Laboratory 1

Moodle quiz: 4/15/24 – 4/22/24

Goal

Explore signals with Matlab or Octave.

1 Software Installation

Matlab is proprietary software developed by The MathWorks, Inc., see `www.mathworks.com`. In contrast, Octave is open-source software. Octave is almost compatible with Matlab, i.e. most Matlab programs run unchanged in Octave and vice versa. You can download Octave for free at:

`https://octave.org`

Independent of whether you use Matlab or Octave, install the `control` and `signal` packages. Installation instructions for Octave packages from Forge can be found on the webpage above.

In this course we use Matlab/Octave, which is widely used throughout academia and industry. If you happen to be a Python aficionado, you may appreciate that the packages `numpy`, `matplotlib`, and `scipy` offer analogous functionalities.

2 Matlab/Octave Primer for Signal Processing

This section introduces scripts and functions to get you started. Play with the scripts and solve the exercises to acquire proficiency at Matlab/Octave as a valuable tool for studying signals and systems. As a rule of thumb, all objects in Matlab/Octave are arrays. Recall from linear algebra that matrices are two-dimensional arrays. One-dimensional arrays store row vectors and column vectors, and scalars are 1×1 matrices or zero-dimensional arrays. Also beware that Matlab/Octave arrays start at index 1 rather than 0.

2.1 Graphs and Sounds

This script plots a cosine signal as shown in Fig. 1:

```
Ts = 1e-3;                % sampling period = 1 ms
t = [0:Ts:1];             % sample times
f0 = 10;                  % cosine frequency in Hz
x = cos(2*pi*f0*t);       % DT cosine signal
plot(t, x);
xlabel('t [s]'), ylabel('cos(2 \pi f_0 t)');
```

Array `x` is a row vector of DT samples $x[n] = x(t)|_{t=nT_s}$ where $x(t) = \cos(2\pi f_0 t)$. The `plot` function applies a linear interpolation between the DT samples, s.t. the curve in Fig. 1 looks like a CT signal. If you zoom far enough into the graph, you can see the straight line segments of the linear interpolation.

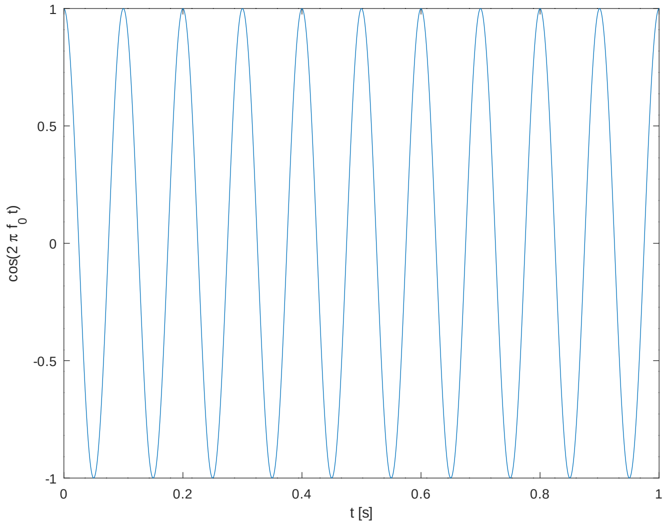

Figure 1: Graph of cosine signal for $f_0 = 10\,\text{Hz}$.

To emphasize the DT character of signal `x`, replace the `plot` function with the `stem` function:

```
stem(t, x);
```

or choose a visible marker for the sample values:

```
plot(t, x, 'o-');
```

We can listen to a sinusoidal signal, if we choose an audible frequency and a proper sampling rate. This script should produce an audible sound of a pure 440 Hz tone:

```
fs = 44100;                % sampling rate = 44100 samples per second
t = [0:1/fs:3];            % 3 seconds of sound
f0 = 440;                  % sine frequency = 440 Hz
x = sin(2*pi*f0*t);        % DT sine signal
sound(x, fs);
```

The quality of the sound depends on your speakers or headphones. Good speakers transmit a wider range of low and high frequencies than bad ones. Change frequency f_0 to explore the range of tones your speakers can produce.

Exercises

1. Zooming into a signal can be controlled precisely by limiting the time interval. For example, command

   ```
   plot(t(1:10), x(1:10))
   ```

plots the first ten samples of signal `x`. Adjust the index range of `t` and `x` to plot one period of `x`.

2. Compare the sound of the 440 Hz sine with

   (a) a 440 Hz cosine,

   (b) the superposition of a 440 Hz sine and a 440 Hz cosine.

   Do you hear the difference? Plot the signals if you want to see the difference.

3. Useful Matlab/Octave functions:

   (a) Try the `help` function on itself:

   ```
   help help
   ```

   (b) Interpret the outputs of these commands:

   ```
   size(t)
   size(x)
   size(f0)
   ```

   (c) You can overlay multiple graphs in one figure. Try highlighting the DT samples of a CT-like interpolated graph:

   ```
   plot(t, x)
   hold on
   plot(t, x, 'o')
   ```

   (d) Generate a PDF file of a graph with a `filename` of your choice:

   ```
   print filename.pdf
   ```

   (e) You can employ multiple figures rather than just one. Use the `help` function to learn about functions `figure`, `gcf`, `clf`, `close`, and `subplot`. Generate signal graphs in two different figures.

2.2 Signals and Functions

The math functions built into Matlab/Octave facilitate the construction of a large variety of signals. For example, here is an exponentially decaying sinusoid using `exp`, see Fig. 2:

```
Ts = 0.01;              % sampling period
t = [0:Ts:10];          % sample times
tau = 1.5;              % time constant of exponential
f0 = 5;                 % sinusoidal frequency
x = exp(-t/tau) .* cos(2*pi*f0*t);
```

The sine cardinal function, aka sinc, is defined as (see `help sinc`):

$$\mathrm{sinc}(x) \;=\; \frac{\sin(\pi x)}{\pi x}\,.$$

Using the built-in `sinc` function liberates us from worrying about a division-by-zero if $x = 0$, see Fig. 3:

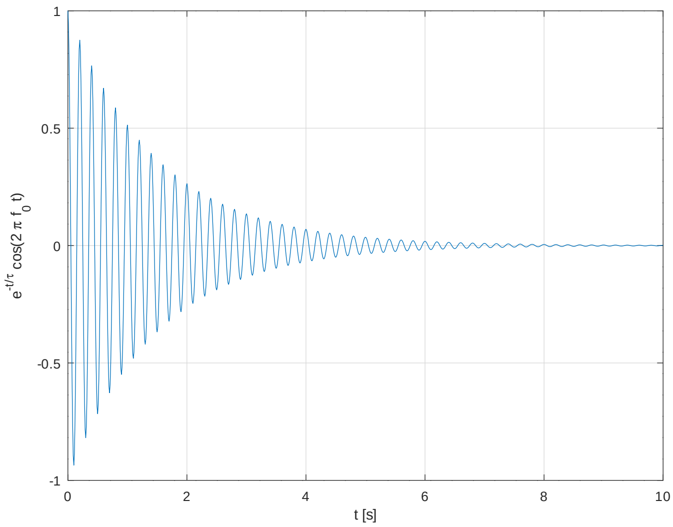

Figure 2: Graph of exponentially decaying cosine for $\tau = 3/2\,\mathrm{s}$ and $f_0 = 5\,\mathrm{Hz}$.

```
Ts = 0.01;              % sampling period
t = [-6:Ts:6];          % sample times
f0 = 1;                 % frequency
y = sinc(2*f0*t);
```

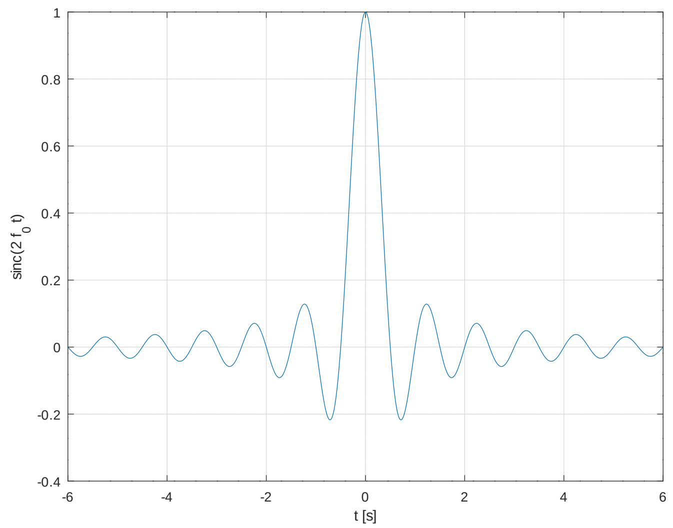

Figure 3: Graph of sinc function for $f_0 = 2\,\mathrm{Hz}$.

Random number generators, such as function `randn`, facilitate the construction of non-deterministic signals with a desired property. For instance, assume we wish to generate bit-valued signals, where about half of the bits are zeros and the other half are ones. There are various ways to accomplish this goal. Each of these two implementations generates a row vector of 20 uniformly distributed random bits, see Fig. 4:

```
b1 = (1 + sign(randn(1,20))) / 2;
b2 = ustep(randn(1,20));
plot(b1, 'o', b2, 'x'), axis([0 21 -0.1 1.1]), pbaspect([4 1]);
```

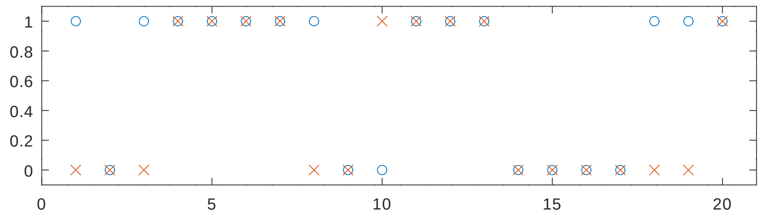

Figure 4: Random bit signals `b1` (○) and `b2` (×).

Unit step function `ustep`, used above to construct signal `b2`, may be implemented as:

```
function y = ustep(n)
  y = (n >= 0);
endfunction
```

Variable name `n` suggests that this is an array of DT time indices, e.g. constructed by

```
n = [-10:10];
```

We may also apply `ustep` to a noninteger (CT) time array as used for the sinc function above:

```
t = [-6:Ts:6];
```

where `Ts` is some real value. The construction of `b2` observes that the `randn` function returns real values, about half of which are negative and the other half is positive. Then, the `ustep` function maps all negative values to 0 and all nonnegative values to 1.

The unit impulse may be implemented with a function shown here:

```
function y = uimpulse(n)
  y = (n == 0);
endfunction
```

As for function `ustep`, variable `n` may be an array of integer (DT) or real (CT) values. **Caution:** Make sure that array `n` includes value 0. This requirement is easily overlooked when contructing `n` with CT time samples such as:

```
Ts = 0.03;
n = [-1:Ts:1];
```

If value 0 is missing in array `n`, function `uimpulse` returns an array of zeros rather than the expected impulse. If in doubt, compute `sum(n==0)` which returns 1 if array `n` contains value 0, otherwise it returns 0.

The third basic signal is the unit ramp, which can be implemented with the aid of `ustep`:

```
function y = uramp(n)
  y = n .* ustep(n);
endfunction
```

Fig. 5 shows the three basic signals `uimpulse`, `ustep`, and `uramp`.

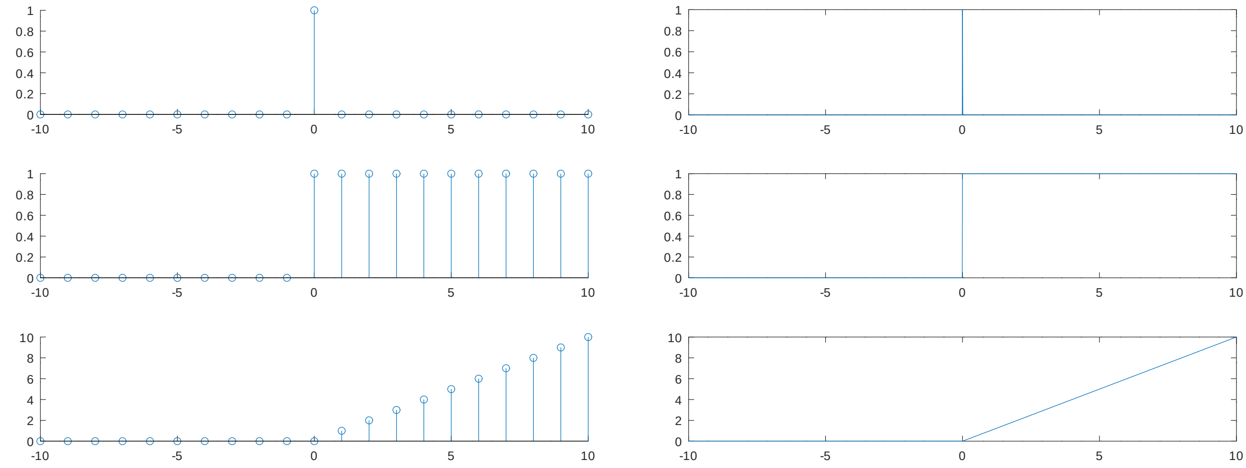

Figure 5: Graphs of basic signals as DT stem plots with `n = [-10:10]` (left) and CT plots with `t = [-10:0.01:10]` (right).

Exercises

1. Implement and plot the following signals.

   (a) Squared sine: $x_1(t) = \sin^2(2\pi f_0 t)$

   (b) Raised cosine: $x_2(t) = \dfrac{1}{2}\big(1 + \cos(2\pi f_0 t)\big)$

   (c) Linear chirp: $x_3(t) = \cos(2\pi f(t)t)$, where $f(t) = t/10$.

   A chirp is a sinusoid with time-varying frequency $f(t)$. A linear chirp has frequency function $f(t) = at + b$.

2. Implement and plot these random signals.

   (a) Random bits with about one quarter 1-bits.

   (b) Uniformly distributed random values ± 3.

3. Use functions `ustep` and `uramp` to construct these piecewise linear signals:

   (a) Rectangular pulse: $x_1(t) = \begin{cases} 1\,, & -1 \le t < 1\,, \\ 0\,, & \text{otherwise} \end{cases}$

   (b) Triangular pulse: $x_2(t) = \begin{cases} t\,, & 0 \le t < 1\,, \\ 2 - t\,, & 1 \le t < 2\,, \\ 0\,, & \text{otherwise} \end{cases}$

   (c) Sawtooth pulse: $x_3(t) = \begin{cases} t\,, & 0 \le t < 1\,, \\ 0\,, & \text{otherwise} \end{cases}$

4. The signal library offers several functions for constructing periodic waveforms with various shapes.

(a) Use function `square` to generate a rectangular wave $\tilde{x}_1(t)$ with period $T = 2\,\mathrm{s}$ that is the periodic extension of rectangular pulse

$$x_1(t) \;=\; \begin{cases} 1\,, & 0 \le t < 1\,, \\ 0\,, & \text{otherwise} \end{cases}$$

(b) Use function `sawtooth` to generate a triangular wave $\tilde{x}_2(t)$ with period $T = 2\,\mathrm{s}$ that is the periodic extension of triangular pulse

$$x_2(t) \;=\; \begin{cases} t\,, & 0 \le t < 1\,, \\ 2 - t\,, & 1 \le t < 2\,, \\ 0\,, & \text{otherwise} \end{cases}$$

5. We study the signal transformations time shifting, time reversal, and time scaling.

(a) Consider a delay system $D_2(s)(t) = s(t-2)$ and a time-reversal system $R(s)(t) = s(-t)$ for all signals $s \in [\mathbb{R} \to \mathbb{R}]$. We wish to know whether D_2 and R commute, i.e. whether the outputs of the series compositions satisfy $z(t) = v(t)$:

$$x \longrightarrow \boxed{\;D_2\;} \xrightarrow{\;y\;} \boxed{\;R\;} \longrightarrow z$$

$$x \longrightarrow \boxed{\;R\;} \xrightarrow{\;w\;} \boxed{\;D_2\;} \longrightarrow v$$

Given input signal

$$x(t) \;=\; \begin{cases} \frac{3}{2}t\,, & 0 \le t < 2\,, \\ 3\,, & 2 \le t < 3\,, \\ 0\,, & \text{otherwise} \end{cases}$$

plot signals $x(t)$, $y(t)$, $z(t)$, $w(t)$, and $v(t)$, and verify or disprove $z(t) = v(t)$.

(b) Plot the time-scaled signals $y(t) = x(2t)$ and $z(t) = x(t/2)$, with $x(t)$ given in part (a). Which of the signals $y(t)$ or $z(t)$ compresses and which expands $x(t)$ in time?