Univerzita Jana Evangelisty Purkyně v Ústí nad Labem
Přírodovědecká fakulta

Řešení 1. a 2. seminární práce k předmětu Matematický software (KI/MSW)

Vypracoval: Martin Zeman

Ústí nad Labem 2024

Obsah

1 Průměrování matice – rozostření

Zadání

Máme provést rozostření námi vybraného obrázku pomocí průměrování a dále porovnat výsledek a časovou náročnost s metodou z knihovny pro zpracování obrazu.

Řešení

Rozostření implementujeme v kódu jako funkci, jejímiž vstupy jsou obrázek ve formě pole a poloměr kernelu. Funkce vrací rozostřený obrázek, opět jako pole.

```
import numpy as np
def average_blur(arr: np.ndarray, r=1) -> np.ndarray:
    y, x, z = arr.shape
    ...
```

Než se začneme zabývat průměrováním, je nutné rozhodnout, jakým způsobem budou v obrázku ošetřeny okraje. Nové hodnoty v obrázku počítáme jako průměr hodnot sousedních, na okrajích ale tito „sousedé“ chybí, je proto nutné je uměle doplnit. Toto vyřešíme rozšířením obrázku pomocí metody `numpy.pad()`.

```
arr_padded = np.pad(arr, ((r,r),(r,r),(0,0)), mode="reflect")
```

Obrázek rozšiřujeme v prvních dvou dimenzích (výška, šířka) o poloměr kernelu.

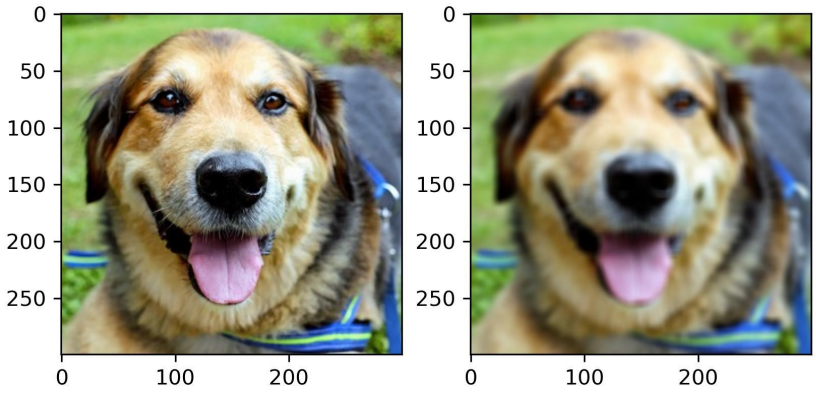
V tuto chvíli se můžeme věnovat samotnému průměrování. Abychom snížili časovou náročnost, průměrování v jednotlivých kanálech provádíme paralelně pomocí `numpy.average()`.

```
out = np.empty_like(arr)
for i in range(y):
        for j in range(x):
                avg = np.average(arr_padded[i:i+2*r+1, j:j+2*r+1],
                                 axis=(0, 1))
                out[i, j] = avg
```

Obrázek 1.1 demonstruje efekt našeho rozostření na fotce psa.

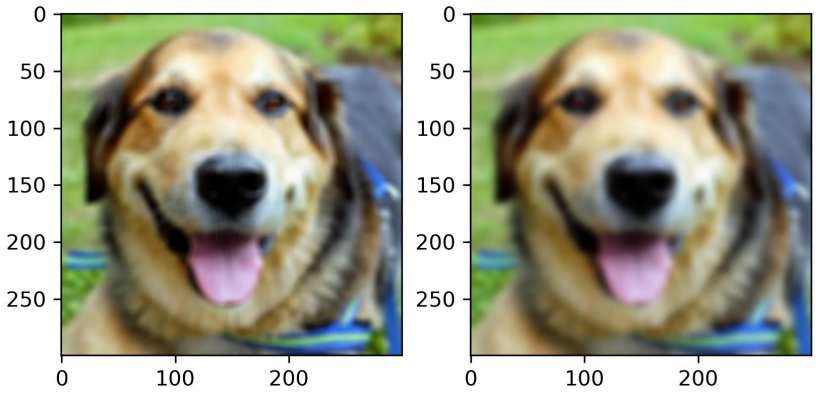
Vlastní implementaci budeme dle zadání porovnávat s metodou `GaussianBlur()` z knihovny `PIL`. Pro homogennost našeho kódu vytvoříme kolem metody wrapper.

```
from PIL import Image
from PIL.ImageFilter import GaussianBlur
def gaussian_blur(img: Image, r=1) -> Image:
    return img.filter(GaussianBlur(radius=r))
```



Obrázek 1.1: Srovnání původního a rozostřeného obrázku. Poloměr kernelu je zde $r = 3$.

Obrázek 1.2 demonstruje rozdíly mezi `average_blur()` a `gaussian_blur()`. Nejnápadnější je zde odlišnost charakterů. Je to dáno tím, že zatímco naše implementace přiřazuje každé ze sousedních hodnot stejnou váhu (je lineární), Gaussovo rozostření přiřazuje váhy na základě normální distribuce, díky čemuž vypadá o něco přirozeněji.



Obrázek 1.2: Srovnání vlastní implementace (vlevo) a Gaussova rozostření z knihovny `PIL` (vpravo).

Dalším rozdílem je zde rychlost – knihovna `PIL` je oproti našemu řešení mnohem více optimalizovaná. Tabulka 1.1 ukazuje, jak zásadní tento rozdíl je. Je zajímavé, že s větším poloměrem kernelu se `PIL` implementace navzdory očekávání zrychlila.

r	average_blur()	gaussian_blur()
1	2.9792	0.0049
3	3.0763	0.0046
5	3.2326	0.0047
7	3.4679	0.0047

Tabulka 1.1: Srovnání časové náročnosti metod v závislosti na velikost kernelu (v sekundách).

2 Klasifikace EMG signálů

Zadání

Máme spočítat tzv. integrované EMG (iEMG) pro signály v souboru `EMG.txt` a dále detekovat oblasti, kde došlo k nárůstu nebo poklesu aktivity. Získané hodnoty budou zaneseny do grafu.

Řešení

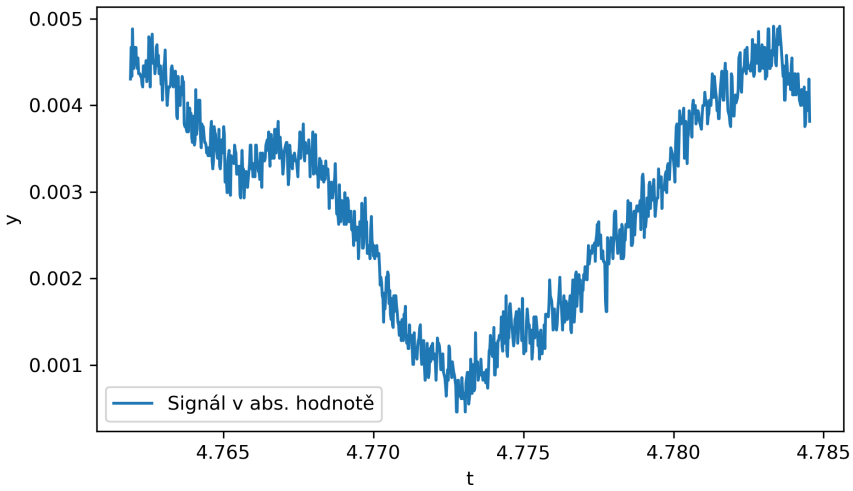
Data ze souboru načteme do proměnné pomocí `numpy.genfromtxt()`. Pro přehlednost si vyčleníme jednotlivé veličiny. V průběhu řešení budeme pracovat pouze s jedním ze signálů, na konci se krátce podíváme i na signál druhý.

```
import numpy as np
data = np.genfromtxt("EMG.txt")
t = data[:,0]
y = data[:,2]
```

Prvním krokem při výpočtu iEMG je převedení signálu na jeho absolutní hodnoty. My toto provedeme pomocí metody `numpy.abs()`.

```
y_abs = np.abs(y)
```

Obrázek 2.1 zobrazuje signál po aplikaci funkce absolutní hodnoty. Vidíme, že obsahuje podstatné množství šumu, což by se později projevilo i na čitelnosti získaného iEMG. Častým opatřením proto bývá signál vyhladit (tzv. *smoothing*). My zde použijeme Savitzky-Golayův filtr, který je součástí knihovny `scipy`.



Obrázek 2.1: Vizualizace signálu v absolutní hodnotě.

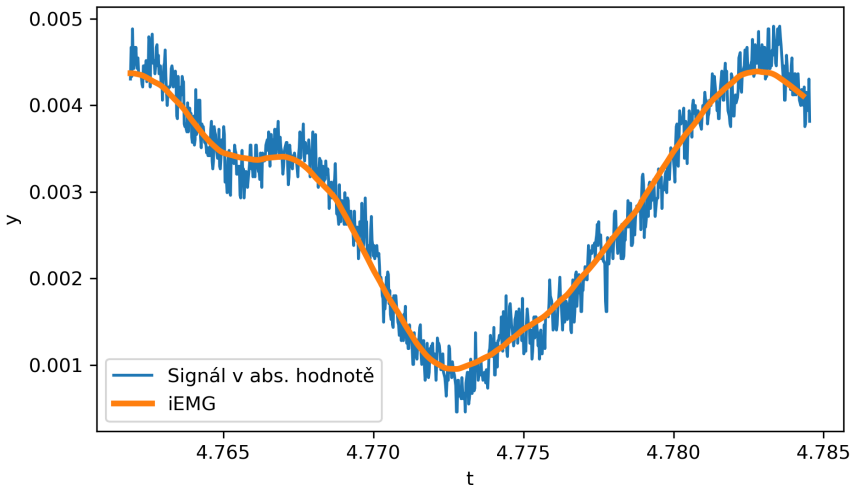
```
from scipy.signal import savgol_filter
smooth_amount = 115
y_smoothed = savgol_filter(y_abs, smooth_amount, 1, mode="nearest")
```

Dostáváme se k samotné integraci signálu. Kvůli podstatě určitého integrálu je nutné signál rozdělit do „okének“, v rámci kterých se bude integrál počítat. My zde zvolíme metodu tzv. plovoucího okna, kterou si pro přehlednost kódu zavedeme jako funkci.

```
def slide(arr: np.ndarray, size: int) -> np.ndarray:
    return np.array([arr[i:i+size]for i in range(len(arr)- size + 1)])
```

Po rozdělení signálu aplikujeme integrační funkci, kterých existuje několik. Pro naše účely postačí `numpy.trapz()`. Výsledek můžeme vidět na obrázku 2.2.

```
window_size = 11
y_int = np.trapz(slide(y_smoothed, window_size), dx=0.1)
```

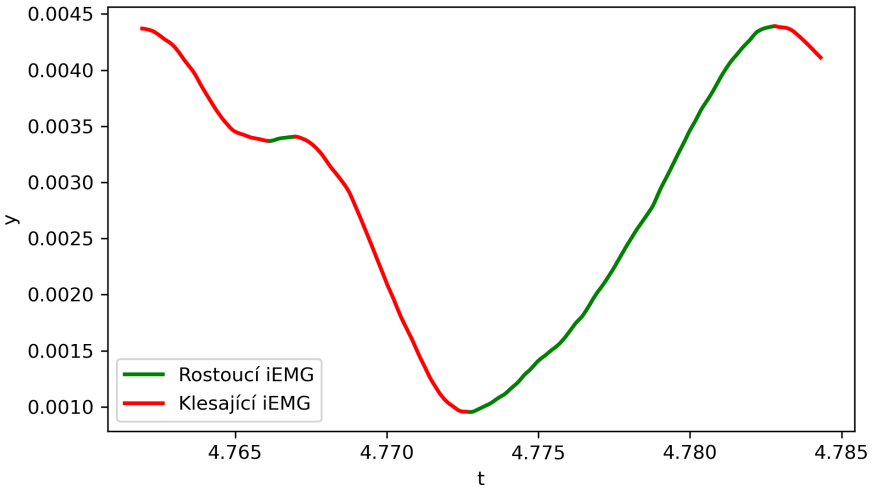


Obrázek 2.2: Signál a jeho iEMG.

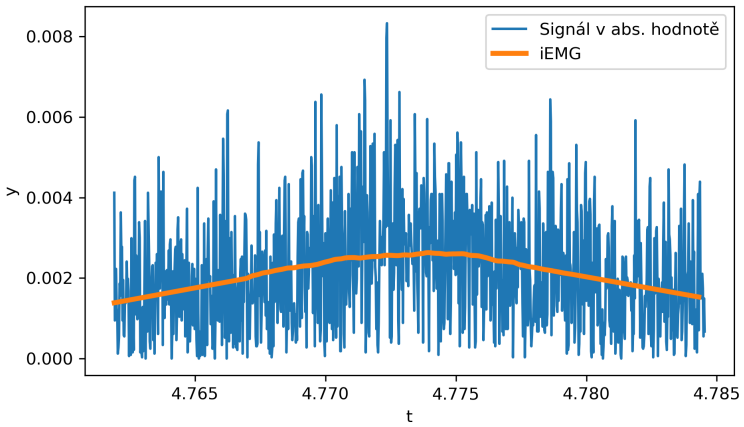
V souladu se zadáním derivujeme získané iEMG pomocí `numpy.gradient()`. Podle znaménka lze rozhodnout, zda zpracovaný signál v daném bodě roste, či klesá, což lze vyjádřit graficky, jak ukazuje obrázek 2.3.

```
dt = t[1] - t[0]
dy = np.gradient(y_int, dt)
```

Obrázek 2.4 zobrazuje, jak bylo slíbeno na začátku, zpracovaný druhý signál ze souboru. Za zmínku stojí, že kvůli vyšší míře šumu zde bylo nutné použít také silnější vyhlazování.



Obrázek 2.3: Získané iEMG obarvené podle znaménka derivace.



Obrázek 2.4: Absolutní hodnota a iEMG druhého signálu.

3 Šíření nemoci – SIR model

Zadání

Máme vyřešit soustavu diferenciálních rovnic modelující vývoj nemoci v uzavřené populaci pomocí Eulerovy metody a ukázat vliv počátečních podmínek a koeficientů na průběh. Výsledky budou zaneseny do grafu.

Řešení

Soustavy diferenciálních rovnic nejsou kvůli podstatě limit počítačově zpracovatelné – převádíme je proto na soustavy rekurencí, které již zpracovat lze. Pokud toto provedeme se zadanou soustavou, obdržíme

$$\begin{aligned} S_{i+1} &= S_i - \beta S_i I_i \\ I_{i+1} &= I_i + \beta S_i I_i - \gamma I_i \\ R_{i+1} &= R_i + \gamma I_i. \end{aligned}$$

V soustavě úmyslně opomíjíme jakési Δt, které by nám určovalo velikost kroku v čase. Je to z toho důvodu, že tento krok lze vždy zohlednit při volbě koeficientů α a β, není proto třeba ho zvláště vyčleňovat.

V programové části je nutné nejprve zavést počáteční podmínky. Těmi jsou velikost populace, počet nakažených a rychlost nákazy a léčby. Koeficienty kvůli lepší čitelnosti píšeme ve formě zlomků.

```
N = 50
I_0 = 10
beta = 1/1000
gamma = 1/100
```

Dále je třeba iniciovat jednotlivé skupiny jedinců a jejich počáteční stavy. Výjimečně zde nepoužijeme knihovnu `numpy` – jak bude za chvíli patrné, byla by nám při řešení spíše na obtíž.

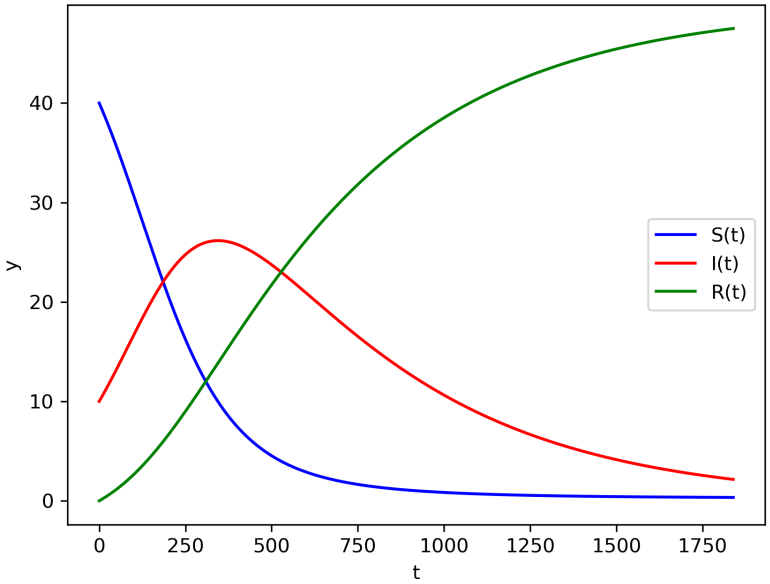
```
S = [ N-I_0 ]
I = [ I_0 ]
R = [ 0 ]
```

Nejdůležitějším krokem je vytvoření generátoru na základě naší soustavy. Program běží do té doby, dokud není 95 % populace promořeno. Rezerva je zde potřeba, jelikož model (bohužel) nepracuje pouze v diskrétních hodnotách – počet vyléčených proto nemusí k plnému počtu v rozumném čase vůbec dokonvergovat!

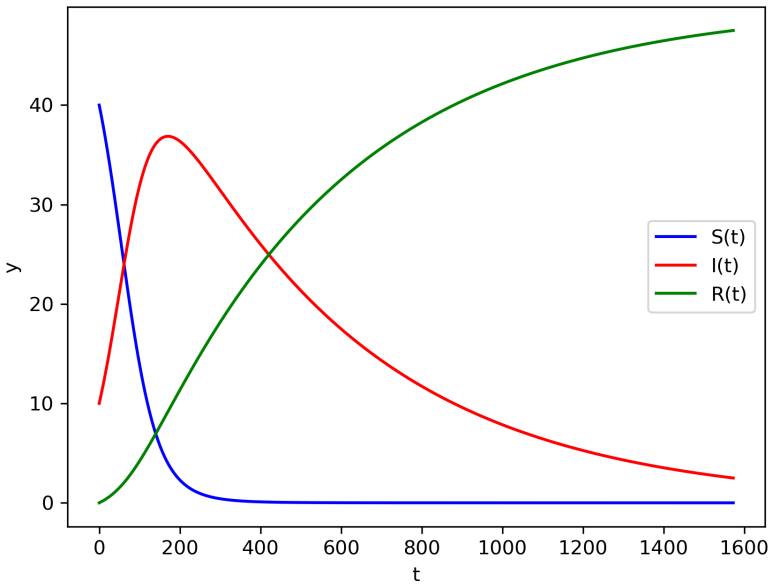
```
t = 0
while R[t] < N*0.95:
    S.append(S[t] - beta*S[t]*I[t])
    I.append(I[t] + beta*S[t]*I[t] - gamma*I[t])
    R.append(R[t] + gamma*I[t])
    t += 1
```

Zde je také konečně vidět, proč jsme nepoužili knihovnu `numpy`. Na seznamech nevoláme metody, které by mohly těžit z paralelizace, a dopředu neznáme ani jejich velikost, abychom si je mohli předem alokovat.

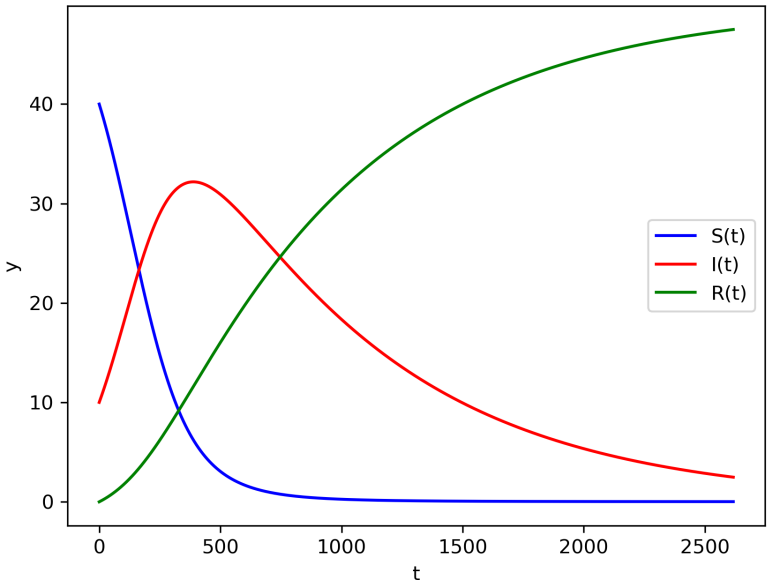
Následují grafické výstupy modelu pro různé počáteční podmínky. Na obrázku 3.1 vidíme průběh nákazy s relativně vyváženými parametry. Pokud zvýšíme nakažlivost, zaznamenáme prudší extrém v počtu nakažených, jak je vidět na obrázku 3.2. Zároveň se nám ale zkrátí doba, za jakou je populace promořena. Nakonec, pokud zmenšíme počet prvotních nakažených (obrázek 3.3), zaznamenáme časový posun v extrému nákazy – trvá déle, než se dostane do svých hraničních hodnot.



Obrázek 3.1: Průběh nákazy s parametry $I_0 = 10$, $\beta = \frac{1}{5000}$, $\gamma = \frac{1}{500}$.



Obrázek 3.2: Průběh nákazy s parametry $I_0 = 10$, $\beta = \frac{1}{2000}$, $\gamma = \frac{1}{500}$.



Obrázek 3.3: Průběh nákazy s parametry $I_0 = 10$, $\beta = \frac{1}{5000}$, $\gamma = \frac{1}{800}$.