Exercise 12.1 Savitzky–Golay filtering and differentiation

The Python code for this exercise is given in Listing 1. The various smoothing and differentiation filters are shown in Fig. 1 for the signal without noise, and in Fig. 2 for the signal with noise. For the signal with noise, we have here used a window of 51 to obtain the derivative. We see here that we largely can remove the noise, and that we can get a good estimate of the derivative of the noisy signal.

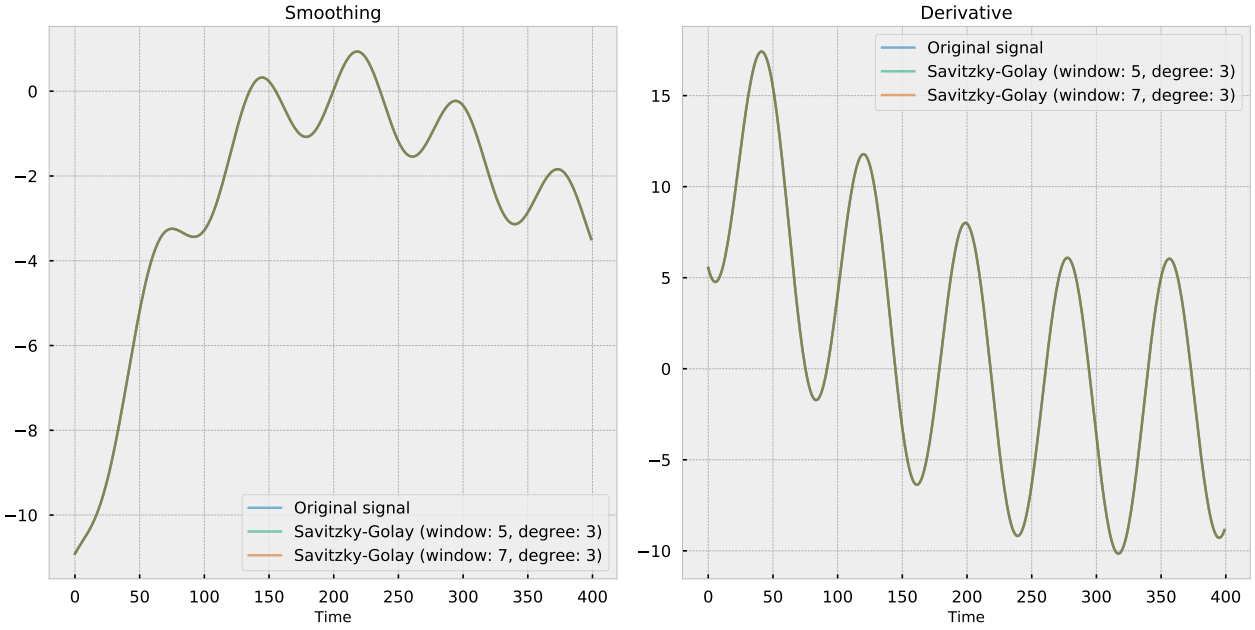

Figure 1: *Savitzky–Golay filtering of the signal without noise.*

Exercise 12.2 Smoothing by convolution

The Python code for this exercise is given in Listing 2. Figures 3 and 4 show the result of the smoothing with two different windowing functions. We see here that we can largely remove the noise present in the signals. We also note that applying these windows gives rise to so-called end effects.

Exercise 12.3 Smoothing & removing a trend

The Python solution to this exercise and be found in Listing 3. The signal, after removing the third-order trend, is shown in Fig. 5. Here, we have used a Bartlett window* to smooth

*https://en.wikipedia.org/wiki/Window_function#Triangular_window

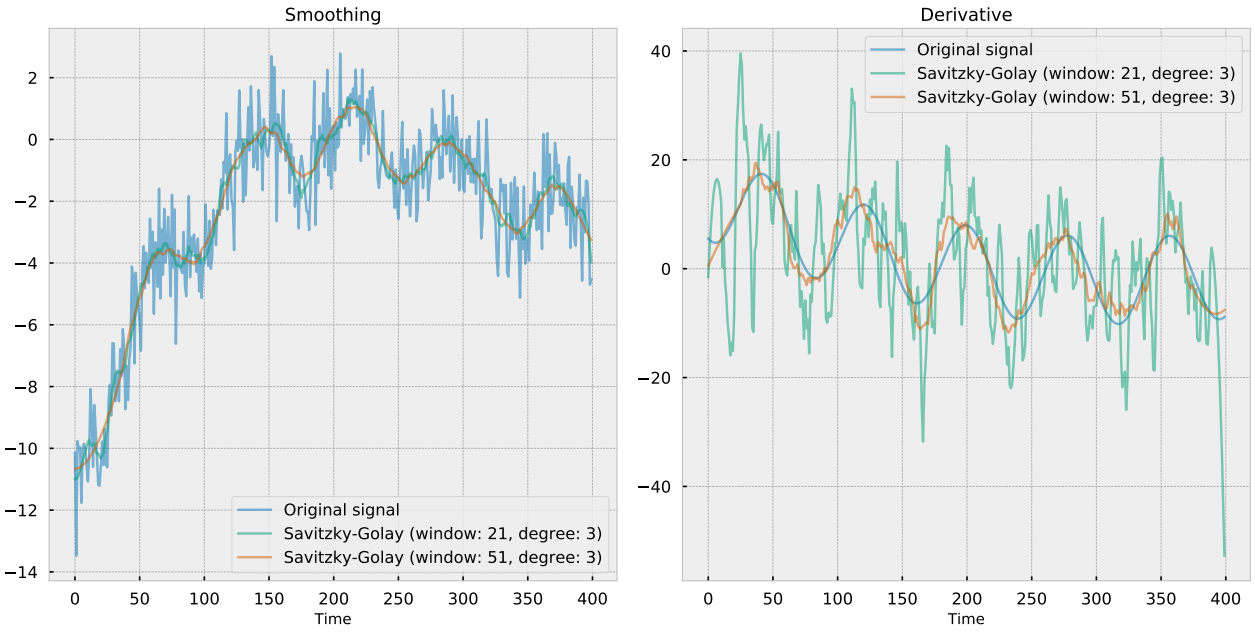

Figure 2: *Savitzky–Golay filtering of the signal with noise.*

the signal via convolution and the resulting signal with the location of the peaks can be found in Fig. 6, and in Table 1.

Location (zero-based index)
204
340
597
707
888
939

Table 1: *Location of peaks for exercise 12.3.*

Exercise 12.4 Removing a spike

The Python code for removing the spike is given in Listing 4. Here, we have implemented the median filter ourselves. We note that we could have avoided implementing it, by making use of scipy and the method scipy.signal.medfilt.[†]

[†] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.medfilt.html

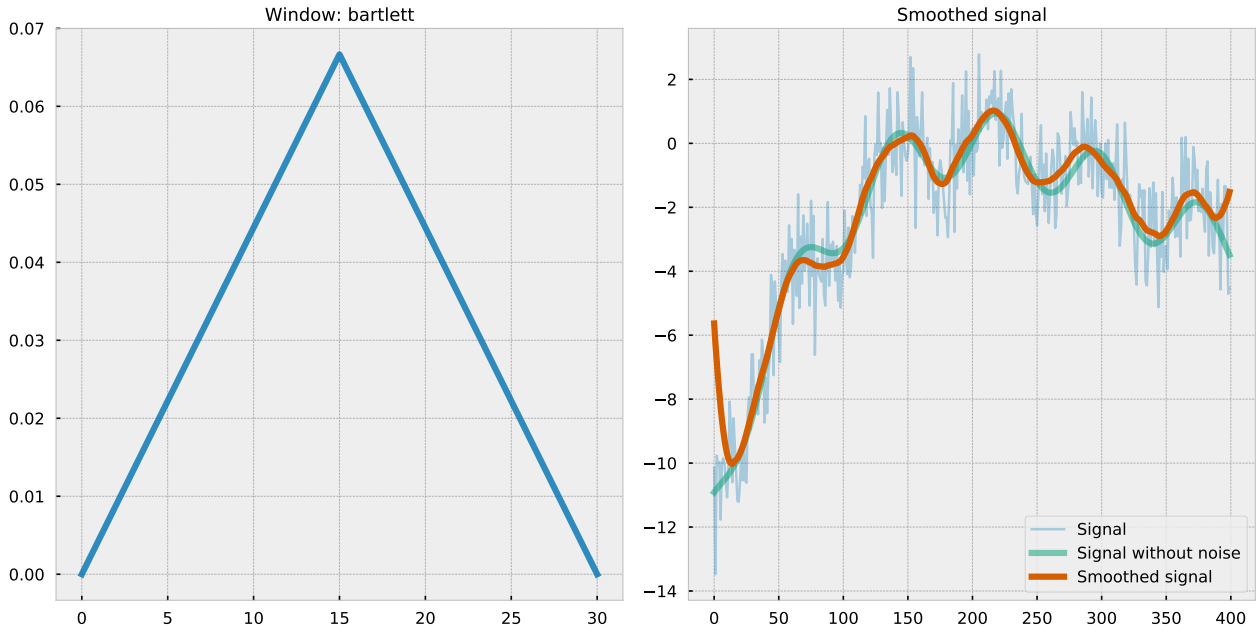

Figure 3: *Smoothing a signal by convolution. (Left) The windowing function used. (Right) The smoothed function.*

The processed signal is given in Fig. 7. Here, we have compared two windows for convolution (the Bartlett[‡] and the Hamming[§] window) with the median filter. We see that the median filter does a better job of removing the spike. The two other windows calculate a weighted average and will thus be affected by the spike. Figure 8 shows the same results where we have zoomed in on the third peak. We see that when we increase the window length, we can remove the spike with convolution, but at a cost: the location of the peak is shifted towards the spike. In conclusion, the median filter is better suited for removing the spike

Exercise 12.5 Multiple Scatter Correction

The Python code for applying the Multiple Scatter Correction can be found in Listing 5. The different spectra and corrections can be found in Fig. 9 and Fig. 10. The calculated sum-of-squares is 30.6 for the centered uncorrected spectra and 6.54 for the centered MSC spectra. We see that applying the MSC reduces the variance in the data, as we would expect (this is reflected in both the calculated $SS0$ and the figures). Finally, when we apply auto-scaling, we largely see that this resembles the MSC spectra, albeit with a larger variance. This is not unexpected, if fact, one can show that MSC and auto-scaling are related by a linear transformation.[¶]

[‡] https://en.wikipedia.org/wiki/Window_function#Triangular_window
[§] https://en.wikipedia.org/wiki/Window_function#Hann_and_Hamming_windows
[¶] See https://doi.org/10.1255/jnirs.30

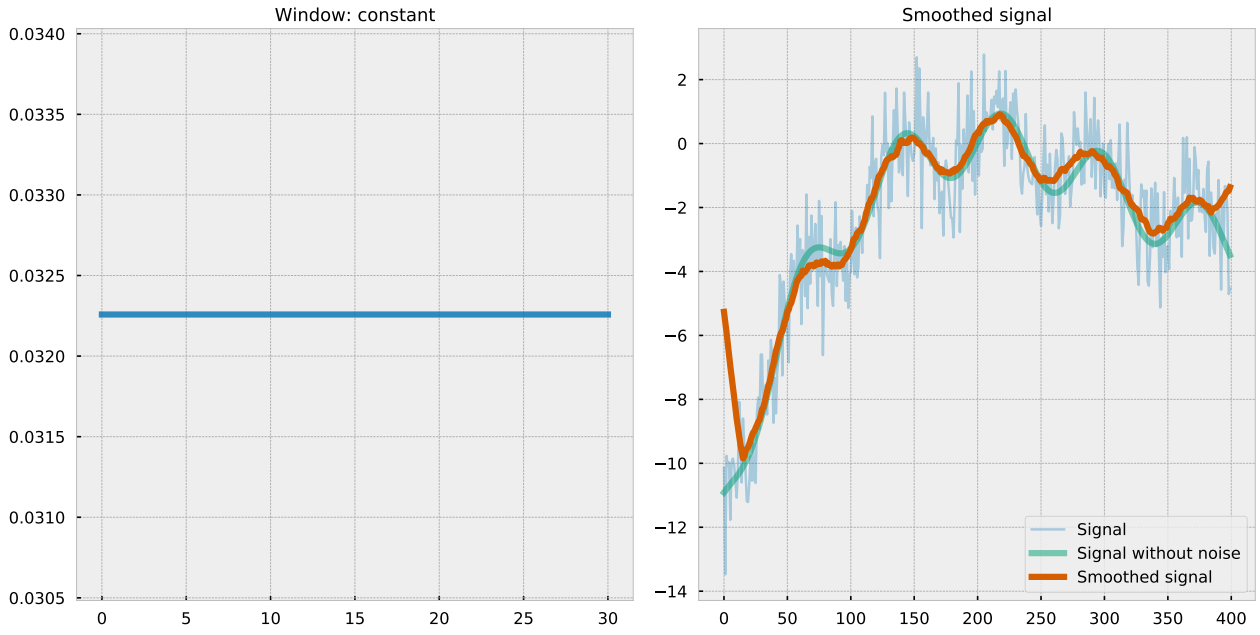

Figure 4: *Smoothing a signal by convolution. (Left) The windowing function used. (Right) The smoothed function.*

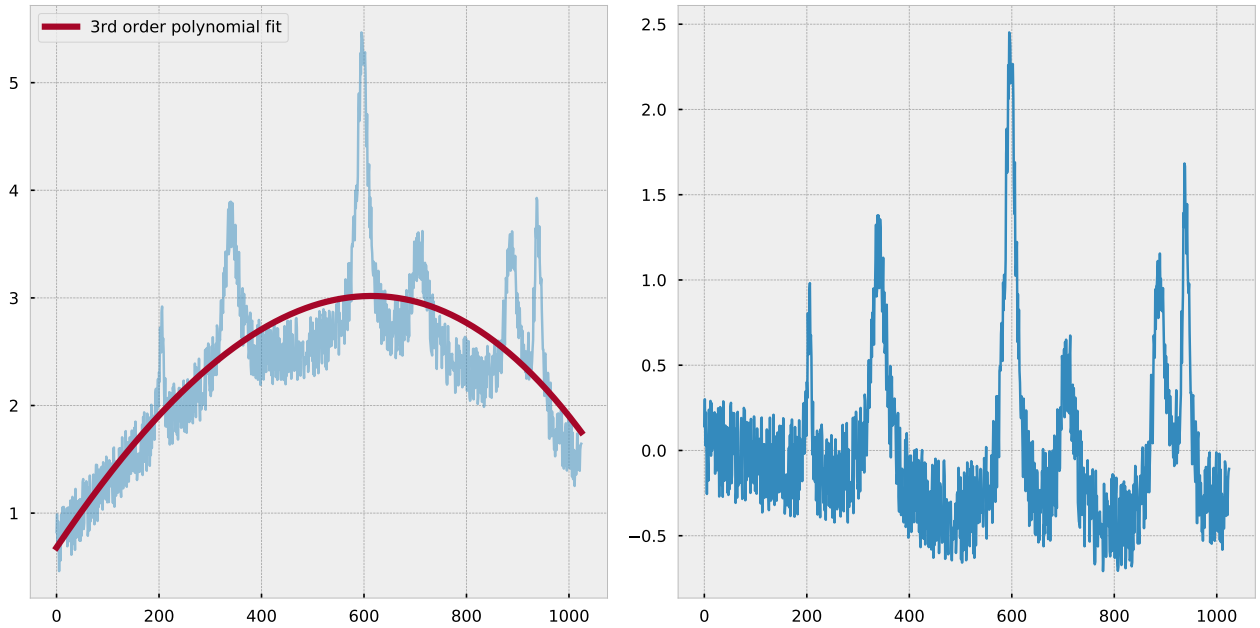

Figure 5: *Removing a trend from a signal. (Left) The third-order polynomial fit to the raw data. (Right) The signal after removing the trend.*

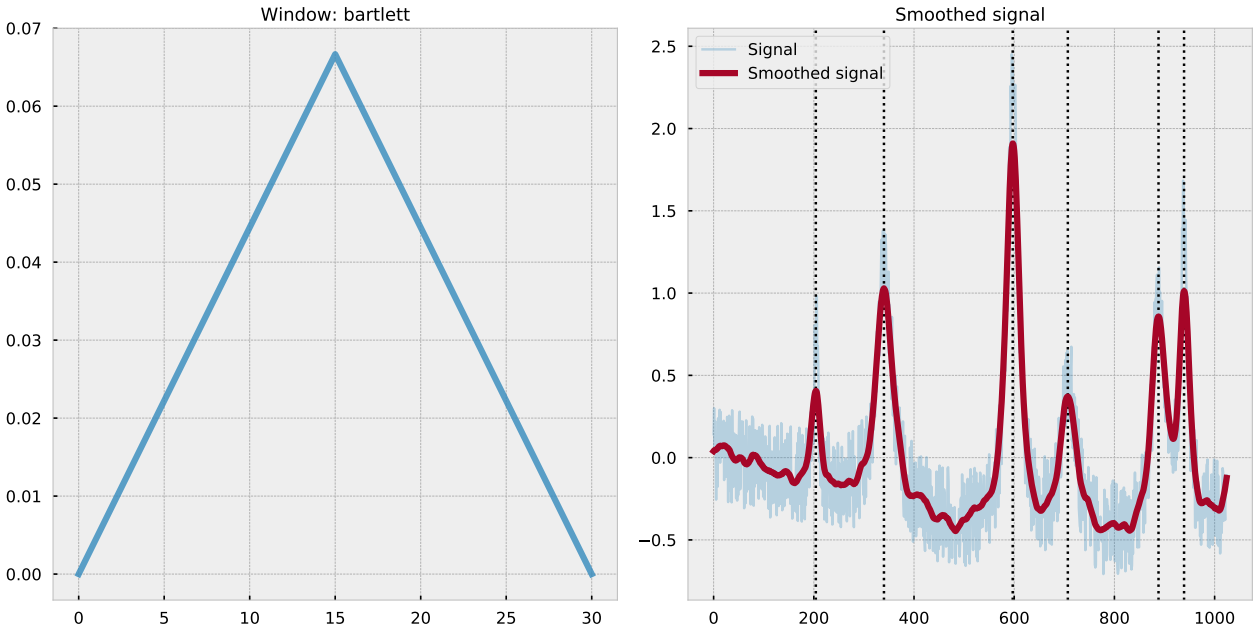

Figure 6: *Smoothing and identification of peaks. (Left) The windowing function used. (Right) The smoothed signal and the locations (vertical dotted lines) of the peaks.*

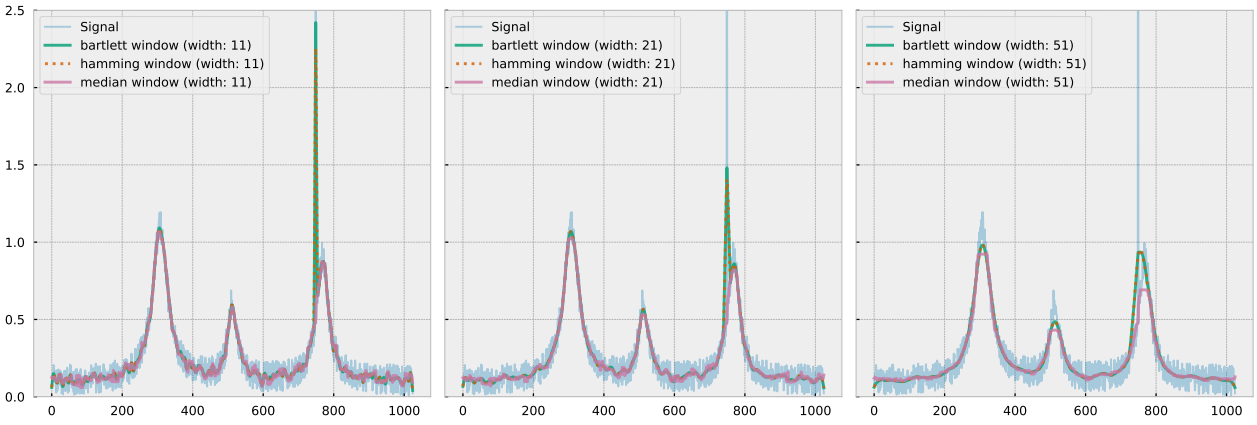

Figure 7: *Removing a spike. (Left) The results when using a window length of 11. (Middle) The results when using a window length of 21. (Right) The results when using a window length of 51.*

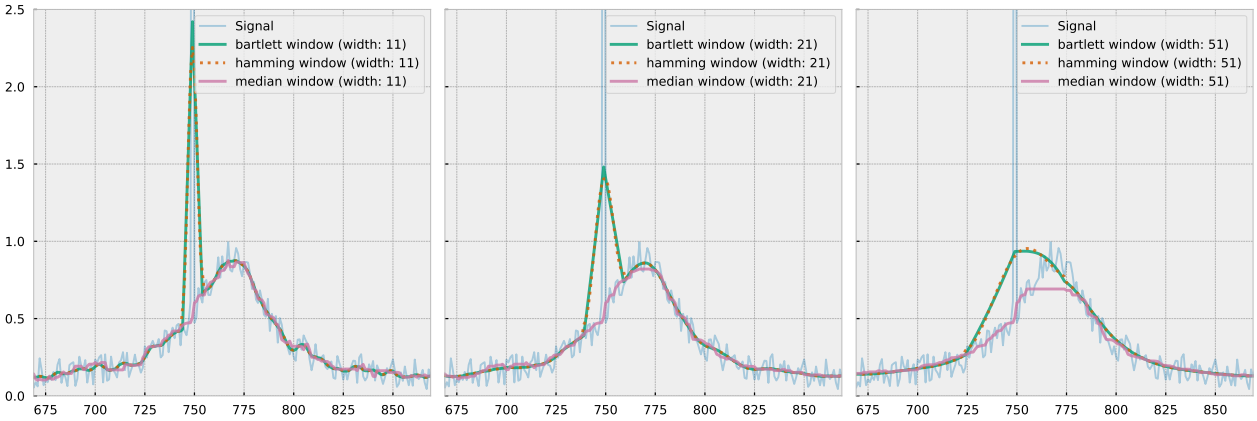

Figure 8: *Removing a spike, zoomed in on the third peak. (Left) The results when using a window length of 11. (Middle) The results when using a window length of 21. (Right) The results when using a window length of 51.*

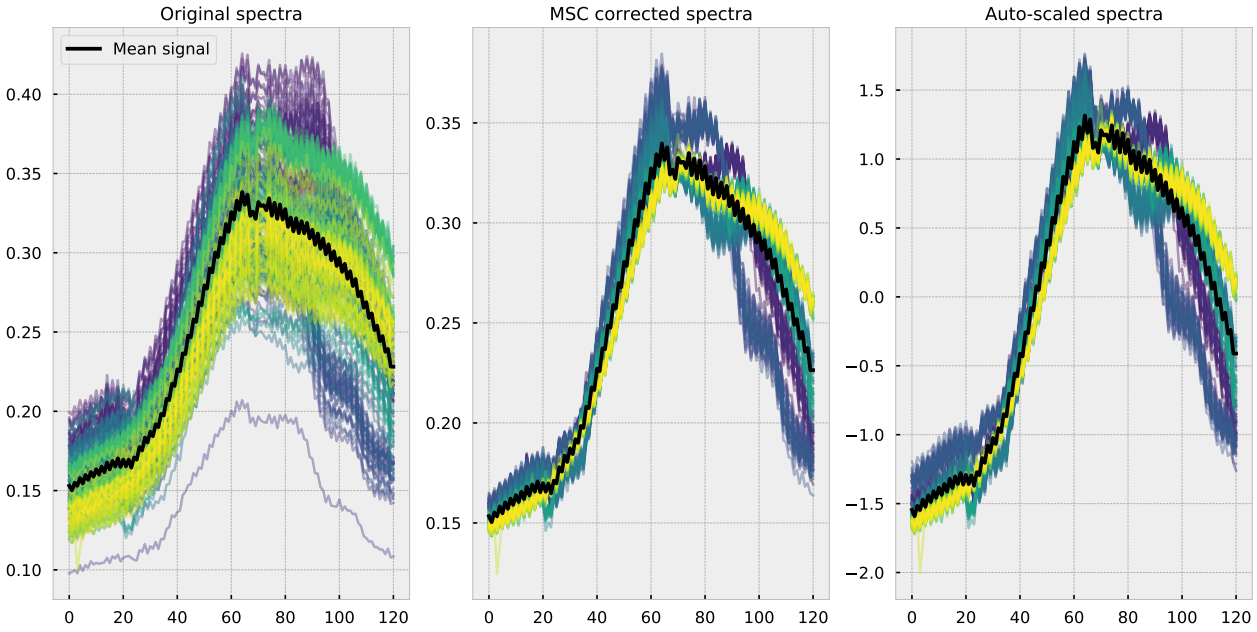

Figure 9: *The effect of applying Multiple Scatter Correction (MSC) and auto-scaling. (Left) The original spectra. (Middle) the MSC corrected spectra. (Right) The auto-scaled spectra.*

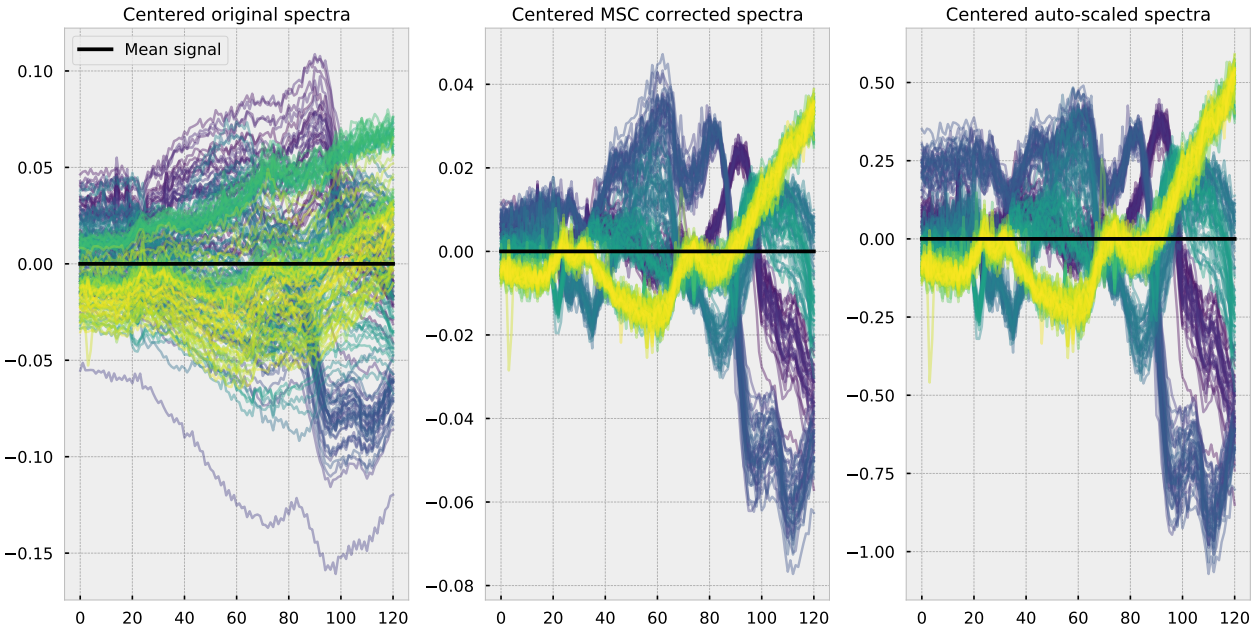

Figure 10: *The effect of applying Multiple Scatter Correction (MSC) and auto-scaling. (Left) The original centered spectra. (Middle) the MSC corrected centered spectra. (Right) The auto-scaled centered spectra.*

Python solutions

```
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
from scipy.signal import savgol_filter
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh', 'seaborn-colorblind'])


def analytical_derivative(t):
    return 8 * np.cos(8 * t) - 3.6 * t + 1.5 * t**2


def filter_signal(data_file, filters):
    data = np.loadtxt(data_file)
    t = data[:, 0]
    delta_t = t[1] - t[0]
    signal = data[:, 1]
    derivative = analytical_derivative(t)
    fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
                                    figsize=(14, 7))
    ax1.set_title('Smoothing')
    ax2.set_title('Derivative')
    ax1.plot(signal, label='Original signal', alpha=0.5)
    ax2.plot(derivative, label='Original signal', alpha=0.5)
    ax1.set_xlabel('Time')
    ax2.set_xlabel('Time')

    for (window, degree) in filters:
        smooth = savgol_filter(signal, window, degree)
        deriv = savgol_filter(signal, window, degree, deriv=1, delta=delta_t)
        ax1.plot(
            smooth,
            label='Savitzky-Golay (window: {}, degree: {})'.format(
                window, degree
            ),
            alpha=0.5,
        )
        ax2.plot(
            deriv,
            label='Savitzky-Golay (window: {}, degree: {})'.format(
                window, degree
            ),
            alpha=0.5,
        )
    ax1.legend()
    ax2.legend()
```

```
def main():
    filters = [(5, 3), (7, 3)]
    filter_signal('Data/signal.txt', filters)
    filters = [(21, 3), (51, 3)]
    filter_signal('Data/signal_noise.txt', filters)
    plt.show()


if __name__ == '__main__':
    main()
```

Listing 1: *Python code for Savitzky–Golay filtering (exercise 12.1).*

```
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
from scipy.signal import savgol_filter
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh', 'seaborn-colorblind'])


def analytical_derivative(t):
    return 8 * np.cos(8 * t) - 3.6 * t + 1.5 * t**2


def constant(length):
    return np.array([1] * length)


def main():
    data0 = np.loadtxt('Data/signal.txt')
    data = np.loadtxt('Data/signal_noise.txt')
    signal0 = data0[:, 1]
    t, signal = data[:, 0], data[:, 1]

    windows = [
        ('constant', constant),
        ('bartlett', np.bartlett),
        ('hanning', np.hanning),
        ('hamming', np.hamming),
        ('blackman', np.blackman),
    ]

    for (name, function) in windows:
        window = function(31)
        window = window / window.sum()
        conv = np.convolve(signal, window, mode='same')
        fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
                                        figsize=(14, 7))
        ax1.set_title('Window: {}'.format(name))
        ax1.plot(window, lw=5, alpha=0.8)
```

```
        ax2.set_title('Smoothed signal')
        ax2.plot(signal, alpha=0.3, label='Signal')
        ax2.plot(signal0, alpha=0.5, lw=5, label='Signal without noise')
        ax2.plot(conv, lw=5, label='Smoothed signal')
        ax2.legend()
        fig1.savefig('smooth_{}'.format(name))

    plt.show()


if __name__ == '__main__':
    main()
```

Listing 2: *Python code for convolution (exercise 12.2).*

```
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
from scipy.signal import find_peaks
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh'])


def main():
    signal = np.loadtxt('Data/peaks.txt')
    t = np.arange(len(signal))

    param = np.polyfit(t, signal, 3)
    trend = np.polyval(param, t)

    signal_trend = signal - trend

    windows = [
        ('bartlett', np.bartlett),
    ]

    fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
                                    figsize=(14, 7))
    ax1.plot(t, signal, alpha=0.5)
    ax1.plot(t, trend, lw=5, label='3rd order polynomial fit')
    ax2.plot(t, signal_trend)
    ax1.legend()

    for (name, function) in windows:
        window = function(31)
        window = window / window.sum()
        conv = np.convolve(signal_trend, window, mode='same')

        peaks = find_peaks(conv, prominence=0.5)[0]

        fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
```

```
                                          figsize=(14, 7))
        ax1.set_title('Window: {}'.format(name))
        ax1.plot(window, lw=5, alpha=0.8)
        ax2.set_title('Smoothed signal')
        ax2.plot(signal_trend, alpha=0.3, label='Signal')

        for i in peaks:
            ax2.axvline(x=i, color='k', ls=':')
            print(i)
        ax2.plot(conv, lw=5, label='Smoothed signal')
        ax2.legend()
        fig1.savefig('remove_trend_{}'.format(name))
    plt.show()


if __name__ == '__main__':
    main()
```

Listing 3: *Python code for exercise 12.3.*

```
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh', 'seaborn-colorblind'])


def median_filter(yvalues, width=11):
    medians = []
    half = width // 2
    length = len(yvalues) - 1
    for i, _ in enumerate(yvalues):
        left = max(0, i - half)
        right = min(length, i + half)
        med = np.median(yvalues[left:right + 1])
        medians.append(med)
    return np.array(medians)


def main():
    signal = np.loadtxt('Data/spike.txt')

    styles = ['-', ':', '-']

    window_length = [11, 21, 51]

    filters = [
        ('bartlett', np.bartlett),
        ('hamming', np.hamming),
    ]
```

```
    all_results = []

    for width in window_length:
        results = []
        for (name, function) in filters:
            window = function(width)
            window = window / window.sum()
            conv = np.convolve(signal, window, mode='same')
            results.append((name, conv))

        # Test with a median filter:
        med = median_filter(signal, width=width)
        results.append(('median', med))
        all_results.append(results)

    fig1, axes1 = plt.subplots(constrained_layout=True,
                               ncols=len(window_length),
                               sharex=True, sharey=True,
                               figsize=(18, 6))
    fig2, axes2 = plt.subplots(constrained_layout=True,
                               ncols=len(window_length),
                               sharex=True, sharey=True,
                               figsize=(18, 6))

    labeltxt = '{} window (width: {})'
    for i, row in enumerate(all_results):
        axi = axes1[i]
        axj = axes2[i]
        for ax1 in (axi, axj):
            ax1.plot(signal, alpha=0.3, label='Signal')
        for j, col in enumerate(row):
            for ax1 in (axi, axj):
                ax1.plot(
                    col[1],
                    label=labeltxt.format(col[0], window_length[i]),
                    ls=styles[j],
                    alpha=0.8,
                    lw=3,
                )
                ax1.legend()
                ax1.set_ylim(0, 2.5)
        axj.set_xlim(669, 869)
    fig1.savefig('remove_spike', bbox_layout='tight')
    fig2.savefig('remove_spike_zoom', bbox_layout='tight')
    plt.show()


if __name__ == '__main__':
    main()
```

Listing 4: *Python code for exercise 12.4.*

```
import numpy as np
import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib.cm import get_cmap
mpl.rcParams["savefig.format"] = 'pdf'
plt.style.use(['seaborn-talk', 'bmh'])


def plot_signals(signals):
    figsize = None if len(signals) == 1 else (14, 7)
    fig1, axes = plt.subplots(constrained_layout=True, ncols=len(signals),
                              figsize=figsize)
    legend = False
    try:
        axes.flatten()
    except AttributeError:
        axes = [axes]
    for (signal, title), axi in zip(signals, axes):
        colors = get_cmap('viridis')(np.linspace(0, 1, signal.shape[0]))
        mean_signal = np.mean(signal, axis=0)
        for i, row in enumerate(signal):
            axi.plot(row, color=colors[i], alpha=0.4)
        axi.plot(mean_signal, color='k', lw=3,
                 label='Mean signal')
        axi.set_title(title)
        if not legend:
            axi.legend()
            legend = True


def main():
    signal = np.loadtxt('Data/nir_msc.txt')
    print(signal.shape)

    plot_signals([(signal, 'Original spectra')])

    mean_row = np.mean(signal, axis=1)
    std_row = np.std(signal, axis=1)
    signal_auto = (signal - mean_row[:, np.newaxis]) / std_row[:, np.newaxis]

    mean_signal = np.mean(signal, axis=0)

    msc = np.zeros_like(signal)

    for i, row in enumerate(signal):
        param = np.polyfit(mean_signal, row, 1)
        msc[i, :] = (row - param[1]) / param[0]

    plot_signals(
        [
```

```
            (signal, 'Original spectra'),
            (msc, 'MSC corrected spectra'),
            (signal_auto, 'Auto-scaled spectra'),
        ]
    )

    msc_centered = msc - np.mean(msc, axis=0)
    signal_centered = signal - mean_signal
    signal_auto_centered = signal_auto - np.mean(signal_auto, axis=0)

    signals = [
        (signal_centered, 'Centered original spectra'),
        (msc_centered, 'Centered MSC corrected spectra'),
        (signal_auto_centered, 'Centered auto-scaled spectra'),
    ]

    plot_signals(signals)

    for signal, label in signals:
        ss0 = np.sum(signal**2)
        print('SS_0 for {}: {}'.format(label, ss0))

    plt.show()


if __name__ == '__main__':
    main()
```

Listing 5: *Python code for multiple scatter correction (exercise 12.4).*