**Stanford University**
**EE 102A: Signal Processing and Linear Systems I**
**Summer 2022**
**Instructor: Ethan M. Liang**

**Homework 2, due Friday, July 15**

*Signal Periodicity*

1. Determine whether each DT signal is periodic and, if so, find its period.

   a. $x[n] = \cos\left(\sqrt{3}n\right)$.

   b. $x[n] = \cos\left(\frac{3\pi}{5}n\right) + \sin\left(\frac{4\pi}{5}n\right)$.

   c. $x[n] = \sin\left(\frac{\pi}{6}n^2\right)$. This is a DT chirped sinusoid. *Hint*: although it may appear that the signal oscillates faster as $|n|$ increases, frequencies differing by multiples of $2\pi$ are indistinguishable.

*Operations on Signals*

2. Consider the CT unit rectangular pulse signal

$$\Pi(t) = \begin{cases} 1 & |t| \le \frac{1}{2} \\ 0 & |t| > \frac{1}{2} \end{cases}.$$

   Sketch the following signals.

   a. $\Pi(t/2)$.

   b. $\Pi(2t-1)$.

   c. $\Pi(-t+1)$.

3. Consider the CT unit triangular pulse signal

$$\Lambda(t) = \begin{cases} 1-|t| & |t| \le 1 \\ 0 & |t| > 1 \end{cases},$$

   and define a signal

$$x(t) = \Lambda(t-1) - \Lambda(t+1).$$

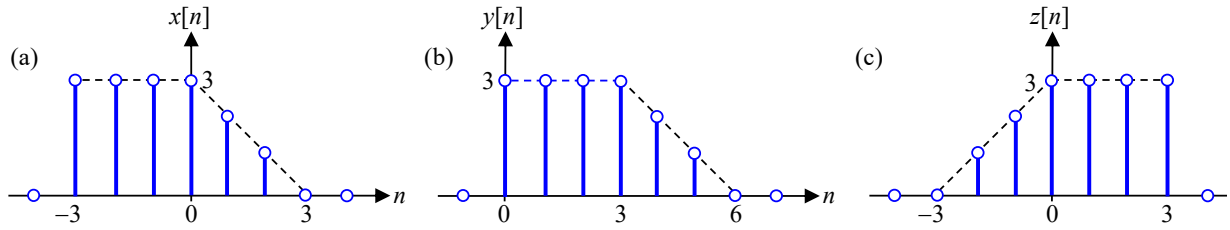
   Sketch the following signals.

   a. $x(t)$.

b. $\dfrac{dx(t)}{dt}$.

c. $\dfrac{d^2x(t)}{dt^2}$.

4. Express each DT signal as a sum (not a product) of scaled, shifted unit step functions $u[n]$ and unit ramp functions $r[n]=n\cdot u[n]$. *Hint*: once you have done part (a), you can use your result to do parts (b) and (c) with minimal effort.




*CT Impulse Functions*

5. Use the sampling property of the CT impulse to simplify each expression.

   a. $\cos(\pi t)\left[\delta(t)+\delta(t-1)\right]$. *Hint*: your answer should involve impulse functions.

   b. $\int_{-\infty}^{\infty}\cos(\pi t)\left[\delta(t)+\delta(t-1)\right]dt$. *Hint*: your answer should be a specific number.

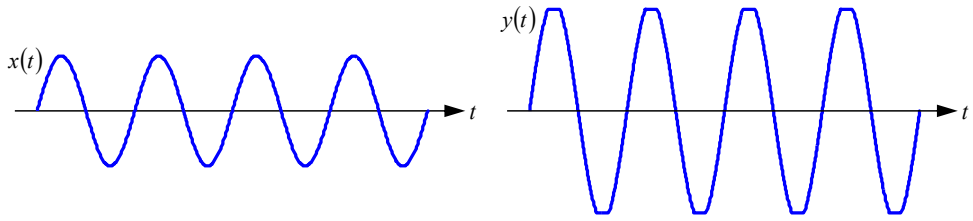
   c. $\int_{-\infty}^{\infty} f(t+1)\delta(t-2)\,dt$. *Hint*: your answer should be $f(t)$ evaluated at a specific value of *t*.

   d. $\int_{0}^{\infty} f(t)\left[\delta(t-1)+\delta(t+1)\right]dt$. *Hint*: note the limits of integration.

*System Classification*

6. Given each system *H* with input *x* and output *y*, state whether the system is: (i) linear and (ii) time-invariant. Briefly justify your answer.

   a. CT amplifier clips at the power supply rails without otherwise distorting the signal. No justification is required here.




b. CT system records the input signal, then plays it backward: $H\left[x(t)\right] = y(t) = x(T-t)$.

c. DT system adds a constant to the input signal: $H\{x[n]\} = y[n] = x[n] + C$.

d. DT system multiplies the input signal by an imaginary exponential signal: $H\{x[n]\} = y[n] = e^{j\Omega_0 n} x[n]$

7. *Systems described by constant-coefficient linear equations*. We stated in lecture that a CT (or DT) system governed by constant-coefficient linear differential (or difference) equations is LTI if the system is initially at rest, i.e., it has zero initial conditions. This means that the output $y(t)$ (or $y[n]$) remains zero until the input $x(t)$ (or $x[n]$) first becomes nonzero. Assuming the system is not at rest can render the system non-LTI. As an example, consider a CT first-order lowpass filter with input $x(t)$ and output $y(t)$ described by a differential equation

$$\tau \frac{dy}{dt} + y(t) = x(t).$$

Assume an initial condition at a time $t = 0$: $y(0) = y_0$. For $t > 0$, the output is

$$H\left[x(t)\right] = y(t) = y_0 e^{-t/\tau} + \frac{1}{\tau}\int_0^t e^{-(t-t')/\tau} x(t')\,dt', \quad t > 0.$$

The output at time $t > 0$ comprises two terms. The first is a contribution from the initial condition, which decays exponentially with time. The second is the contribution from inputs at times $0 \le t' \le t$, weighted by a factor $\frac{1}{\tau} e^{-(t'-t)/\tau}$, which gives more weight to newer inputs and less to older inputs.

a. Show that the system is not linear if $y_0 \neq 0$.

b. Show that the system is not time-invariant.

*Representing DT LTI Systems by Convolutions*

8. For each of the following DT LTI systems *H* with input $x[n]$ and output $y[n]$: (i) find an impulse response $h[n]$ such that $H\{x[n]\} = y[n] = x[n] * h[n]$, (ii) sketch $h[n]$, and (iii) state whether the system is causal. (In case we haven't covered causality in lecture yet: a system is causal iff its output

at time $t$ depends only on inputs at times $t' \le t$. An LTI system is causal iff its impulse response $h(t)$ is zero for $t < 0$. See *EE 102A Course Reader*, pages 62-63.)

a. Infinite running summation

$$H_a\{x[n]\} = \sum_{k=-\infty}^{n} x[k]$$

Denote the impulse response by $h_a[n]$.

b. Finite summation

$$H_b\{x[n]\} = \sum_{k=n-2}^{n+2} x[k]$$

Denote the impulse response by $h_b[n]$. *Hint*: one method of solution is to choose an input $x[n] = \delta[n]$, and directly compute the output $y[n] = h_b[n]$. A second method is to think of the finite summation as the difference between two infinite summations with different upper limits, and express $h_b[n]$ as the difference between two shifted unit step functions.

*Properties of Convolution*

9. *Sum or Area of Convolution*. This property is true for both DT and CT convolution:

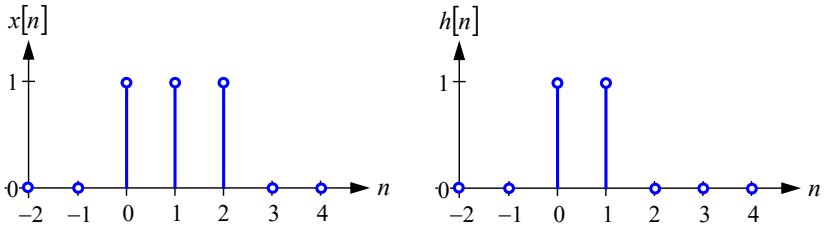
$$\text{If } y[n] = x[n] * h[n], \text{ then } \sum_{n=-\infty}^{\infty} y[n] = \left(\sum_{n=-\infty}^{\infty} x[n]\right)\left(\sum_{n=-\infty}^{\infty} h[n]\right).$$

$$\text{If } y(t) = x(t) * h(t), \text{ then } \int_{-\infty}^{\infty} y(t)\,dt = \left(\int_{-\infty}^{\infty} x(t)\,dt\right)\left(\int_{-\infty}^{\infty} h(t)\,dt\right).$$

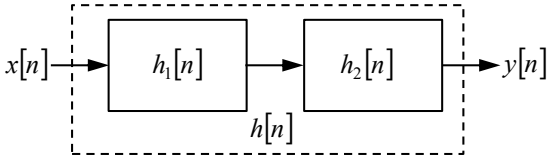
Prove this property for the DT case. *Note*: if $h$ represents the impulse response of an LTI system, the sum or integral of $h$ on the right-hand side is called the *d.c. gain* of the system, as it describes how the sum or area of the output $y$ is scaled relative to the sum or area of the input $x$. Later we will see that the sum or integral of $h$ is equal to the system frequency response at zero frequency.

*Evaluating DT Convolution Sums*

10. Evaluate the convolution $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$ for the two signals shown using the "flip and drag method". *Hint*: first sketch $x[k]$ vs. $k$. Then, for each of $n = -1, 0, 1, 2, 3, 4$, sketch $h[n-k]$ vs. $k$, multiply $x[k]$ by $h[n-k]$ sample-by-sample, and sum over $k$ to obtain $y[n]$. Finally, sketch $y[n]$ vs. $n$.




11. LTI systems with impulse responses $h_1[n]$ and $h_2[n]$ are cascaded to form an LTI system with impulse response $h[n]$.




Both are first-order systems. Their impulse responses are

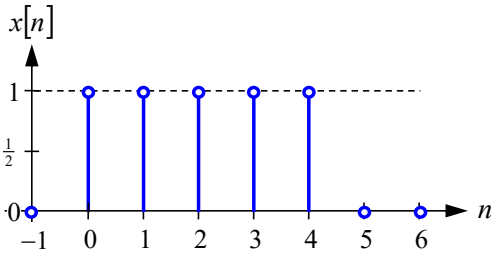
$$h_1[n] = a^n u[n]$$
$$h_2[n] = b^n u[n],$$

where $|a|, |b| < 1$ and $a \neq b$.

a. Find an expression for $h[n] = h_1[n] * h_2[n]$. Simplify your expression for $h[n]$ so it is clearly a linear combination of $h_1[n]$ and $h_2[n]$. *Hint*: you can solve this by summing a geometric series, without using "flip and drag".

b. Verify that $\sum_{n=-\infty}^{\infty} h[n] = \left(\sum_{n=-\infty}^{\infty} h_1[n]\right)\left(\sum_{n=-\infty}^{\infty} h_2[n]\right)$, as expected from Problem 9.

12. A rectangular pulse $x[n]$, shown below, is input to different LTI systems, each specified by an impulse response $h[n]$. For each system: (i) Find an expression for the output $y[n]$ by evaluating the convolution $x[n] * h[n]$. (ii) Sketch the output $y[n]$ without using a calculator or computer. (iii) Verify that $\sum_{n=-\infty}^{\infty} y[n] = \left(\sum_{n=-\infty}^{\infty} x[n]\right)\left(\sum_{n=-\infty}^{\infty} h[n]\right)$, as expected from Problem 9.



a. Two-sample moving average $h[n]=\frac{1}{2}\left(\delta[n]+\delta[n-1]\right)$.

b. Edge detector $h[n]=\frac{1}{2}\left(\delta[n]-\delta[n-1]\right)$.

c. First-order lowpass filter $h[n]=\left(\frac{1}{2}\right)^n u[n]$. In this part, an approximate sketch of $y[n]$ is sufficient. You do not need to evaluate $\sum_{n=-\infty}^{\infty} y[n]$ exactly; it is sufficient for you to estimate it from your sketch. *Hint*: you can express the input as the difference between two shifted step functions. Then, exploiting the linearity and time invariance of the system, you can express the output as the difference between two shifted step responses.

## Laboratory 2

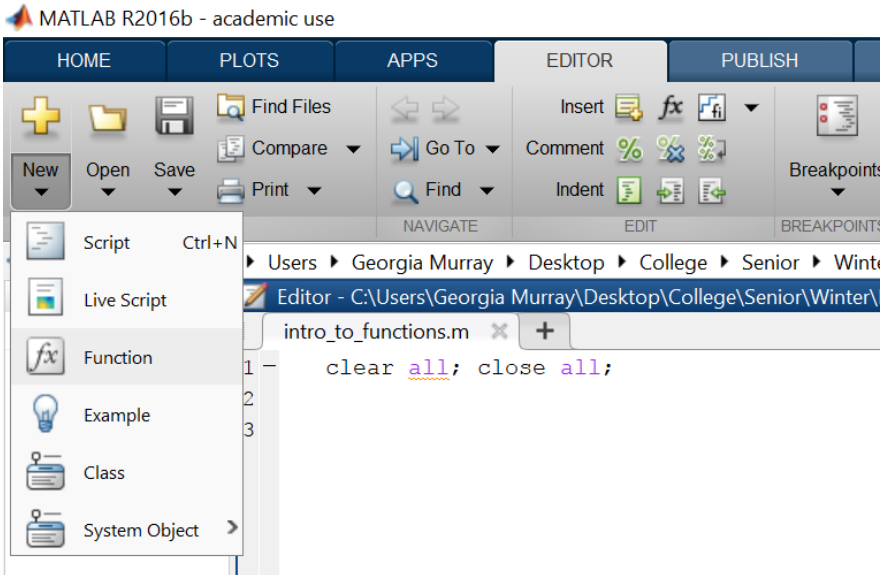
### Introduction to MATLAB Functions

In HW1, we looked at how to use MATLAB *scripts*; this week we'll look at the other main component of MATLAB programing: *functions*. For those with a CS 106A/B background, you can generally think of MATLAB scripts as your run/main function and MATLAB functions as all other methods/functions (this is a general analogy to help you think about it, but they are not exact parallels).

Unlike scripts, functions accept input variables and yield output variables; thus, they are ideal for computations that you will need to do repeatedly on a variety of data. First, let's walk through the basics of how to write a function (this part is not to be turned in). Then we will ask you to write your own functions in the following tasks. Here, we're going to write a function that calculates the mean of an input vector. MATLAB already has a function do this, **`mean`**, so we will create an alternative function, **`avg`**, and use **`mean`** to check that our function works correctly. In the future, you should simply use the built-in **`mean`** function whenever you wish to compute the mean.

Note that functions cannot be run on their own but rather must be called from scripts, other functions, or the command line. So first, let's write a script to call the function.

That done, here's how we create a function:




You should now be in a new window with the following auto-generated text:

```
function [ output_args ] = Untitled2( input_args )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

end
```

Change to the function name and list the desired outputs and inputs as indicated by the template. In our case, that will look like this:

```
function [ mu ] = avg( vec )
% Function to calculate and return the mean value
% of the input vector vec

end
```

Now insert the body of your function into the template below the commented function description and above the key-word **end**. Here are two ways you might do that:

Option 1:

```
total = 0;
for i = 1:length(vec)
     total = total + vec(i);
end
mu = total / length(vec);
```

Option 2:

```
mu = sum(vec) / length(vec);
```

Option 1 is similar to how we might compute the mean in, say, C++ or Java. Option 2 uses MATLAB's extensive library of built-in vector and matrix functions, in this case, **sum** and **length**. We recommend that you use these vector and matrix functions, instead of constructs like FOR loops, whenever possible.

Now to test your function, go back to the script that you created initially and compare the results of the MATLAB-provided function **mean** and your own function **avg**. Below, we provide an example script of how of you might do this. We're intentionally using some built-in MATLAB functions and capabilities that you haven't seen before. Please look these up on the extremely well-documented MATLAB support website (https://www.mathworks.com/help/index.html) until you fully understand how our testing script works. Alternatively, you can use MATLAB's built-in help function, which you can invoke by typing **help** at the command line.

```
clear all; close all;

vec_len = 10; %num of elements in vector to find mean
num_tests = 10; %num of tests you intend to run
success = 0; %num of test cases where your function achieves same as
mean

for n = 1:num_tests
    test = 10*rand(vec_len,1);
    mu_matlab = mean(test);
    mu_mine = avg(test);

    if (mu_matlab == mu_mine)
        success = success + 1;
    end
end

if (success == num_tests)
    disp('Victory!')
else
    disp('Try, try again.')
end
```

In this lab, we will consider a system described by a first-order difference equation. We will look at three different approaches for determining the output of this first-order system, using a rectangular pulse as the input in all three cases.

**Input and System Definition**

To demonstrate good nomenclature and MATLAB practice, we define the rectangular pulse input for you below, and you may use this exact code in your own submission; however, please make sure you understand how it works. Perhaps the easiest way to check your understanding is to make a stem plot of the final vector **x**, and see how that plot changes when you change the parameters (i.e., **n_x1**, **n_x2**, **n_on**, **n_off**). However, you should not submit any of these stem plots, and you should make sure to restore all variables to the values defined below before continuing the assignment.

```
% Create input signal (rectangular pulse)
n_x1 = -5; n_x2 = 20; n_x = n_x1:n_x2;
n_on = 0; n_off = 11;
x = u(n_x - n_on) - u(n_x - n_off);
```

This calls the function **u.m**, which computes a unit step function:

```
function y = u(x)
% unit step function
y = double(x>=0);
end
```

In lecture, we wrote a first-order difference equation in the following form:

$$y[n] - ay[n-1] = x[n], \tag{1}$$

and we derived the impulse response of this system to be

$$h[n] = a^n u[n].$$

Throughout this lab, we will choose the exponent parameter as $a = 1/2$.

```
% Define system;
a = 0.5;          % Exponent parameter
```

**Approach A: Direct Solution of the Difference Equation**

A conceptually straightforward way to compute the output $y[n]$ of the first-order system is by solving the difference equation iteratively. In lecture, we used this method considering a delta function input $x[n] = \delta[n]$, so the output we found was the impulse response, $y[n] = h[n]$. Here, we will use that method when the input $x[n]$ is a rectangular pulse.

A straightforward way to implement the iterative method used in lecture would be with a FOR loop. Consider the following bit of code, which should be easy for you to understand.

```
n_0 = 0;                % Time at which we start solving the equation
k_0 = find(n_x==n_0);   % Index of time at which we start solving eqn.
n_yA1 = n_x1; n_yA2 = n_x2; n_yA = n_yA1:n_yA2;
y_A = zeros(size(n_yA));
for k = k_0:length(n_yA)
    y_A(k) = a*y_A(k-1) + x(k)
end
```

However, this approach fails to take advantage of the many built-in MATLAB functions, which we should try to use when possible for efficiency and readability. In this case, MATLAB has a convenient and efficient way for performing the iterative computation without writing it explicitly: the **`filter`** command. In order to understand its use, we recall the general canonical form of the difference equation shown in lecture (see *EE 102A Course Reader*, page 71):

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \tag{2}$$

The **`filter`** command can be invoked as

**`filter(b, a, x)`**, where

**`b`** is the vector of the $M+1$ coefficients $b_k$ appearing in (2), in order $k = 0, 1, \ldots, M-1, M$.

**`a`** is the vector of the $N+1$ coefficients $a_k$ appearing in (2), in order $k = 0, 1, \ldots, N-1, N$.

**`x`** is the input signal vector, in this case, the rectangular pulse defined above.

Thus, we find the output using the **`filter`** command as shown below. (The A's in the variable names are to distinguish the results of this approach from those obtained later using Approaches B and C.)

```
n_0 = 0;                % Time at which we start solving the equation
k_0 = find(n_x==n_0);   % Index of time at which we start solving eqn.
n_yA1 = n_x1; n_yA2 = n_x2; n_yA = n_yA1:n_yA2;
y_A = zeros(size(n_yA));

% The a and b parameters from the general filter command:
a_fo = [1 -a]; b_fo = 1;
y_A(k_0:end) = filter(b_fo,a_fo,x(k_0:end));
```

Now plot both the input `x` and output `y` on the same plot using **`subplot`**. We will provide the code to generate the plots for Approach A, but you will need to edit this code slightly to generate similar plots for Approaches B and C.

```
figure(1)
subplot(211)
stem(n_x,x);
lw = 1.5; l=get(gca,'children'); set(l,'linewidth',lw)
xlabel('n'); ylabel('x[n]');
title('Input Signal')

subplot(212)
stem(n_yA,y_A);
lw = 1.5; l=get(gca,'children'); set(l,'linewidth',lw)
xlabel('n'); ylabel('y[n]');
title('Output Signal, Solution of Difference Equation')
```

**Approach B: Convolving Input with Impulse Response**

A straightforward way to determine the output of a system for a given input is to convolve the input with the impulse response of the system. This is the approach we will take here.

When we perform these convolutions analytically, we are often considering infinite-length sequences. For example, the impulse response $h[n]$ of an infinite impulse response system, including the first-order system studied here, has infinite length. In MATLAB, however, all sequences must be represented by finite-length numerical vectors. For the rectangular pulse input $x[n]$ we consider, this is a non-issue, as it is already defined to have finite length. For the impulse response $h[n]$, however, this does introduce a potential complication. We must make sure that when we represent $h[n]$ by a finite-length numerical vector, it is sufficiently long that our numerical convolution yields a sufficiently accurate numerical representation of the output $y[n]$, and any samples not included in the finite-length representation of $h[n]$ are sufficiently small as to be negligible. (Obviously, the concepts of "sufficiently accurate" and "sufficiently small" depend on the application.) Here, we will define a cut-off value such that, if the impulse response is below this value, we can consider it negligible. We will then define our impulse response to be long enough that all values above this cut-off are included.

```
c = 1e-5;                          % cut-off value
n_h2 = ceil(log(c)/log(abs(a))); % cut-off index
n_h1 = n_x1; nh = n_h1:n_h2;     % impulse response indexing
n_yB1 = n_x1+n_h1; n_yB2 = n_x2+n_h2; n_yB = n_yB1:n_yB2; % output
indexing
```

Here is a function **hfo.**m, that computes the impulse response of the first-order system described by (1):

```
function h = hfo(n,a)
% first-order system impulse response
h = (a.^n).*u(n);
end
```

We derived this impulse response in lecture, so please review your lecture notes to make sure you understand the derivation (see *EE 102A Course Reader*, page 42). We convolve the impulse response with the input to determine the output vector:

```
h_fo_trunc = hfo(nh,a);
y_B = conv(x,h_fo_trunc);
```

Create and include plots similar to those from Approach A.

**Approach C: Combining Step Responses**

Before beginning out final approach, let's recall how we defined our input rectangular pulse:

```
x = u(n_x-n_on) - u(n_x-n_off);
```

This input is simply the sum of two scaled and shifted step functions. Since our first-order difference system is linear and time-invariant, this implies that the output **y** is a sum of two similarly scaled and shifted step responses. Create a MATLAB function **sfo.m**, which accepts a time index vector **n** and the first-order system parameter **a**, and returns a vector of the samples of the step response, just as **hfo.m** accepted

these parameters and returned the impulse response. Using the **`sfo.m`** function, generate the output vector **`y_C`**. Include plots similar to those generated in Approaches A and B.