unit_test

April 1, 2023

1 Test Your Algorithm

1.1 Instructions

1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:

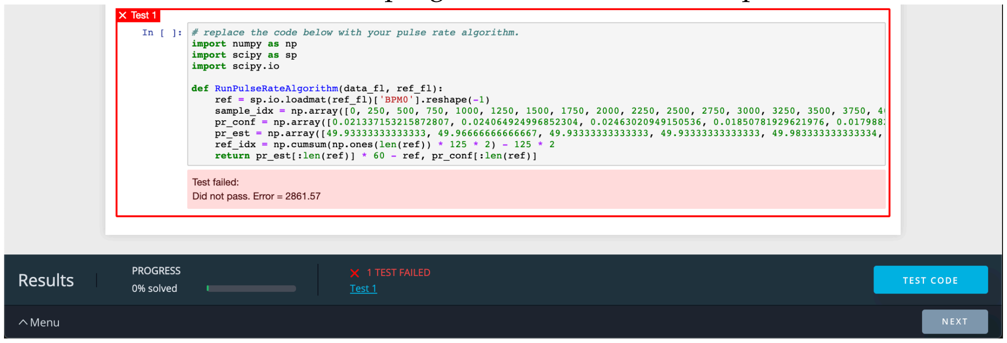
- Copy over all the **Code** section to the following Code block.
- Download as a Python (`.py`) and copy the code to the following Code block.

2. In the bottom right, click the Test Run button.

1.1.1 Didn't Pass

If your code didn't pass the test, go back to the previous Concept or to your local setup and continue iterating on your algorithm and try to bring your training error down before testing again.

1.1.2 Pass

If your code passes the test, complete the following! You **must** include a screenshot of your code and the Test being **Passed**. Here is what the starter filler code looks like when the test is run and should be similar. A passed test will include in the notebook a green outline plus a box with **Test passed:** and in the Results bar at the bottom the progress bar will be at 100% plus a checkmark with **All cells passed**. 

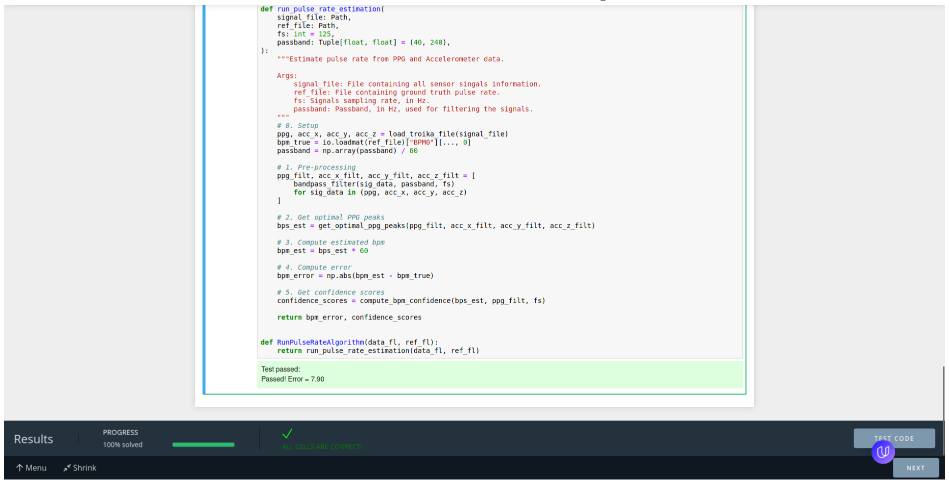
```
# replace the code below with your pulse rate algorithm.
import numpy as np
import scipy as sp
import scipy.io

def RunPulseRateAlgorithm(data_fl, ref_fl):
    ref = sp.io.loadmat(ref_fl)['BPM0'].reshape(-1)
    sample_idx = np.array([0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4
    pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.024630209491505336, 0.018507819296219776, 0.017988
    pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.93333333333333, 49.983333333333334,
    ref_idx = np.cumsum(np.ones(len(ref)) * 125 * 2) - 125 * 2
    return pr_est[:len(ref)] * 60 - ref, pr_conf[:len(ref)]
```

```
Test failed:
Did not pass. Error = 2861.57
```

1. Take a screenshot of your code passing the test, make sure it is in the format `.png`. If not a `.png` image, you will have to edit the Markdown render the image after Step 3. Here is an example of what the `passed.png` would look like
2. Upload the screenshot to the same folder or directory as this jupyter notebook.

3. Rename the screenshot to `passed.png` and it should show up below.



```
def run_pulse_rate_estimation(
    signal_file: Path,
    ref_file: Path,
    fs: int = 125,
    passband: Tuple[float, float] = (40, 240),
):
    """Estimate pulse rate from PPG and Accelerometer data.

    Args:
        signal_file: File containing all sensor singals information.
        ref_file: File containing ground truth pulse rate.
        fs: Signals sampling rate, in Hz.
        passband: Passband, in Hz, used for filtering the signals.
    """
    # 0. Setup
    ppg, acc_x, acc_y, acc_z = load_troika_file(signal_file)
    bpm_true = io.loadmat(ref_file)["BPM0"][..., 0]
    passband = np.array(passband) / 60

    # 1. Pre-processing
    ppg_filt, acc_x_filt, acc_y_filt, acc_z_filt = [
        bandpass_filter(sig_data, passband, fs)
        for sig_data in (ppg, acc_x, acc_y, acc_z)
    ]

    # 2. Get optimal PPG peaks
    bps_est = get_optimal_ppg_peaks(ppg_filt, acc_x_filt, acc_y_filt, acc_z_filt)

    # 3. Compute estimated bpm
    bpm_est = bps_est * 60

    # 4. Compute error
    bpm_error = np.abs(bpm_est - bpm_true)

    # 5. Get confidence scores
    confidence_scores = compute_bpm_confidence(bps_est, ppg_filt, fs)

    return bpm_error, confidence_scores


def RunPulseRateAlgorithm(data_fl, ref_fl):
    return run_pulse_rate_estimation(data_fl, ref_fl)
```

```
Test passed:
Passed! Error = 7.90
```

4. Download this jupyter notebook as a `.pdf` file.
5. Continue to Part 2 of the Project.

```
In [2]: from copy import deepcopy
        from pathlib import Path
        from typing import Iterable, Tuple

        import numpy as np
        import pandas as pd
        from scipy import io, signal
        from matplotlib import pyplot as plt


        def load_troika_file(file_path: Path) -> np.array:
            """
            Loads and extracts signals from a troika data file.

            Args:
                data_fl: (str) filepath to a troika .mat file.

            Returns:
                Numpy arrays for ppg, accx, accy, accz signals.
            """
            return io.loadmat(file_path)["sig"][2:]


        def bandpass_filter(
            signal_data: np.array, passband: Tuple[float, float] = (2 / 3, 4), fs: int = 125
        ) -> np.array:
            """Bandpass filter the signal bfor a given passband

            Args:
                signal: Signal data to filter.
                passband: Tuple of lower and upper bound frequencies (in Hz).
        fs: Sampling rate (in Hz).

    Returns:
        Filtered signal whose frequencies are within the passband.
    """
    b, a = signal.butter(3, passband, btype="bandpass", fs=fs)
    return signal.filtfilt(b, a, signal_data)


def get_optimal_ppg_peaks(
    ppg: np.array,
    acc_x: np.array,
    acc_y: np.array,
    acc_z: np.array,
    fs: int = 125,
    top_k: int = 7,
) -> np.array:
    """Compute optimal PPG peaks from PPG and Accelerometer signals by avoiding
        spurious correlations.

    Args:
        ppg: PPG signal.
        acc_x: Accelerometer signal (X axis).
        acc_y: Accelerometer signal (Y axis).
        acc_z: Accelerometer signal (Z axis).
        fs: Signals sampling rate, in Hz.
        top_k: Select top K frequency candidates.

    Returns:
        Optimal PPG peaks.
    """
    # 1. Compute spectrograms for each signal
    (
        (ppg_spec, ppg_freqs, _, _),
        (acc_x_spec, acc_x_freqs, _, _),
        (acc_y_spec, acc_y_freqs, _, _),
        (acc_z_spec, acc_z_freqs, _, _),
    ) = [
        plt.specgram(signal, Fs=fs, NFFT=8 * fs, noverlap=6 * fs)
        for signal in (ppg, acc_x, acc_y, acc_z)
    ]
    plt.close("all")

    # 2. Compute best frequency estimate for each window
    ppg_peaks = []
    for ppg_win, acc_x_win, acc_y_win, acc_z_win in zip(
        ppg_spec.T, acc_x_spec.T, acc_y_spec.T, acc_z_spec.T
    ):
        # 2.1. Compute max frequency component of each accelerometer signal
        acc_x_max_freq = acc_x_freqs[np.argmax(acc_x_win)]
        acc_y_max_freq = acc_y_freqs[np.argmax(acc_y_win)]
        acc_z_max_freq = acc_z_freqs[np.argmax(acc_z_win)]

        # 2.2. Get top K PPG frequencies according to their magnitude
        ppg_freq_sorted = ppg_freqs[ppg_win.argsort()[::-1]][:top_k]

        # 2.3. Discard those frequencies present in the accelerometer
        ppg_freq_filt = ppg_freq_sorted[
            ~(
                (ppg_freq_sorted == acc_x_max_freq)
                | (ppg_freq_sorted == acc_y_max_freq)
                | (ppg_freq_sorted == acc_z_max_freq)
            )
        ]

        # 2.4. Select optimal PPG peak
        if len(ppg_peaks) == 0:
            ppg_peaks.append(ppg_freq_filt[0])
        else:
            # select closest to previous estimate
            ppg_peaks.append(
                ppg_freq_filt[np.abs(ppg_freq_filt - ppg_peaks[-1]).argmin()]
            )

    return np.array(ppg_peaks)


def compute_bpm_confidence(
    bps_estimates: np.array,
    ppg: np.array,
    fs: int = 125,
) -> np.array:
    """
    Compute BPM estimates confidence scores by summing up the frequency spectrum within
        a window around the pulse rate estimate and dividing it by the sum of the entire
        spectrum. Use first harmonic as window size.

    Args:
        bps_estimates: PPG peaks, which are the BPM estimates in seconds.
        ppg: PPG signal.
        fs: Signals sampling rate, in Hz.

    Returns:
        Confidence scores.
    """
    # 0. PPG Spectrogram
```

```python
    ppg_spec, ppg_freqs, _, _ = plt.specgram(ppg, Fs=fs, NFFT=8 * fs, noverlap=6 * fs)
    plt.close("all")

    # 1. Compute confidence per estimated peak
    confs = []
    for bps_est, ppg_win in zip(bps_estimates, ppg_spec.T):
        # 1.1. Get frequencies around estimate (within window)
        lower_bound = bps_est - (bps_est * 2)
        upper_bound = bps_est + (bps_est * 2)
        conf_win = (lower_bound < ppg_freqs) & (ppg_freqs < upper_bound)

        # 1.2. Compute confidence
        confs.append(np.sum(ppg_win[conf_win]) / np.sum(ppg_win))

    return np.array(confs)


def run_pulse_rate_estimation(
    signal_file: Path,
    ref_file: Path,
    fs: int = 125,
    passband: Tuple[float, float] = (40, 240),
):
    """Estimate pulse rate from PPG and Accelerometer data.

    Args:
        signal_file: File containing all sensor singals information.
        ref_file: File containing ground truth pulse rate.
        fs: Signals sampling rate, in Hz.
        passband: Passband, in Hz, used for filtering the signals.
    """
    # 0. Setup
    ppg, acc_x, acc_y, acc_z = load_troika_file(signal_file)
    bpm_true = io.loadmat(ref_file)["BPM0"][..., 0]
    passband = np.array(passband) / 60

    # 1. Pre-processing
    ppg_filt, acc_x_filt, acc_y_filt, acc_z_filt = [
        bandpass_filter(sig_data, passband, fs)
        for sig_data in (ppg, acc_x, acc_y, acc_z)
    ]

    # 2. Get optimal PPG peaks
    bps_est = get_optimal_ppg_peaks(ppg_filt, acc_x_filt, acc_y_filt, acc_z_filt)

    # 3. Compute estimated bpm
    bpm_est = bps_est * 60
```

```
        # 4. Compute error
        bpm_error = np.abs(bpm_est - bpm_true)

        # 5. Get confidence scores
        confidence_scores = compute_bpm_confidence(bps_est, ppg_filt, fs)

        return bpm_error, confidence_scores


    def RunPulseRateAlgorithm(data_fl, ref_fl):
        return run_pulse_rate_estimation(data_fl, ref_fl)
```

In [ ]: