noisy} to attenuate its noisy content. To obtain the impulse response of `filter7`, say, `h7`, start from the ideal lowpass spectrum in continuous time

$$h(t) = \frac{\sin(Wt)}{\pi t} = \text{sinc}\Big(\frac{W}{\pi}t\Big) \longleftrightarrow H(j\omega) = \begin{cases} 1, & |\omega| \le W \\ 0, & \text{otherwise} \end{cases}$$

Choose W appropriately and use the modulation property to place the passband in the appropriate interval of (positive and negative) frequencies. To numerically compute `h7` choose a symmetric time interval around zero, e.g.,

```
th = 1/samplerate*(-300:300)';
h7 = ...   % Complete this
```

Finally, test your bandpass filter on the noisy signal as

```
% Compensate for filter delay with circshift
y7 = circshift(filter(h7/samplerate,1,x_noisy),1-find(th == 0));
```

↳ Understand this

and get the frequency response as

```
[H7,f] = cftransform(h7,find(th == 0),1e3,samplerate);
```

In the argument of `filter` the "continuous-time" impulse response `h7` is divided by `samplerate` to create a discrete-time impulse response whose frequency response still has unit gain in the passband (this is what is needed for equivalent discrete-time processing of continuous-time signals, and will be clarified when you tackle sampling).

Plot the impulse response, the frequency response, and compare the original and filtered signals in both the time and frequency domains. Comment on your results. Can you make an informed guess about the shape of the original signal?

Sampling I

To simulate the sampling process proceed as follows:

- Define `samplerate` as a fine temporal scale. "Continuous-time" signals will actually be simulated as discrete-time signals generated for this scale
- Define the "original continuous-time signal" `xc`
- Define a sampling frequency `fs` which is an integer fraction of `samplerate`, i.e., `fs = samplerate/M`, where the integer `M` is often referred to as the decimation factor
- Generate the "sampled signal" `xd` using the `downsample` function
- If you process `xd` to create some `yd` you can perform the steps required to reconstruct a "continuous-time" signal `yc` from it at the fine temporal scale using the `resample` function

The following code simulates sampling of a continuous-time sinusoid with 5 Hz after sampling with a rate of 100 Hz

```
samplerate = 44.1e3;
t = (0:1/samplerate:4)';
xc = cos(2*pi*5*t);
M = 441;    % fs = samplerate/M = 100 Hz
xd = downsample(xc,M);
subplot(2,1,1); plot(t,xc)
subplot(2,1,2); stem(xd)
```

Note above how `stem` is commonly used instead of `plot` to represent discrete-time signals. Now reconstruct the "continuous-time" signal

```
yc = resample(xd,M,1);  % yd = xd
yc = resize(yc,numel(xc));
plot(t,[xc yc])
```

Q5.3 Plot `xc`, `yc`, and `xc-yc` and comment on the results.

Repeat for decimation factor $M = 6300$, i.e., a sampling frequency of 7 Hz. Note that, while `xc` and `yc` are now quite different, their values coincide on time instants that are integer multiples of the sample interval. Measure the frequencies of `xc` and `yc`; are they related to the Nyquist frequency in the way that you expected?

6 Sampling II

Now create and visualise, in the time and frequency domains, the following "continuous-time" signal (don't worry about the details of how `xc1` is generated)

```
t = (-2^15:2^15-1)'/samplerate;
W = 2*pi*100;

xc1 = imag(hilbert((W/pi)*sinc(W/pi*t)));
[Xc1,f] = cftransform(xc1,find(t == 0),1e4,samplerate,'whole');

subplot(2,1,1); plot(t,xc1); grid on
subplot(2,1,2); plot(f,imag(Xc1)); grid on
```

Q6.1 Downsample `xc1` by `M = 256`, reconstruct as before, and compare with the original signal

```
M = 256;
yc1 = resample(downsample(xc1,M),M,1);
Yc1 = cftransform(yc1,find(t == 0),f,samplerate);

subplot(2,1,1); plot(t,[xc1 yc1]); grid on
subplot(2,1,2); plot(f,imag([Xc1 Yc1])); grid on
```

Why is the real part of both spectra essentially zero? Explain how the spectrum of `yc1` resulted from that of `xc1`, including how their effective maximum frequencies are related.

Q6.2 In previous sections you plotted the Fourier transform of continuous-time signals by using `freqz`, inside the wrapper `cftransform`, on sampled versions (i.e., in discrete time) and dividing the output by `samplerate`. Justify this procedure and clarify under which conditions it holds.

Anti-aliasing prefiltering

Start by loading and playing a music clip

```
[xc,samplerate] = audioread('antialias.wav');
soundsc(xc,samplerate)
```

and visualizing its spectrum

```
[Xc,f] = cftransform(xc,1,1e3,samplerate);
plot(f,abs(Xc))
```

Similarly to what you have done above, downsample by a large factor and listen to the result (keep in mind the "hidden" processing done by the computer and sound card as the discrete-time signal is converted to a true continuous-time acoustic signal played by the speakers)

```
M = 8;     % Decimation factor
xd = downsample(xc,M);
soundsc(xd,samplerate/M)
```

We will now preprocess the original music signal with an anti-aliasing filter so that downsampling becomes less damaging

```
N = 100;    % Anti-aliasing filter order
aa = fir1(N,1/M);
yc = filter(aa,1,xc);
% Check original and filtered spectra
[Xc,f] = cftransform(xc,1,1e3,samplerate);
Yc = cftransform(yc,1,f,samplerate);
plot(f,abs([Xc Yc]))
```

Listen to `xc` (again) and `yc`. Now downsample as before and play

```
yd = downsample(yc,M);
soundsc(yd,samplerate/M)
```

Q6.3 Compare and explain the differences in perceived quality of the downsampled music signals with and without anti-alias preprocessing.

Aliasing paradox

Read and play another music clip

```
[xc,samplerate] = audioread('paradox.wav');
soundsc(xc,samplerate)
```

Now downsample it by a factor of 3 (don't use an anti-aliasing filter) and play it again.

```
M = 3;
xd = downsample(xc,M);
soundsc(xd,samplerate/M)
```

Aliasing destroys spectral components of signals and normally leads to a degradation in quality. Why, then, does the quality of this particular music clip *improve* after downsampling? Does the same conclusion hold if you downsample by other factors, say, 2 or 4?