

Chapter 4 HW Hints

Problem 1. You need to verify equation (4.15) from the notes, which is:

$$\sum_{k=-\infty}^{\infty} \delta(t - kT) = \frac{1}{T} \sum_{n=-\infty}^{\infty} e^{jn\omega_s t} \quad (1)$$

using various numbers of terms (*i.e.* finite limits on the summation of (1) instead of ∞). In effect you are evaluating the right-hand side of (1) to see if it produces a train of impulses. I wrote a MATLAB function `train.m` to accomplish this, with syntax

```
>> help train
```

```
Function [t,IMP] = TRAIN(T,N) creates a Fourier series approximation to a
train of impulses with period T using N harmonic terms each side of zero.
Three periods of the function are generated. Returned variable "IMP" is
the impulse train, while "t" is a corresponding time vector, useful in
plotting.
```

Returned vector `'imp'` is the approximation to the impulse train, while returned vector `'t'` is the associated time vector (useful for plotting). I called the function `'train'` since we're simulating a train of impulses. For simplicity I selected sample period $T = 1$ seconds; this also fixes the sample frequency ω_s . I think you have to use a MATLAB `"for"` loop to sum over the n frequency index. You can, however, define a fixed time vector t (I used $0 \dots 3T$ with 0.005 second spacing) and perform the exponentiation using the entire time vector t (without using the MATLAB element-by-element `"."` operator).

Note that even though you're performing an exponentiation using an imaginary exponent, your result must be real (because the left-hand side of (1) is real...another reason is because the impulse train is an *even* function and will have no sine (Imag) terms in the series). There may be a *very small* imaginary part created—you can eliminate it by taking the real part of the result—you might have a statement like this within your loop: `imp = imp+real(exp(j*n*ws*t))`—actually it is instructive *not* to delete the imaginary part at first just to see how large it is (it's small). Just make sure you take the **real** part before you try to plot (or use the `abs` function to obtain the magnitude: $\sqrt{\text{Real}^2 + \text{Imag}^2}$).

Problem 2. You can use MATLAB to find the $G(z)$ (the forward path transfer function) using the `'zoh'` method in `c2d`, or you can do it by hand following the examples in Section 4.5. It might be worthwhile to check your MATLAB result by hand (or *vice-versa*).

Anyway, once you get $G(z)$ you can then find the closed-loop transfer function—I'll call it $W(z)$ —by reducing the feedback loop manually, where

$$W(z) = \frac{G(z)}{1 + G(z)}$$

or, you can use the MATLAB `feedback` function to find the closed-loop transfer function, something like this:

```
>> W = feedback(Gd,1); % Forward path discrete TF is Gd, feedback path is 1 (unity)
```

Now, you can find the *samples* of the step response of $W(z)$ (this will be output $Y(z)$) by using MATLAB `step`, but this will draw a "stairstep" response if you call it like

```
>> step(W); % Find and plot unit step response
```

However, this is *not quite* what I asked for—I want you to find the *complete* unit step response $y(t)$ —you need to "connect the dots." Therefore, I would recommend you invoke `step` like this:

```
>> y = step(W); % Find unit step response and return in y
```

Now you can take the first three samples ($k = 0, 1, 2$) and work with them (since we only want the first three samples you can “connect the dots” manually). As discussed in Section 4.5.3, the response *between* samples is the (scaled and biased) plant ($G(s)$) step response. In equation (2) below, g_{step} is the unit step response; A does the scaling, and B does the biasing. So, to find the response between samples you fit the curve

$$y(t) = Ag_{step}(t) + B \quad (2)$$

where (2) represents the *scaled* and *biased* unit step response. Parameters A and B can be found using boundary conditions which are the two sample points the function connects. Remember, consider only samples 0, 1, and 2, *i.e.* you will fit equation (2) twice.

Problem 3. Use the same discretization method (`zoh`) here as in Problem 2, although we’re not concerned with anything between the samples. If you do these manually (not a bad idea) I think you will have to perform a partial-fraction expansion for (c), since the $F(s)$ given there is *not* in our transform table. Note that one check on your results is that the poles of $G(s)$ and the $G(z)$ you find *must* be related by $z = e^{sT}$. You can use MATLAB `c2d` to check your hand calculations (or *vice-versa*).

If you’re interested, compare the frequency-response characteristics (*i.e.* Bode plots) of one of the $G(z)$ found here (don’t use (c) since it’s unstable) to one of the “integration rules” (*e.g.* trapezoidal, prewarped) we used in Chapter 3.

Problem 4. Here you’re reconstructing a continuous time function using *only* its samples—pretty amazing! The *ideal* reconstruction formula is equation (4.28),

$$r(t) = \sum_{k=-\infty}^{\infty} r(kT) \operatorname{sinc} \frac{\pi(t - kT)}{T} \quad (3)$$

You’ll only be summing from $k = 0..5$, and the samples $r(kT)$ come from the 1 Hz sine wave. I wrote a MATLAB script file to compute the reconstruction using two nested loops: (1) the outer loop over time (0 to 1 seconds with 0.01 time step), and (2) the inner loop over k to compute the summation of equation (3).

NOTE: The MATLAB `sinc` function (it’s in the *Signal Processing Toolbox*—if you don’t have that toolbox you don’t have `sinc`). Anyway, the MATLAB `sinc` *will not work* for this assignment—it inherently incorporates a “ π ” scaling parameter. You must use the `sinc.m` function I will have on my website (in **Homework tips and answers**). Just copy and paste it into the MATLAB editor and save it in your working directory as `sinc.m`.

Your reconstruction—while not perfect—should look pretty good for only six samples. You can imagine that with many more samples this “ideal” desampling filter should perform very well.