Výpočet tepové frekvence z EKG signálu

Vypracovali Marek Tremel, Radek Novák a Kseniia Mahalias

Zadání : Ve zdrojové databázi najdete celkem 18 měření EKG signálu pro různé věkové skupiny. Signál obsahuje různé anomálie a nemusí být vždy centralizován podle vodorovné osy. EKG signál obsahuje dominantní peaky, které se nazývají R vrcholy. Vzdálenost těchto vrcholů určuje dobu mezi jednotlivými tepy. Počet tepů za minutu je tedy počet R vrcholů v signálu o délce jedné minuty. Navrhněte algoritmus, který bude automaticky detekovat počet R vrcholů v EKG signálech a prezentujte tepovou frekvenci při jednotlivých jízdách/měřeních. Vás algoritmus následně otestujte na databázi MIT-BIH https://physionet.org/content/nsrdb/1.0.0/ a prezentujte jeho úspěšnost vzhledem k anotovaným datům z databáze.

Použité knihovny

```
import pandas as pd
import wfdb
import os
import numpy as np
from scipy.signal import butter, filtfilt
```

Parametry pro nastavení zpracování dat

```
sampfrom = 0  # Počáteční index vzorku
nsamp = 100000  # Počet vzorků
lowcut = 0.5  # Dolní hranice pásmové filtrace (Hz)
highcut = 2.211  # Horní hranice pásmové filtrace (Hz)
filter_order = 4  # Pořadí filtru
threshold = 0.0001  # Práh pro detekci R vrcholů
```

Pásmová filtrace

Proces, který propouští signály v určitém frekvenčním rozsahu a zároveň potlačuje signály mimo toto pásmo.

1. Výpočet Nyquistovy frekvence: Nyquistova frekvence je polovina vzorkovací frekvence (fs). Je to maximální frekvence, kterou lze správně reprezentovat při dané vzorkovací frekvenc
2. Normalizace mezních frekvencí: Dolní a horní mezní frekvence jsou normalizovány vzhledem k Nyquistově frekvenci.
3. Funkce *butter* z knihovny *scipy.signal* navrhne Butterworthův filtr s daným řádem a normalizovanými mezními frekvencemi. Výstupem jsou koeficienty filtru *b* a *a*

4. Funkce *filtfilt* z knihovny *scipy.signal* aplikuje navržený filtr na vstupní signál. Tato funkce provádí obousměrné filtrování, což znamená, že signál je filtrován dopředu i dozadu, aby se minimalizovalo fázové zkreslení.

```
def bandpass_filter(signal, lowcut, highcut, fs, order):
    """Funkce pro pásmovou filtrační metodu

    Args:
        signal: vstupní signál, který má být filtrován.
        lowcut: dolní mez frekvenčního pásma
        highcut: horní mez frekvenčního pásma.
        fs: vzorkovací frekvence signálu.
        order: řád filtru, který určuje strmost filtru.

    Returns:
        Filtrovaný signál.
    """
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype="band")
    return filtfilt(b, a, signal)
```

Funkce pro výpočet tepové frekvence (BPM) z detekovaných R vrcholů

Pokud je detekováno více než jedno maximum (`peaks.size > 1`)

1. Spočítá se rozdíl mezi sousedními indexy (`np.diff(peaks)`), což odpovídá počtu vzorků mezi R-vrcholy.
2. Tím, že se rozdíly vydělí vzorkovací frekvencí (`sampling_frequency`), získají se časové intervaly mezi jednotlivými údery srdce v sekundách.
3. Průměrný interval mezi R-vrcholy se použije k výpočtu tepové frekvence

```
def calculate_heart_rate(peaks, sampling_frequency):
    """Funkce pro výpočet tepové frekvence (BPM) z detekovaných R
vrcholů.

    Args:
        peaks: indexy detekovaných R vrcholů.

    Returns:
        Tepová frekvence v BPM.
    """
    if peaks.size > 1:
        # Výpočet intervalů mezi R vrcholy
        # np.diff(peaks) vypočítá rozdíly mezi po sobě jdoucími indexy
vrcholů, což představuje počet vzorků mezi vrcholy.
        # Dělením vzorkovací frekvencí (sampling_frequency) se tyto
```

```
rozdíly převedou na časové intervaly v sekundách.
        r_peak_intervals = np.diff(peaks) / sampling_frequency
        # Výpočet průměrné tepové frekvence:
        # np.mean(r_peak_intervals) vypočítá průměrný interval mezi R
vrcholy.
        # Tepová frekvence (BPM) je rovna 60 děleno průměrným
intervalem mezi R vrcholy.
        # Pokud nebyly detekovány žádné R vrcholy, je tepová frekvence
nastavena na 0.
        return 60 / np.mean(r_peak_intervals) if r_peak_intervals.size
> 0 else 0
    else:
        return 0
```

1. Ze složky se naleznout unikátní názvy souborů. Podmínka ošetří unikátnost a valididní soubory s daty.
2. Zpracování každého souboru
   - Načte EKG signál a jeho parametry pomocí `wfdb.rdsamp()`.
   - Načte referenční R-vrcholy pomocí `wfdb.rdann()`.
   - Aplikuje se `pásmová filtrace`
   - Vypočítá se derivace signálu
   - Určí nulové průchody v derivaci signálu a vyfiltruje je podle nastaveného prahu.
3. Výpočítá se BPM pomocí funkce `calculate_heart_rate()` pro detekované i referenční R-vrcholy.
4. Vytvoření DataFrame a výpočet přesnosti.
5. Vrací DataFrame s výsledky pro další analýzu.

```
def get_data():
    # Načtení dat
    data: list = []

    # Načtení názvů souborů
    filenames = set()
    main_directory = "../data/real_ekg"
    for idx, file in enumerate(os.listdir(main_directory)):
        # if idx == 20:
        #     break
        if file.endswith(".dat"):
            filenames.add(file.split(".")[0])

    print(f"Počet souborů: {len(filenames)}")
    # Procházení souborů
    for name in filenames:
        # Načtení signálu a informací o signálu
        record_path = os.path.join(main_directory, name)

        # Načtení signálu a informací o signálu
        record = wfdb.rdsamp(record_path, sampfrom=sampfrom,
```

```
sampto=sampfrom + nsamp)

        # Načtení referenčních R vrcholů
        true_peaks = wfdb.rdann(
            record_path, extension="atr", sampfrom=sampfrom,
sampto=sampfrom + nsamp
        ).sample

        # Signál
        signal = record[0][:, 0]
        fields = record[1]

        # Vzorkovací frekvence (Hz)
        sampling_frequency = fields["fs"]

        # Normalizace signálu
        # Tak, aby měl nulový průměr a jednotkovou směrodatnou
odchylku
        signal_normalized = (signal - np.mean(signal)) /
np.std(signal)

        # Aplikace filtrace
        try:
            signal_processed = bandpass_filter(
                signal_normalized, lowcut, highcut,
sampling_frequency, filter_order
            )
        except ValueError as e:
            print(f"Chyba filtrace: {e}")
            signal_processed = signal_normalized

        # Derivace signálu je užitečná pro detekci změn v signálu,
jako jsou vrcholy a průchody nulou.
        # V kontextu EKG signálu se derivace používá k detekci R
vrcholů, které odpovídají srdečním úderům.
        derivative = np.gradient(signal_processed)

        # Hledání nulových průchodů podle derivace
        zero_crossings = np.where((derivative[:-1] > 0) &
(derivative[1:] < 0))[0]

        # Filtrace nulových průchodů podle prahu
        peaks = zero_crossings[
            (signal_processed[zero_crossings] > threshold)
            & (zero_crossings < len(signal_processed))
        ]
        # Přidání výsledků do seznamu
        # Název souboru, vypočítaná tepová frekvence, referenční
tepová frekvence
        data.append(
```

```
            [
                name,
                calculate_heart_rate(peaks, sampling_frequency),
                calculate_heart_rate(true_peaks, sampling_frequency),
            ]
        )
    # Vytvoření DataFrame
    df = pd.DataFrame(data, columns=["name", "heart_rate",
"true_heart_rate"])
    # Výpočet přesnosti
    df["accuracy"] = (
        1 - abs(df["heart_rate"] - df["true_heart_rate"]) /
df["true_heart_rate"]
    ) * 100
    return df
```

Zavolání finální funkce a výpis výsledků

```
df: pd.DataFrame = get_data()
print(df)
print(f"Průměrná přesnost: {df['accuracy'].mean()} %")
```

```
Počet souborů: 18
     name  heart_rate  true_heart_rate   accuracy
0   16539   82.740077        84.097411  98.385998
1   18184   81.846837        92.202413  88.768650
2   16273   94.647887        95.401818  99.209731
3   19088   91.050476       106.032775  85.870125
4   17453   79.210142        79.366940  99.802438
5   16773   75.180318       106.757338  70.421686
6   16786   73.780384        74.013170  99.685481
7   16795   73.497237        81.295861  90.407108
8   19140   93.075991        97.008908  95.945819
9   16265  100.119751       101.144658  98.986691
10  19090   83.077755        88.180426  94.213375
11  16420   85.855143        88.636377  96.862197
12  17052   85.137820        85.904148  99.107927
13  19093   67.771486        73.764882  91.875000
14  16483   89.514254        89.892856  99.578830
15  19830  105.535049       109.362954  96.499816
16  18177  107.706989       114.703745  93.900151
17  16272   89.190569        76.501980  83.414039
Průměrná přesnost: 93.49639231274452 %
```

Slovní hodnocení

Původní hodnocení

V tomto úseku jsme porovnávali vytvořený algoritmus proti referenčním datům.

V rámci řešení zadaného úkolu jsme se potýkali s problémem načtení referenčních dat v původním prostředí Jupyter Notebook. Důvodem byla inkompatibilita datového typu int8, která způsobila přetečení. Problém byl vyřešen přesunutím do klasického Python skriptu.

Téměr okamžitě se nám podařilo načíst a zpracovat. Po úpravě parametrů pro zajištění kompatibility dat byl signál zpracován a byla provedena extrakce hodnot BPM. Výsledné hodnoty byly uloženy do struktury Pandas DataFrame a následně vyhodnoceny s výslednou úspěšností 93,5 %.

Když se úkol dodělal pomocí skriptu. Zkusili jsme vytvořit nový Jupyter Notebook a začalo to opět fungovat.

Tato část seminární práce byla pro mě zajímavá, protože jsme museli vyřešit přetečení datového typu, která byla nakonec úspěšně překonána.