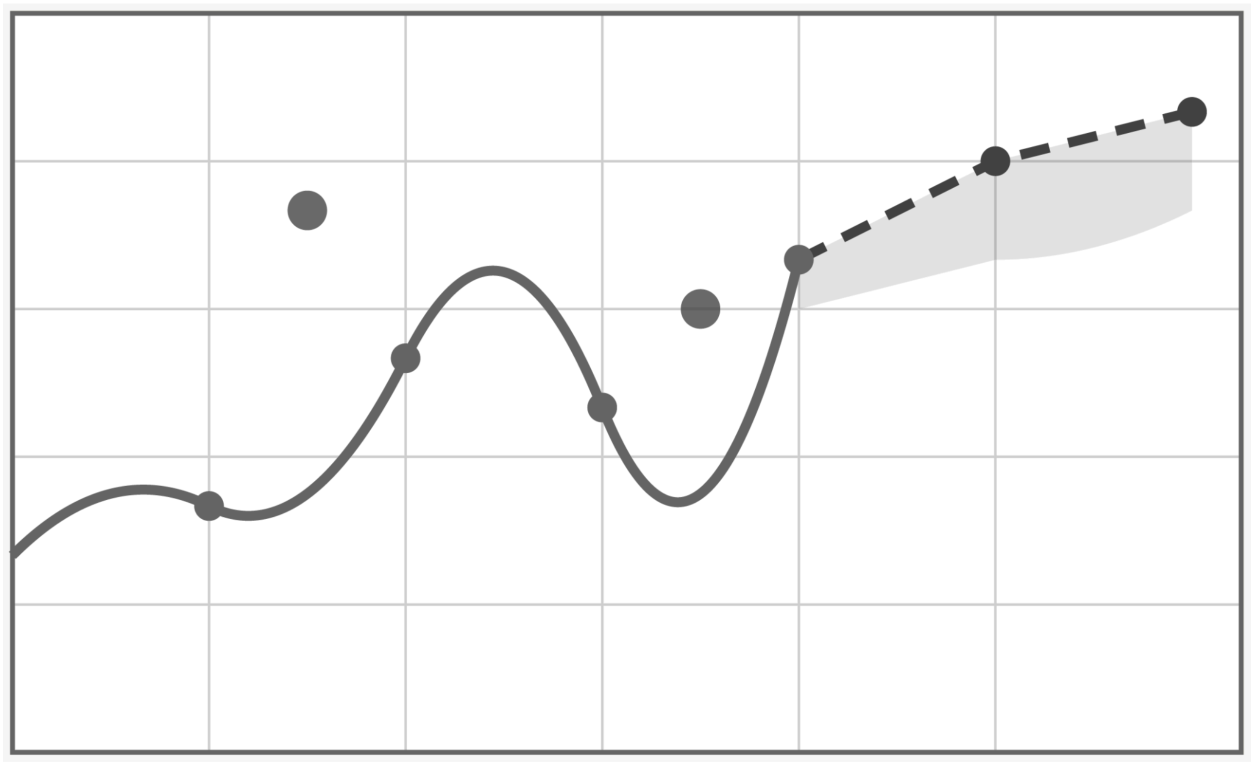
Exercises 1 (Introduction)

DATA.ML.450 Time Series Analysis using Machine Learning (Autumn 2025)



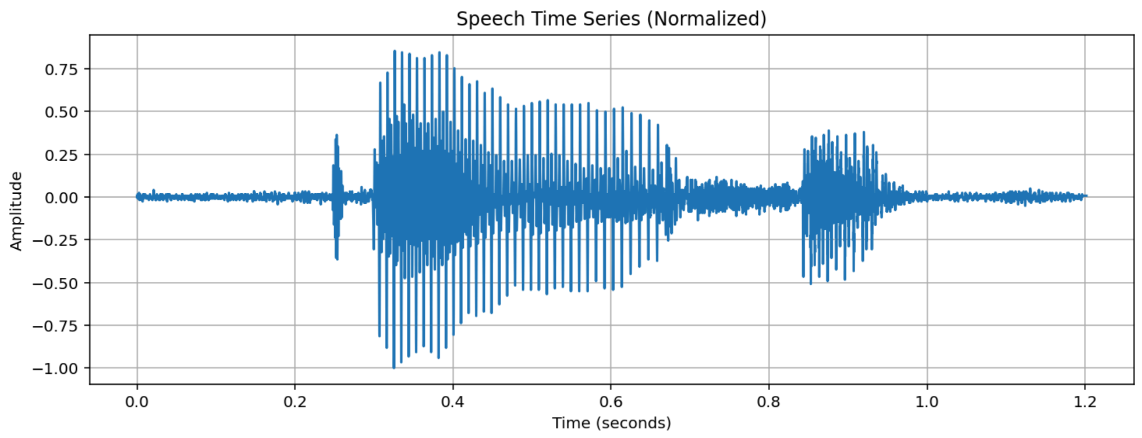
Anas Uddin

14.09.2025

MSc in Data Science

Exercise 1

The given speech signal ‘Kuusi.wav’ has been converted from integer to double precision, mean-centered, and scaled to the range –1 to 1. The purpose of this is to remove bias and ensure consistent amplitude scaling. The plotted time series displays the nonsymmetric waveform of speech, as expected.

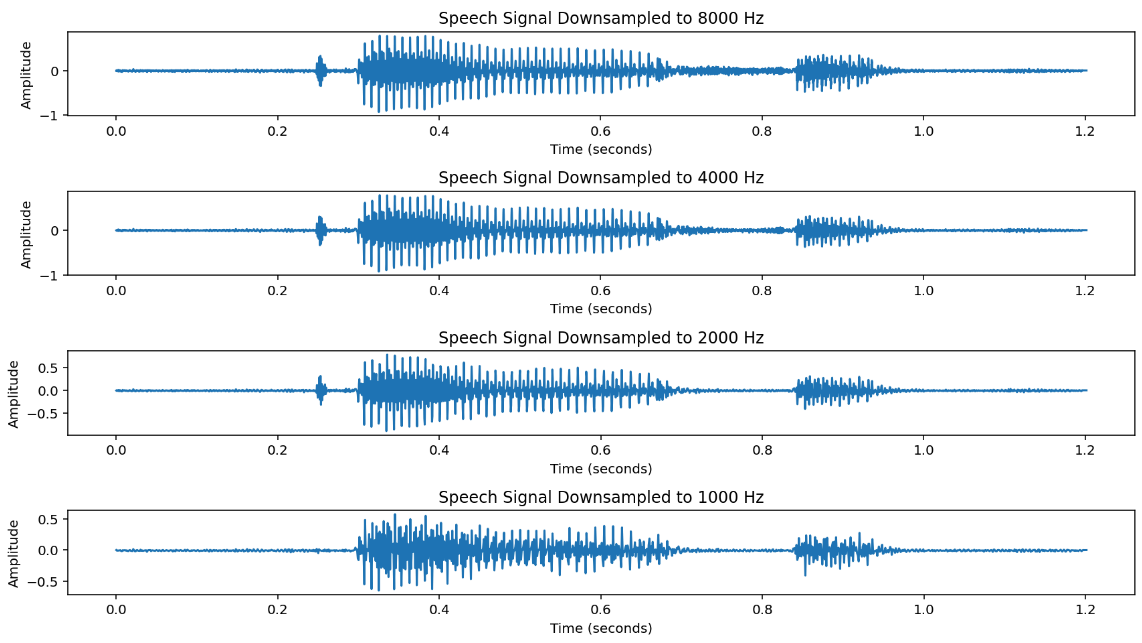


Code:

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile


Fs, y = wavfile.read('./Kuusi.wav')

y = y.astype(np.float64)

y = y - np.mean(y)

y = y / np.max(np.abs(y))


time = np.arange(len(y)) / Fs

plt.figure(figsize=(12, 4))

plt.plot(time, y)

plt.title('Speech Time Series (Normalized)')
plt.xlabel('Time (seconds)')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()
```

Exercise 2

The speech signal was progressively downsampled to lower sampling rates and played back. The waveform was plotted against time in seconds for clarity. After listening tests, it was determined that the speech message remained understandable down to 2000 Hz, but became unclear at 1000 Hz, indicating that rates below 2000 Hz significantly degrade intelligibility.

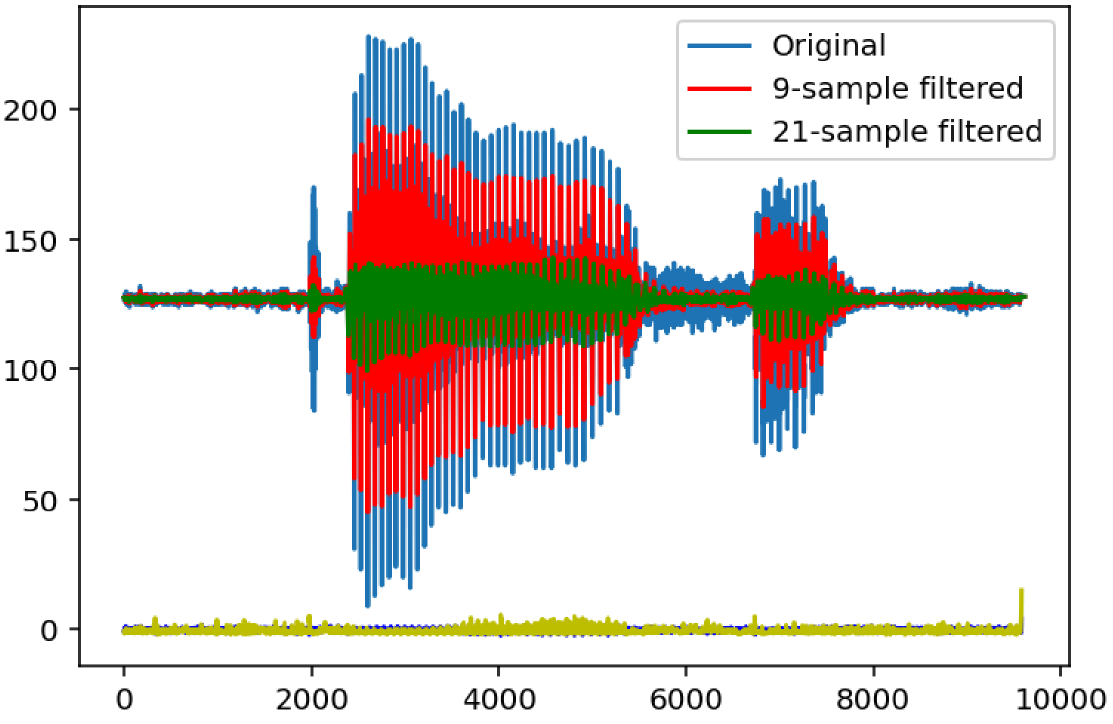


Code:

```
import librosa

import numpy as np

import pygame

import scipy.io.wavfile as wavfile

import tempfile
import os
import matplotlib.pyplot as plt

# Test different sampling rates to find lowest understandable rate
sampling_rates = [8000, 4000, 2000, 1000, 500]
responses = {}

plt.figure(figsize=(12, 8))

for i, sr in enumerate(sampling_rates, start=1):
    print(f"\nTesting sampling rate: {sr} Hz")

    y_low, s = librosa.load("Kuusi.wav", sr=sr)

    audio = y_low * (2**15 - 1) / np.max(np.abs(y_low))
    audio = audio.astype(np.int16)

    temp_file = tempfile.NamedTemporaryFile(suffix=".wav", delete=False)
    wavfile.write(temp_file.name, sr, audio)
    temp_file.close()

    pygame.mixer.init(frequency=sr)
    pygame.mixer.music.load(temp_file.name)
    pygame.mixer.music.play()

    while pygame.mixer.music.get_busy():
        pygame.time.wait(100)
    response = input("Can you understand the message? (y/n): ")
    responses[sr] = response.lower()

    pygame.mixer.quit()
    os.unlink(temp_file.name)

    plt.subplot(len(sampling_rates), 1, i)
    time = np.arange(len(y_low)) / sr
    plt.plot(time, y_low)
    plt.title(f"Speech Signal Downsampled to {sr} Hz")
    plt.xlabel('Time (seconds)')
    plt.ylabel("Amplitude")

    if response.lower() == "n":
        print(f"Message becomes unclear at {sr} Hz")
        break

plt.tight_layout()
plt.savefig("downsampling_results.png")
plt.show()

print("\nUser responses:", responses)
```

Exercise 3

1) The given `average_filter.py` code was modified to load 'Kuusi.wav', compute 'skewness' and 'kurtosis' frame by frame, and plot all results for clear comparison.

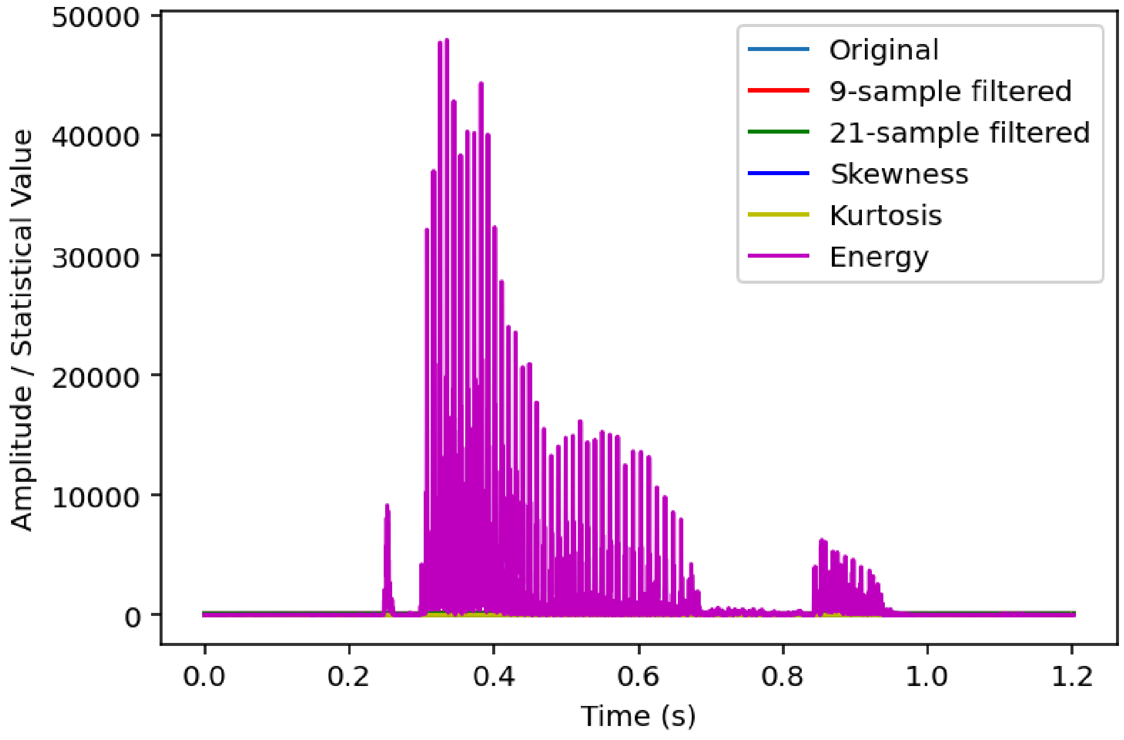


Code:

```
#Jari Turunen, TUNI

import numpy as np

from numpy import cos, sin, pi, absolute, arange, mean

from matplotlib import pyplot as plt

from scipy.stats import skew, kurtosis

from scipy.io import wavfile


fs, data = wavfile.read("Kuusi.wav")

data = data.astype(float)

print(data.shape)
```

```
len1 = 4  #length of the average filter (trend) (1+2*len)=window size
len2 = 10  #longer trend (1+2*len2)
x = data.copy() * 0  #fast initialization
x2 = data.copy() * 0
x3 = data.copy() * 0
x4 = data.copy() * 0
for i in range(len(data)):
    print("%d / %d\n" % (i, len(data)))
    start = i - len1
    if start < 1:  #for initializing the window
        start = 1
    start2 = i - len2
    if start2 < 1:  #for initializing the window2
        start2 = 1

    ending = i + len1
    if ending > len(data):  #taking care of the
        ending = len(data)  #end of the window

    ending2 = i + len2
    if ending2 > len(data):  #taking care of the
        ending2 = len(data)  #end of the window2

    if len(data[start:ending]) < 2:
        x[i] = 0
    else:
        x[i] = np.mean(data[start:ending])  #sliding window mean
    if len(data[start2:ending2]) < 2:
        x2[i] = 0
        x3[i] = 0
        x4[i] = 0
    else:
        x2[i] = np.mean(data[start2:ending2])  #sliding window mean
        x3[i] = skew(data[start2:ending2], axis=0, bias=True)
        x4[i] = kurtosis(data[start2:ending2], axis=0, bias=True)

plt.plot(data)
plt.plot(x, 'r')
plt.plot(x2, 'g')  #plot the results
plt.plot(x3, 'b')
plt.plot(x4, 'y')  #plot the results
plt.legend([
    'Original',
    str(len1 * 2 + 1) + '-sample filtered',
    str(len2 * 2 + 1) + '-sample filtered'
])
plt.show()
```

2) By increasing the window size len1, the energy curve becomes smoother, which reduces rapid fluctuations. Smoothness is important for machine processing as it makes patterns more stable and easier to detect, avoiding noise-driven spikes that could mislead algorithms.



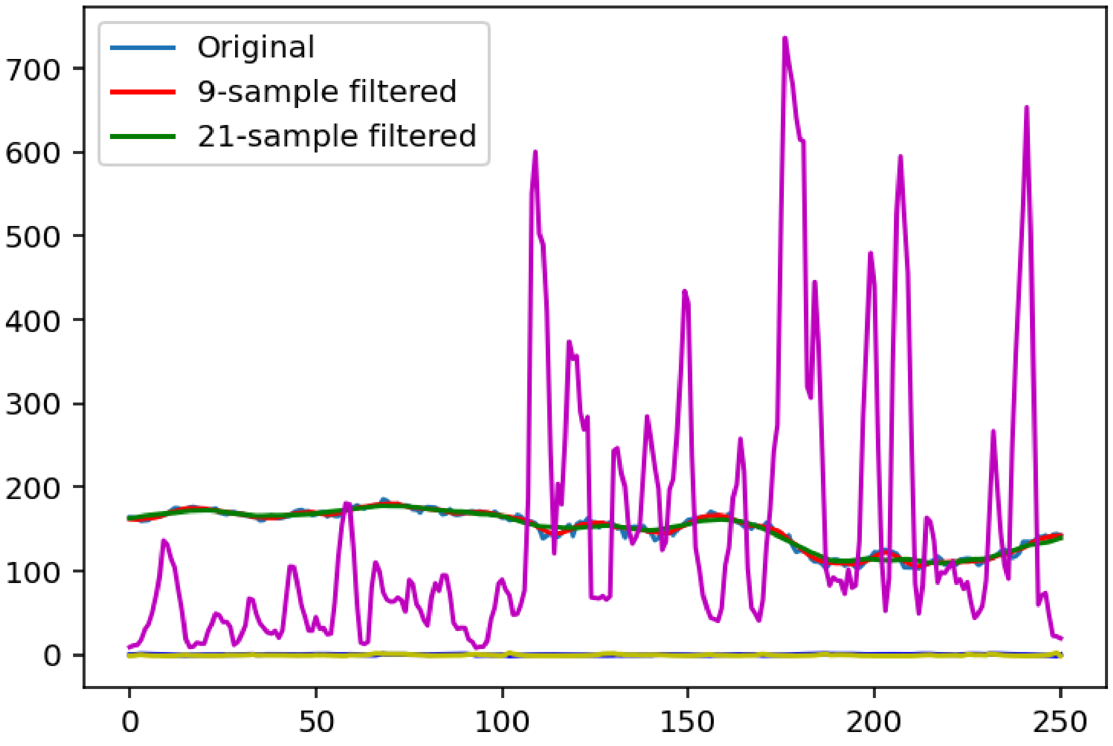
Code:

```
#Jari Turunen, TUNI

import numpy as np

from numpy import cos, sin, pi, absolute, arange, mean

from matplotlib import pyplot as plt

from scipy.stats import skew, kurtosis

from scipy.io import wavfile


fs, data = wavfile.read("Kuusi.wav")

data = data.astype(float)

print(data.shape)


len1 = 4  #length of the average filter (trend) (1+2*len)=window size

len2 = 10  #longer trend (1+2*len2)

x = data.copy() * 0  #fast initialization
```

```
x2 = data.copy() * 0
x3 = data.copy() * 0
x4 = data.copy() * 0
energy = data.copy() * 0
for i in range(len(data)):
    print("%d / %d\n" % (i, len(data)))
    start = i - len1
    if start < 1:  #for initializing the window
        start = 1
    start2 = i - len2
    if start2 < 1:  #for initializing the window2
        start2 = 1

    ending = i + len1
    if ending > len(data):  #taking care of the
        ending = len(data)  #end of the window

    ending2 = i + len2
    if ending2 > len(data):  #taking care of the
        ending2 = len(data)  #end of the window2

    if len(data[start:ending]) < 2:
        x[i] = 0
        energy[i] = 0
    else:
        window_data = data[start:ending]
        x[i] = np.mean(data[start:ending])  #sliding window mean
        energy[i] = np.sum((window_data - np.mean(window_data))**2)

    if len(data[start2:ending2]) < 2:

        x2[i] = 0

        x3[i] = 0

        x4[i] = 0

    else:

        x2[i] = np.mean(data[start2:ending2])  #sliding window mean

        x3[i] = skew(data[start2:ending2], axis=0, bias=True)

        x4[i] = kurtosis(data[start2:ending2], axis=0, bias=True)


plt.plot(data)

plt.plot(x, 'r')

plt.plot(x2, 'g')  #plot the results

plt.plot(x3, 'b')

plt.plot(x4, 'y')  #plot the results

plt.plot(energy, 'm')

plt.legend([

    'Original',

    str(len1 * 2 + 1) + '-sample filtered',

    str(len2 * 2 + 1) + '-sample filtered'

])

plt.show()
```

3) ‘Skewness’ measures the asymmetry of a distribution. It shows whether values lean more to the left or right of the mean. On the other hand, ‘kurtosis’ measures the ‘peakedness’ or heaviness of the tails compared to a normal distribution. Energy reflects the overall power or variability in a window of the signal. In time series analysis, these features can be used as inputs for classification (for example, distinguishing signal types or patterns) or regression (for example, predicting continuous outcomes from signal behavior). Lastly, the Python function ‘moment’ computes the k-th central moment of a dataset, forming the basis for higher-order statistics like ‘variance’ (2nd moment), ‘skewness’ (3rd), and ‘kurtosis’ (4th).

4) The code has been modified again to analyze Amazon stock data. The current window size for the energy calculation is 9 samples ((2 * len1) + 1 = (2 * 4) + 1), which should provide a reasonably smooth energy curve for tracking stock price fluctuations. For a smoother energy curve, len1 can be increased to 10 - 20 for window sizes of 21 - 41 samples.



Code:

```
#Jari Turunen, TUNI

import numpy as np

from numpy import cos, sin, pi, absolute, arange, mean

from matplotlib import pyplot as plt

from scipy.stats import skew, kurtosis

import pandas as pd


df = pd.read_pickle("AMZN.pkl")

data = df['Open'].values.astype(float)
```

```
print(data.shape)

len1 = 4  #length of the average filter (trend) (1+2*len)=window size
len2 = 10  #longer trend (1+2*len2)
x = data.copy() * 0  #fast initialization
x2 = data.copy() * 0
x3 = data.copy() * 0
x4 = data.copy() * 0
energy = data.copy() * 0
for i in range(len(data)):
    print("%d / %d\n" % (i, len(data)))
    start = i - len1
    if start < 1:  #for initializing the window
        start = 1
    start2 = i - len2
    if start2 < 1:  #for initializing the window2
        start2 = 1

    ending = i + len1
    if ending > len(data):  #taking care of the
        ending = len(data)  #end of the window

    ending2 = i + len2
    if ending2 > len(data):  #taking care of the
        ending2 = len(data)  #end of the window2

    if len(data[start:ending]) < 2:
        x[i] = 0
        energy[i] = 0
    else:
        window_data = data[start:ending]
        x[i] = np.mean(data[start:ending])  #sliding window mean
        energy[i] = np.sum((window_data - np.mean(window_data))**2)
    if len(data[start2:ending2]) < 2:
        x2[i] = 0
        x3[i] = 0
        x4[i] = 0
    else:
        x2[i] = np.mean(data[start2:ending2])  #sliding window mean
        x3[i] = skew(data[start2:ending2], axis=0, bias=True)
        x4[i] = kurtosis(data[start2:ending2], axis=0, bias=True)

plt.plot(data)
plt.plot(x, 'r')
plt.plot(x2, 'g')  #plot the results
plt.plot(x3, 'b')
plt.plot(x4, 'y')  #plot the results
plt.plot(energy, 'm')
plt.legend([
    'Original',
    str(len1 * 2 + 1) + '-sample filtered',
    str(len2 * 2 + 1) + '-sample filtered'
])
plt.show()
```

Exercise 4

1) **What is your own opinion about machine learning? What does it contain?**
   Ans: I see machine learning as a way to let algorithms learn patterns from data and make predictions without explicitly programming every possibility and rule. It ranges from simple methods like linear regression to complex models that handle high-dimensional, non-linear data. I find it fascinating to see its ability to generalize from what it has learned to new, unseen situations.
2) **Why are neural networks having so a dominant role in machine learning?**
   Ans: Neural networks, in my opinion, are dominant because they can model complex nonlinear relationships through layers and activation functions. They automatically learn hierarchical features, and with modern computing power and large datasets, they outperform traditional methods in many tasks. I also find it fascinating that, in theory, they can approximate any continuous function.
3) **Why are there so many architectures in neural networks?**
   Ans: From my perspective, different neural network architectures exist because different problems and data types need different tools. For example, I would use CNNs for images, RNNs for sequences, transformers for language, and autoencoders for dimensionality reduction. Each architecture has built-in assumptions that make it effective for certain jobs.
4) **How is the architecture decided?**
   Ans: When choosing an architecture, one should consider the problem type, data structure, and available computational resources, and usually rely on a mix of domain knowledge, experimentation, and insights from recent research that has been done.