Key for Long Problem Set 7

(1). The goal when smoothing data is to improve the signal-to-noise ratio without distorting the underlying signal. The data in the file "StepFunction.RData" consists of four objects: the vector x, which contains 200 values for plotting on the x-axis; the vector y, which contains 200 values for a step-function that satisfies the following criteria

$$y = 0 \text{ for } x \leq 75 \text{ and for } x \geq 126$$

$$y = 1 \text{ for } 75 < x < 126$$

the vector n, which contains 200 values drawn from random normal distribution with a mean of 0 and standard deviation of 0.1, and the vector s, which is the sum of y and n. In essence, y is the pure signal, n is the noise, and s is a noisy signal. Using this data, complete the following tasks: (a) Determine the mean signal, the standard deviation of the noise, and the signal-to-noise ratio for the noisy signal using just the data in the object s. (b) Explore the effect of applying to the noisy signal, one pass each of moving average filters of widths 5, 7, 9, 11, 13, 15, and 17. For each moving average filter, determine the mean signal, the standard deviation of the noise, and the signal-to-noise ratio. Organize these measurements using a table and comment on your results. Prepare a single plot that displays the original noisy signal and the smoothed signals using widths of 5, 9, 13, and 17, off-setting each so that all five signals are displayed. Comment on your results. (c) Repeat (b) using Savitzky-Golay quadratic/cubic smoothing filters of widths 5, 7, 9, 11, 13, 15, and 17; see Table I of the original paper (link on the course schedule). (d) Considering your results for (b) and for (c), what filter and width provides the greatest improvement in the signal-to-noise ratio with the least distortion of the original signal's step-function? Be sure to justify your choice.

Answer. For part (a), we begin by examining the data, plotting the signal as a function of its index in Figure 1.

```
load("StepFunction.RData")
plot(x, s, type = "l", lwd = 2, col = "blue")
abline(v = 20, lty = 2, col = "red")
abline(v = 55, lty = 2, col = "red")
abline(v = 90, lty = 3, col = "red")
abline(v = 110, lty = 3, col = "red")
```

To calculate the signal-to-noise ratio we find the mean for some portion of the signal at the top of the step function, perhaps from 90 to 110, and we find the standard deviation for some portion of the noise, perhaps from 20 to 55. The choice of limits is not particularly important; however, anticipating that we later will calculate the signal-to-noise ratio following filtering, these limits will allow us to use the same range of data

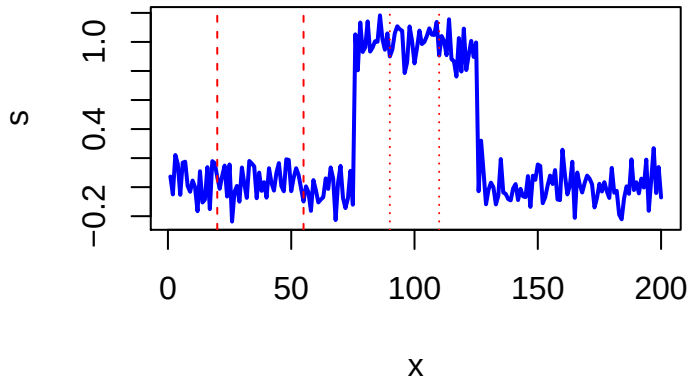

Figure 1: The original noisy signal. The vertical dashed lines show the points used to calculate the noise's standard deviation, and the vertical dotted lines show the points used to calculate the signal's mean.

for all calculations without the need to worry about how the filter distorts the signal's shape; thus, the mean signal is 0.981, the standard deviation of the noise is 0.102, and the signal-to-noise ratio is 9.836.

For (b), we need to create the moving average filters and apply them to the noisy signal, which we will do by combining both into a single command. The table below compares the mean for the signal, the standard deviation for the noise, and the signal-to-noise ratio for each filter using the same code as above for the noisy signal in part (a); note that the window of size 0 is the original, unfiltered data.

```
library(xtable)
s5 = filter(s, rep(1/5, 5))
s7 = filter(s, rep(1/7, 7))
s9 = filter(s, rep(1/9, 9))
s11 = filter(s, rep(1/11, 11))
s13 = filter(s, rep(1/13, 13))
s15 = filter(s, rep(1/15, 15))
s17 = filter(s, rep(1/17, 17))
window = c(0, seq(5, 17, 2))
mean_signal = c(mean(s[90:110]), mean(s5[90:110]), mean(s7[90:110]),
                mean(s9[90:110]), mean(s11[90:110]), mean(s13[90:110]),
                mean(s15[90:110]), mean(s17[90:110]))
sd_noise = c(sd(s[20:55]), sd(s5[20:55]), sd(s7[20:55]),
             sd(s9[20:55]), sd(s11[20:55]), sd(s13[20:55]),
             sd(s15[20:55]), sd(s17[20:55]))
signal_to_noise = mean_signal/sd_noise
df = data.frame(window, mean_signal, sd_noise, signal_to_noise)
print(xtable(df, digits = c(0, 0, 4, 4, 2)), type = "latex")
```

% latex table generated in R 3.2.3 by xtable 1.8-2 package % Sun Nov 20 12:59:57 2016

	window	mean_signal	sd_noise	signal_to_noise
1	0	1.0041	0.1021	9.84
2	5	1.0085	0.0426	23.68
3	7	1.0071	0.0350	28.78
4	9	1.0073	0.0280	35.99
5	11	1.0061	0.0211	47.63
6	13	1.0044	0.0182	55.23
7	15	1.0030	0.0158	63.64
8	17	1.0022	0.0166	60.32

Note that signal's mean more closely approximates its true value of 1.00 as the window's size increases and that the standard deviation of the noise decreases as the window's size increases; both are the expected result of smoothing the noise. As a result, the signal-to-noise ratio improves as the window's size increases. The one exception to this trend is the 15-point moving average filter where the standard deviation of the noise and the signal-to-noise ratio are slightly worse than for the 13-point moving average filter; this is an artifact of the particular set of indexed values used for these calculations.

A plot of the pure signal, the noisy signal, and the signal after applying 5-point, 9-point, 13-point, and 17-point moving average filters is shown in Figure 2. One interesting feature of the smoothed data is the visible distortion of the signal's underlying step-function, which becomes broader and trapezoidal in shape as the size of the moving average filter increases.

```
plot(x, s, type = "l", col = "blue", ylim = c(0, 8))
lines(x, s5 + 1.5, type = "l", col = "blue")
lines(x, s9 + 3.0, type = "l", col = "blue")
lines(x, s13 + 4.5, type = "l", col = "blue")
lines(x, s17 + 6.0, type = "l", col = "blue")
```

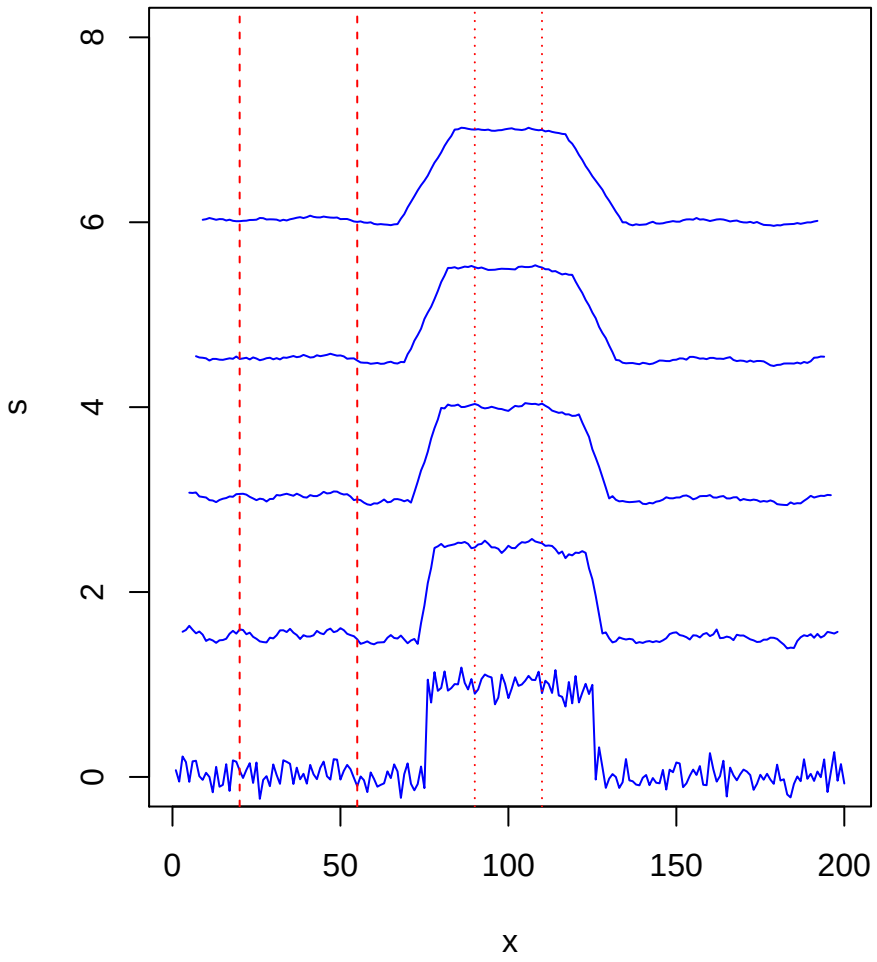

Figure 2: The original noisy signal after applying a moving average filter of size (from bottom-to-top) 0, 5, 9, 13, and 17. The vertical dashed lines show the points used to calculate the noise's standard deviation, and the vertical dotted lines show the points used to calculate the signal's mean.

```
abline(v = 20, lty = 2, col = "red")
abline(v = 55, lty = 2, col = "red")
abline(v = 90, lty = 3, col = "red")
abline(v = 110, lty = 3, col = "red")
```

For (c), we follow the same process using the coefficients for a Savitzky-Golay filter. The table below compares the mean for the signal, the standard deviation for the noise, and the signal-to-noise ratio for each filter using the same code as above for the noisy signal in part (a); note that the window of size zero is the original, unfiltered data.

```
sg5 = filter(s, c(-3, 12, 17, 12, -3)/35)
sg7 = filter(s, c(-2, 3, 6, 7, 6, 3, -2)/21)
sg9 = filter(s, c(-21, 14, 39, 54, 59, 54, 39, 14, -21)/231)
sg11 = filter(s, c(-36, 9, 44, 69, 84, 89, 84, 69, 44, 9, -36)/429)
sg13 = filter(s, c(-11, 0, 9, 16, 21, 24, 25, 24, 21, 16, 9, 0, -11)/143)
sg15 = filter(s, c(-78, -13, 42, 87, 122, 147, 162, 167,
                   162, 147, 122, 87, 42, -13, -78)/1105)
sg17 = filter(s, c(-21, -6, 7, 18, 27, 34, 39, 42, 43, 42,
                   39, 34, 27, 17, 7, -6)/323)
```

```
window = c(0, seq(5, 17, 2))
mean_signal = c(mean(s[90:110]), mean(sg5[90:110]), mean(sg7[90:110]),
                mean(sg9[90:110]), mean(sg11[90:110]), mean(sg13[90:110]),
                mean(sg15[90:110]), mean(sg17[90:110]))
sd_noise = c(sd(s[20:55]), sd(sg5[20:55]), sd(sg7[20:55]), sd(sg9[20:55]),
             sd(sg11[20:55]), sd(sg13[20:55]), sd(sg15[20:55]),
             sd(sg17[20:55]))
signal_to_noise = mean_signal/sd_noise
df = data.frame(window, mean_signal, sd_noise, signal_to_noise)
print(xtable(df, digits = c(0, 0, 4, 4, 2)), type = "latex")
```

% latex table generated in R 3.2.3 by xtable 1.8-2 package % Sun Nov 20 12:59:58 2016

	window	mean_signal	sd_noise	signal_to_noise
1	0	1.0041	0.1021	9.84
2	5	1.0083	0.0644	15.66
3	7	1.0093	0.0515	19.61
4	9	1.0079	0.0485	20.76
5	11	1.0087	0.0450	22.40
6	13	1.0094	0.0427	23.66
7	15	1.0092	0.0362	27.84
8	17	1.0698	0.0295	36.31

Note that signal's mean more closely approximates its true value of 1.00 as the window's size increases and that the standard deviation of the noise decreases as the window's size increases; both are the expected result of smoothing the noise. As a result, the signal-to-noise ratio improves as the window's size increases. When compared to a moving average filter of equal size, the Savitzky-Golay filter provides less improvement in the signal-to-noise ratio.

A plot of the pure signal, the noisy signal, and the signal after applying 5-point, 9-point, 13-point, and 17-point Savitzky- filters is shown in Figure 3. One interesting feature of the smoothed data is the visible distortion of the signal's underlying step-function, which becomes broader and trapezoidal in shape as the size of the filter increases, but the effect is not as pronounced as for the moving average filters.

```
plot(x, s, type = "l", col = "blue", ylim = c(0, 8))
lines(x, sg5 + 1.5, type = "l", col = "blue")
lines(x, sg9 + 3.0, type = "l", col = "blue")
lines(x, sg13 + 4.5, type = "l", col = "blue")
lines(x, sg17 + 6.0, type = "l", col = "blue")
abline(v = 20, lty = 2, col = "red")
abline(v = 55, lty = 2, col = "red")
abline(v = 90, lty = 3, col = "red")
abline(v = 110, lty = 3, col = "red")
```

For (d), a five-point moving average filter provides a substantial improvement in the signal-to-noise ratio with minimal distortion to the signal's shape; although the Savitzky-Golay five-point smooth likely has less distortion, the difference in peak shape is not sufficient to overcome its somewhat poorer improvement in the signal-to-noise ratio.

(2). The file "FFT.RData" contains four time-domain signals (a, b, c, and d), each consisting of 32 values and each consisting of a single peak. Examine the data and describe how the time-domain signals differ from each other in terms of peak position, peak width at the baseline, peak area, and peak height. Complete a FFT of each time-domain signal and examine the real part of each in the frequency-domain. Describe how the frequency-domain signals differ from each other in terms of initial intensity, the distance between peak maxima (or between peak minima), and the rate at which the peak maxima (or peak minima) decrease in value. How do the features in the time-domain signals relate to the features in the frequency-domain signals?

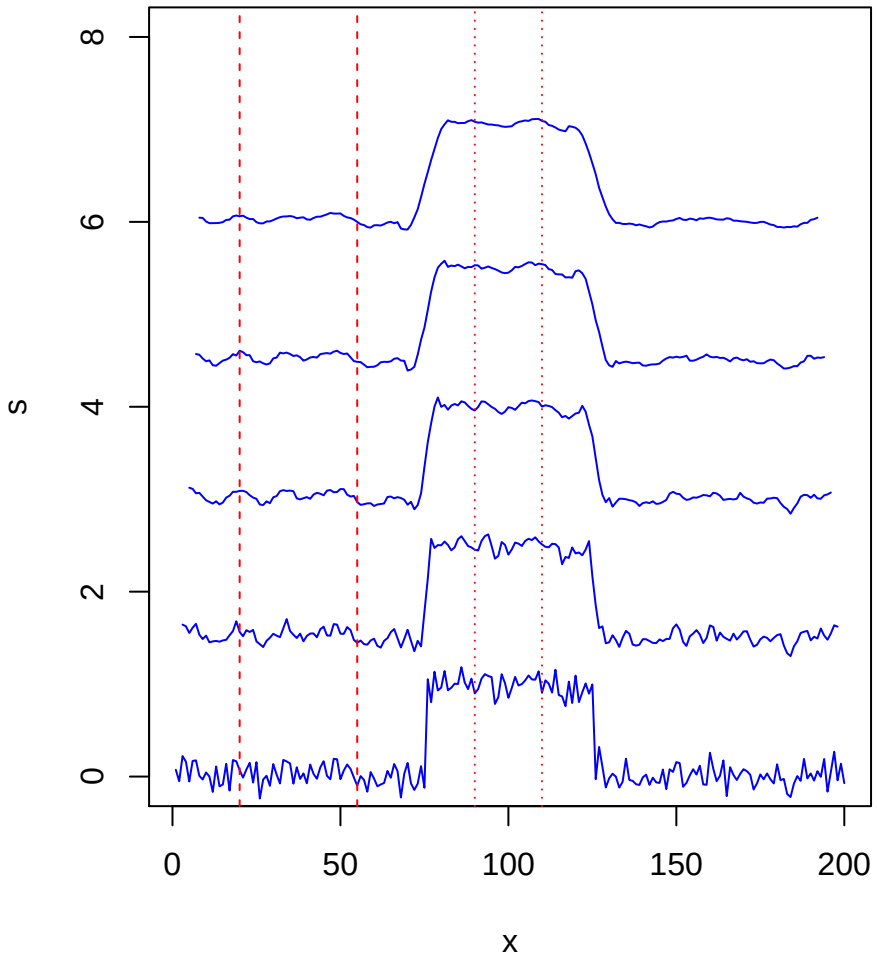

Figure 3: The original noisy signal after applying a Savitzky-Golay filter of size (from bottom-to-top) 0, 5, 9, 13, and 17. The vertical dashed lines show the points used to calculate the noise's standard deviation, and the vertical dotted lines show the points used to calculate the signal's mean.

Answer. Each of the four functions (not shown here) consists of a single triangular peak characterized by its position, its height, and its width at the baseline. All four peaks share a common area of $0.5 \times \text{width} \times \text{height} = 1.00$; the details are summarized here:

function	position (index)	width (Δ index)	height	area
a	7	2	1.00	1.00
b	7	4	0.50	1.00
c	4	4	0.50	1.00
d	7	6	0.33	1.00

After taking the Fourier transform of each function, we plot the real portion of the results in Figure 4..

```
load("FFT.RData")
old.par = par(mfrow = c(2, 2))
plot(Re(fft(a)), type = "l", lwd = 2, col = "blue", main = "function a")
plot(Re(fft(b)), type = "l", lwd = 2, col = "blue", main = "function b")
plot(Re(fft(c)), type = "l", lwd = 2, col = "blue", main = "function c")
plot(Re(fft(d)), type = "l", lwd = 2, col = "blue", main = "function d")
```

```
par(old.par)
```

All four peaks in the time-domain have the same area, and all four functions in the frequency-domain have the same initial intensity at $x = 1$, suggesting that area and intensity are related; this is not surprising, as the area under a peak is measure of intensity. As we see when comparing functions a, b, and d to function c, the greater the peak's position in the time-domain, the shorter the period between successive peak maxima in the frequency-domain. Finally, we see that a broader peak in the time-domain results in a faster decay (damping) of the oscillation in the frequency-domain; compare, for example, functions a, b, and d (omitting c as it also has a change in position) and note that function a has the narrowest width (indeed, it is a spike) in the time-domain and yields a single non-damped cosine wave in the frequency domain, and that d is the broadest of the three peaks in the time-domain and that its intensity decreases to near zero after one full oscillation instead of after two full oscillations for b. The reason for the relationship between width and the rate of damping is that a broader peak in the time domain requires a greater number of cosine waves in the frequency domain and this increase the amount of destructive interferance when the cosine waves are added together.

(3). The file "VeryNoisySignal.RData" consists of a single object, *sn*, that contains 1024 points of noisy data with a hint of a signal. Prove that there is a signal in this file by using any one moving average or Savitzky-Golay smoothing filter of your choice and a Fourier filter. Present your results in a single figure that shows the original signal, the signal after smoothing, and the signal after Fourier filtering. Comment on your results.

Answer. A plot of the noisy signal (see Figure 5) suggests that there is a peak centered at approximately 500 with a width at the baseline that extends from approximately 400 to 600. This is broad enough, that we safely can use a smoothing filter with a fairly large window. In general, a Savitzky-Golay filter is less likely than a moving average filter to distort the signal, so we will use a 17-point window. For the Fourier filter, a plot (not shown) of the real portion of the fft suggests that results from an index of 51 to 974 represent noise; setting these values to zero and taking the inverse Fourier transform returns the filtered data. All three signals—the original noisy signal (bottom), the smoothed signal using Savitzky-Golay filter (middle), and the smoothed signal using FT filtering (top)—are shown in Figure 5.

```
load("VeryNoisySignal.RData")
sg17 = filter(sn, c(-21, -6, 7, 18, 27, 34, 39, 42, 43,
                    42, 39, 34, 27, 17, 7, -6)/323)
sn.fft = fft(sn)
sn.fft[51:974] = 0 + 0i
```

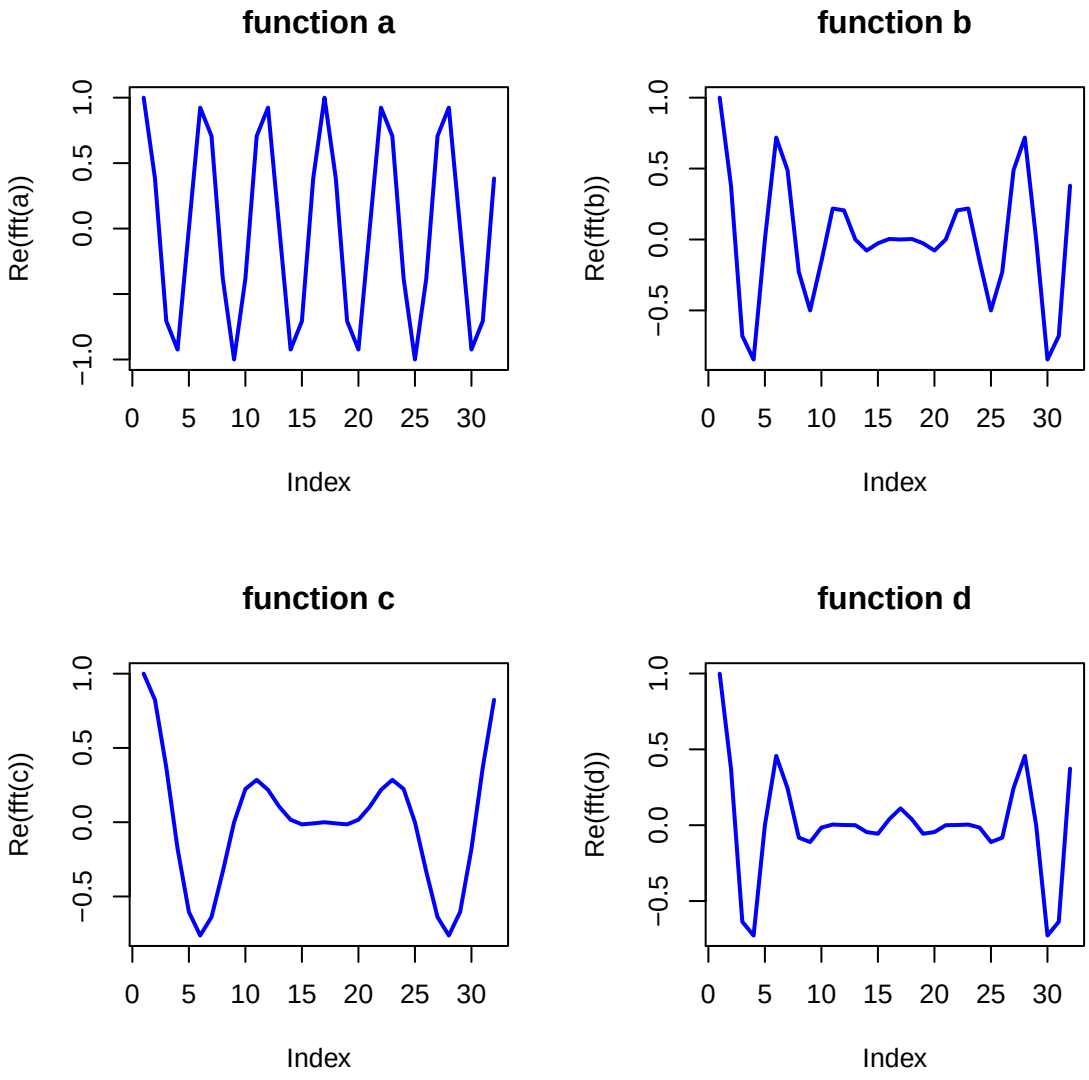

Figure 4: Fourier transforms of four triangular peak shapes.

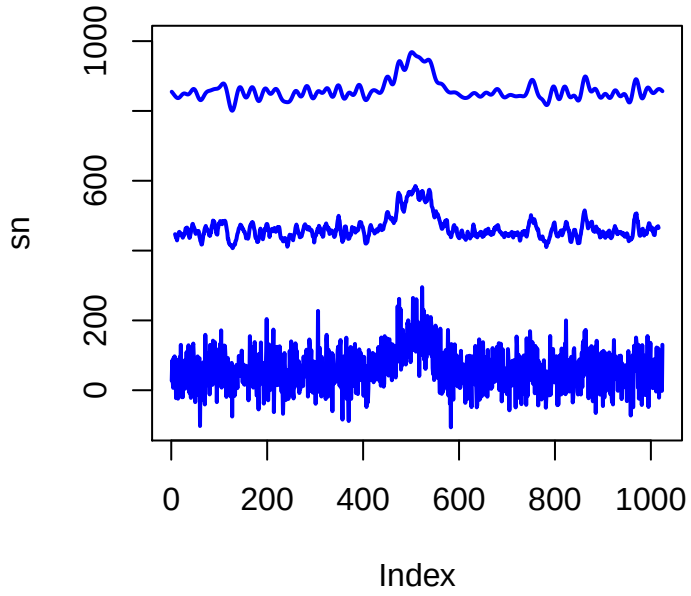

Figure 5: The original noisy signal (bottom), and the smoothed signal after applying a 17-point Savitzky-Golay filter (middle), and a Fourier transform filter (top).

```
plot(sn, type = "l", lwd = 2, col = "blue", ylim = c(-100, 1000))
lines(sg17 + 400, lwd = 2, col = "blue")
lines(Re(fft(sn.fft, inverse = TRUE)/length(sn)) + 800, lwd = 2, col = "blue")
```

(4). Use a Savitzky-Golay nine-point cubic second-derivative filter to remove the background from the data in file "BkgdRemoval.RData." Build a calibration model using these results, and report the calibration equation and a plot of the calibration curve.

Answer. A nine-point cubic Savitzky-Golay second derivative filter allows us to take the second-derivative of the five spectra. Given the shape of the second derivative, the maximum net signal is the difference between the minimum response at an index of 207 and the maximum response at an index of 196. A linear model using the data show here

```
load("BkgdRemoval.RData")
sdf = c(28, 7, -8, -17, -20, -17, -8, 7, 28)/462
y1.sd = filter(y1, sdf)
y2.sd = filter(y2, sdf)
y3.sd = filter(y3, sdf)
y4.sd = filter(y4, sdf)
y5.sd = filter(y5, sdf)
plot(wavelength, y5.sd, type = "l", lwd = 2, col = "blue", ylab = "signal")
lines(wavelength, y4.sd, lwd = 2, col = "blue")
lines(wavelength, y3.sd, lwd = 2, col = "blue")
lines(wavelength, y2.sd, lwd = 2, col = "blue")
lines(wavelength, y1.sd, lwd = 2, col = "blue")
abline(v = 245, col = "red", lty = 2, lwd = 2)
abline(v = 256, col = "red", lty = 2, lwd = 2)
```

provides the calibration curve of response $= 0.0001 + 2.0164 \times$ conc in Figure 7.

```
y.signal = c(y1.sd[191] - y1.sd[207], y2.sd[191] - y2.sd[207],
             y3.sd[191] - y3.sd[207], y4.sd[191] - y4.sd[207],
```

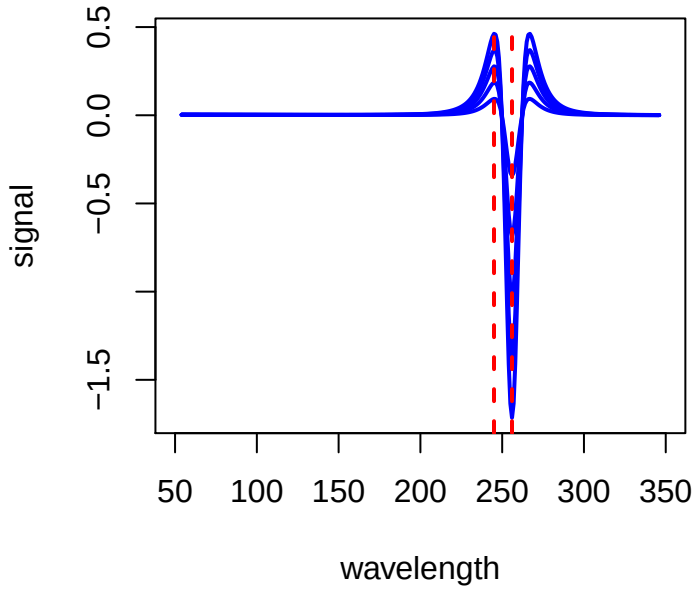

Figure 6: Results of applying a nine-point second derivative Savitzky-Golay filter to the original data; the vertical dashed lines show the values used to calculate the net response.

```
            y5.sd[191] - y5.sd[207])
plot(conc, y.signal, pch = 19, col = "blue", xlim = c(0, 1), ylim = c(0, 2.5))
lm.r = lm(y.signal ~ conc)
abline(lm.r, lwd = 2, lty = 2, col = "blue")
```

```
print(xtable(lm.r, digits = c(0, 4, 4, -2, 4), type = "latex"))
```

```
## Warning in summary.lm(x): essentially perfect fit: summary may be
## unreliable
```

% latex table generated in R 3.2.3 by xtable 1.8-2 package % Sun Nov 20 13:00:01 2016

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0001	0.0000	7.46E+10	0.0000
conc	2.0164	0.0000	1.10E+15	0.0000

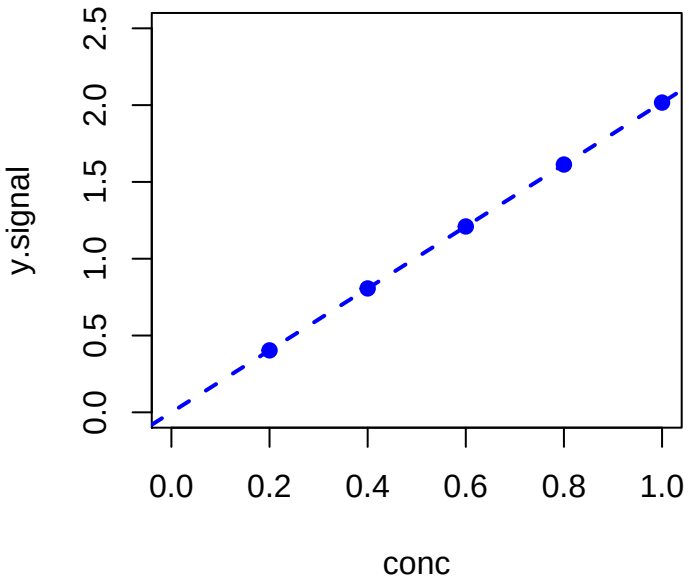

Figure 7: Calibration curve developed using the second-derivative curves from Figure 6.