Homework Program 9: Approximating Periodic Pulses with Cosine Waves

due Friday, Dec. 1 at 6:00 am

In this assignment, you will write a MATLAB function that approximates an input vector as the sum of cosine waves. The approximated vector will minimize the sum of squared errors over all the elements. Specifically, given the row vector

$$\mathbf{s} = [s(1)\ s(2)\ s(3), \cdots, s(T)], \tag{1}$$

the assignment is to compute the approximation $\tilde{\mathbf{s}} = [\tilde{s}(1)\ \tilde{s}(2)\ \tilde{s}(3), \cdots, \tilde{s}(T)]$, where

$$\tilde{s}(k) = c_0 + c_1 \cos(2\pi f_0 k) + c_2 \cos(2\pi(2f_0)k) + c_3 \cos(2\pi(3f_0)k) + \cdots + c_n \cos(2\pi(nf_0)k) \tag{2}$$

for $k = 1, \cdots, T$.

Note that $\cos(2\pi ft)$ is a cosine wave at frequency f cycles/per second, where t is the continuous time variable. In (2) the values $\cos(2\pi fk)$ then correspond to "samples" of the cosine wave at times $k = 1, \cdots, T$. This is illustrated in Fig. 1. The frequency f_0 is called the "fundamental" frequency, and each cosine wave has a frequency, which is an integer multiple of f_0, that is, the frequencies used in the approximation are $f_0,\ 2f_0, \cdots, nf_0$, where n is the number of sinusoidal terms. The combining coefficients $c_0, \cdots, c_n$ will be selected to minimize the least squares objective

$$\mathcal{E} = \|\mathbf{s} - \tilde{\mathbf{s}}\|^2 = \sum_{k=1}^{T} [s(k) - \tilde{s}(k)]^2 \tag{3}$$

We can think of the vector $\mathbf{s}$ as a continuous-time function, or "signal" sampled in time. For example, the continuous-time signal $s(t)$ may represent the position of a space craft, a communications signal received at a cell phone, temperature in Evanston over time, a musical tone (or chord) from several instruments, neural spikes, or pressure associated with heartbeats. The vector $\mathbf{s}$ in (1) then consists of samples of $s(t)$ at discrete times $t = t_0,\ 2t_0, \cdots, Tt_0$, where t_0 represents the time between samples. Here we assume that t_0 is normalized to one. For this assignment we will assume that $s(t)$ is periodic, meaning it consists of a segment which repeats itself over and over. Examples of signals which display periodic characteristics are the position of a pendulum or rotating object, pressure from a heartbeat, and a fixed note from a musical instrument. It turns out that those types of signals can generally be closely approximated as the sum of many periodic sine and cosine waves.

An example of the approximation (2) with $n = 1$ (a single cosine wave) is shown in Fig. 2. The signal $\mathbf{s}$ is a series of square pulses, called a "pulse train", which repeats itself every 150 samples. Each pulse has width 50 samples and the time between successive pulses is 100 samples. The approximate signal is a cosine wave with period 150, or frequency $f_0 = 1/150$. That is, if the time between samples $t_0 = 1$ second, then each cycle lasts 150 seconds and so the frequency is 1/150 cycles per second.

The assignment is to generate approximations of the pulse train for larger values of n (adding up many cosine waves). Decomposing a signal into a sum of sinusoidal signals, as in this assignment, has many applications, and is called "Fourier analysis". Such a decomposition will typically contain `sin` terms at the same frequencies as the `cos` terms shown in (2). For the square wave example considered in this assignment, only the `cos` terms are needed.

The assignment consists of writing two functions:

- `gen_pulse_train` generates a vector `pulse_train` that switches between 0 and 1, as shown in Fig. 2. Fig. 3 shows a single pulse, which is repeated in Fig. 2. Referring to Fig. 3, the function inputs are the number of elements in the initial "off" period, `off` (where the signal is equal to zero), the number of elements in the "on" period, `on` (where the signal is equal to one), and the number of complete cycles `cycles`, or repetitions, where each cycle consists of the single pulse shown in Fig. 3. The *period* of the

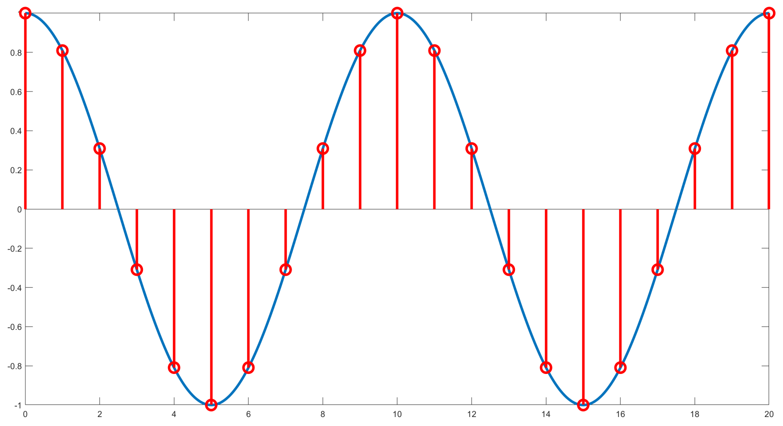

Figure 1: A cosine waveform with samples shown in red. The continuous-time waveform has period 10, or frequency 1/10, and so the vector of samples has elements $\cos(2\pi \cdot k/10)$, $k = 0, 1, 2, \cdots, 20$.

pulse train is then `on+(2*off)`. For example, if `off` is 5 and `on` is 3 then one cycle of the pulse train would be the array `[0 0 0 0 0 1 1 1 0 0 0 0 0]`.

The function has two outputs: `pulse_train` and the fundamental frequency `f0`, which is 1/(period). Check that each of the inputs `on`, `off`, and `cycles` are scalars, and that each is a positive integer.

Hint: To generate the pulse train define an array for the single pulse shown in Fig. 3, and then replicate it by either concatening the array with itself or using the Matlab command `repmat`.

- `approx_pulse` computes the approximate signal $\tilde{\mathbf{s}}$. The inputs are the signal (vector) `sig` to be approximated, the number of terms `n` in (2), and the fundamental frequency `f0`. The outputs are the combining coefficients, `coeffs` $(c_0, \cdots, c_n)$, the approximated signal `sig_approx`, and the approximation error `error` given by $\mathcal{E}$ in (3). The function should print the expression for the approximate signal along with the approximation error. For the example in Fig. 2, after generating `sig` with `gen_pulse_train` the function call and output should be:

```
>> [coeffs, sig_approx, approx_error] = approx_pulse(pulse_train,1,f0);

approximate signal = 0.33 + -0.55 cos(2*pi*1*f0*k)

The approximation error is 42.170934
```

  For larger n, the approximate signal should show the additional terms.

  In the function check that `sig` is a row vector, and that `n` is a positive integer.

Check your code by generating a pulse train with period 150 and 4 cycles, as shown in Figures 2 and 3, that is, `off` = `on` = 50. Use `approx_pulse` to approximate this signal for $n = 20$, as shown in Fig. 4.

Repeat this for `off` = 25 and `on` = 50, corresponding to a symmetric square wave, and observe how well the approximation matches this square wave for $n = 1$, 3, 10, and 20. You do not have to submit the plots – just observe how the approximation improves as n increases, and where it deviates the most from `sig`.

Write a script to compute the approximation error for `n=1:30` along with the *maximum overshoot error* `max(sig_approx-1)` and the *maximum undershoot error* `max(-sig_approx)`. The overshoot error is the amount the approximation "overshoots" the height of each pulse, and similarly, the undershoot error is the

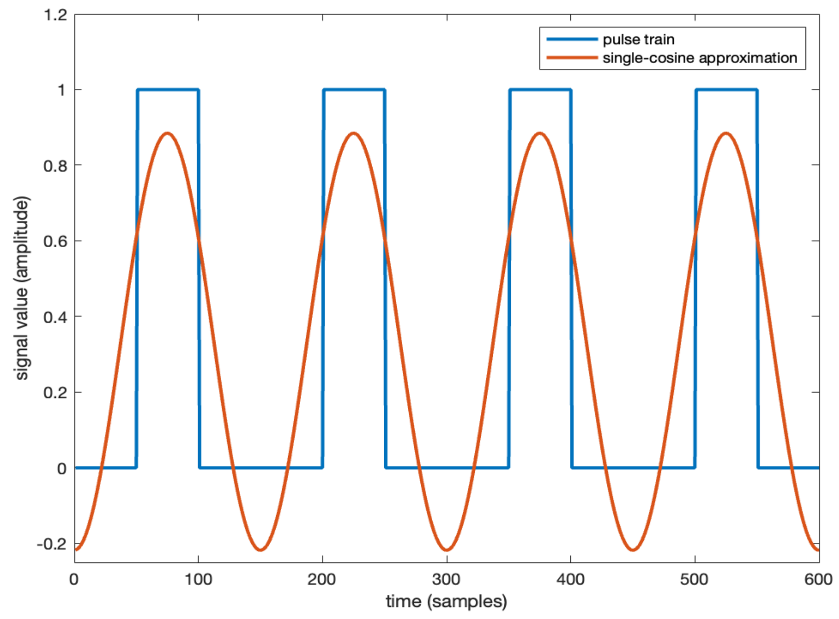

Figure 2: Pulse train and a cosine wave approximation.

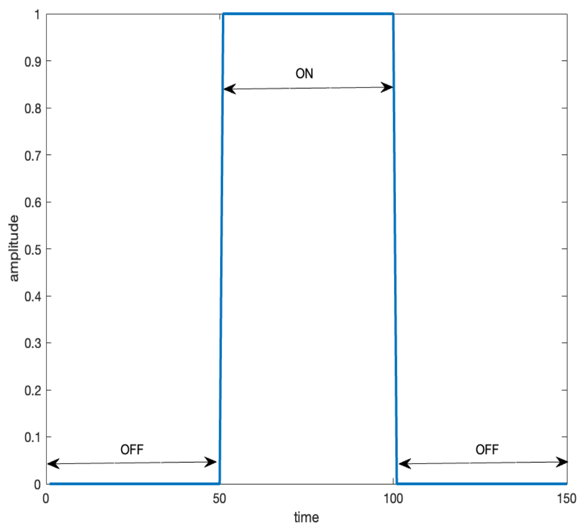

Figure 3: Single pulse showing “on” and “off” periods.

amount the approximation undershoots the bottom of the pulse. These are illustrated in Fig. 4. Notice that the maximum overshoot and undershoot occur close to the discontinuities, where the pulse rises and falls.

- Looking at the errors, which terms in the approximation (2) can be omitted without increasing the errors? That is, the errors remain the same whether or not these terms are included. Indicate which specific terms can be omitted.

Finally, repeat the preceding example with a narrower pulse train: referring to Fig. 3, set `off` = 80 and `on` = 40. Generate 4 cycles for the pulse train. Run the preceding script to print the same three errors for `n=1:30`. Plot the pulse train and the approximation for $n = 5$, and observe where the maximum overshoot and undershoot occur. Again, you do not have to submit the plot, just answer the following question.

- In this example, where does the maximum overshoot occur – in the middle of each narrow pulse or near an edge? As n increases, does this change?

Submit the functions `gen_pulse` and `approx_pulse` as one file along with the outputs from running the script for the two examples. Include comments that clearly delineate the three sections of code. Display the three errors as column vectors and append these to your functions along with the answers to the preceding questions. The values for `n=1:3` for the first example are shown below to illustrate how the output should look.

```
Least Squares Error      Overshoot Error       Undershoot Error
72.000240               -0.425835              0.174165
35.366009               -0.123204              0.160457
19.097525                0.078467              0.120844
```

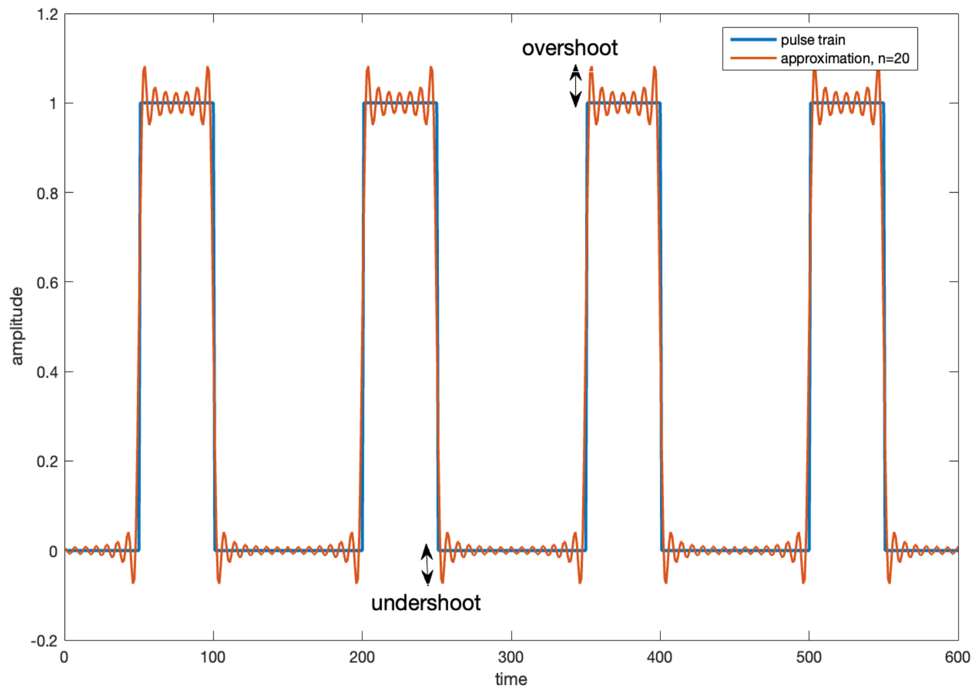

Figure 4: Illustration of the maximum overshoot and undershoot for $n = 20$.