Matematický software

Zápočtový dokument

Jméno: Radek Černý (f21081)

Kontaktní email: radek.cerny847@gmail.com

Datum odevzdání: 30.9.2023

Odkaz na repozitář: Github

Formální požadavky

Cíl předmětu:
Cílem předmětu je ovládnout vybrané moduly a jejich metody pro jazyk Python, které vám mohou být užitečné jak v dalších semestrech vašeho studia, závěrečné práci (semestrální, bakalářské) nebo technické a výzkumné praxi.

Získání zápočtu:
Pro získání zápočtu je nutné částečně ovládnout alespoň polovinu z probraných témat. To prokážete vyřešením vybraných úkolů. V tomto dokumentu naleznete celkem 10 zadání, která odpovídají probíraným tématům. Vyberte si 5 zadání, vypracujte je a odevzdejte. Pokud bude všech 5 prací korektně vypracováno, pak získáváte zápočet. Pokud si nejste jisti korektností vypracování konkrétního zadání, pak je doporučeno vypracovat více zadání a budou se započítávat také, pokud budou korektně vypracované.

Korektnost vypracovaného zadání:
Konkrétní zadání je považováno za korektně zpracované, pokud splňuje tato kritéria:

1. Použili jste numerický modul pro vypracování zadání místo obyčejného pythonu
2. Kód neobsahuje syntaktické chyby a je interpretovatelný (spustitelný)
3. Kód je čistý (vygooglete termín clean code) s tím, že je akceptovatelné mít ho rozdělen do Jupyter notebook buněk (s tímhle clean code nepočítá)

Forma odevzdání:
Výsledný produkt odevzdáte ve dvou podobách:

1. Zápočtový dokument
2. Repozitář s kódem

Zápočtový dokument (vyplněný tento dokument, který čtete) bude v PDF formátu. V řešení úloh uveďte důležité fragmenty kódu a grafy/obrázky/textový výpis pro ověření funkčnosti. Stačí tedy uvést jen ty fragmenty kódu, které přispívají k jádru řešení zadání. Kód nahrajte na veřejně přístupný repozitář (github, gitlab) a uveďte v práci na něj odkaz v titulní straně dokumentu. Strukturujte repozitář tak, aby bylo pro nás hodnotitele intuitivní se vyznat v souborech (doporučuji každou úlohu dát zvlášť do adresáře).

Podezření na plagiátorství:
Při podezření na plagiátorství (významná podoba myšlenek a kódu, která je za hranicí pravděpodobnosti shody dvou lidí) budete vyzváni k fyzickému dostavení se na zápočet do prostor univerzity, kde dojde k vysvětlení podezřelých partií, nebo vykonání zápočtového testu na místě z matematického softwaru v jazyce Python.

Kontakt:
Při nejasnostech ohledně zadání nebo formě odevzdání se obraťte na vyučujícího.

1. Knihovny a moduly pro matematické výpočty

Zadání:
V tomto kurzu jste se učili s některými vybranými knihovnami. Některé sloužily pro rychlé vektorové operace, jako numpy, některé mají naprogramovány symbolické manipulace, které lze převést na numerické reprezentace (sympy), některé mají v sobě funkce pro numerickou integraci (scipy). Některé slouží i pro rychlé základní operace s čísly (numba).

Vaším úkolem je změřit potřebný čas pro vyřešení nějakého problému (např.: provést skalární součin, vypočítat určitý integrál) pomocí standardního pythonu a pomocí specializované knihovny. Toto měření proveďte alespoň pro 5 různých úloh (ne pouze jiná čísla, ale úplně jiné téma) a minimálně porovnejte rychlost jednoho modulu se standardním pythonem. Ideálně proveďte porovnání ještě s dalším modulem a snažte se, ať je kód ve standardním pythonu napsán efektivně.

Řešení:
V mém případě jsem zjišťoval:

- faktoriál pomocí modulu math a čistého pythonu
- násobení matic/e v numpy a čistém pythonu
- skalární součin v numpy a čistém pythonu
- derivaci pres sympy a čistý python
- integraci pres scipy, numba a čistý python

```
modul math:
time: 0.0019991397857666016, faktorial: 154940020174866655446905670067469383963570984285647262525
vanilla python:
time:0.018358707427978516, faktorial: 154940020174866655446905670067469383963570984285647262522

numpy:
[[ 321925  127475]
 [ 292850 1963800]
 [ 338900   88650]], time:0.0
vanilla python:
[[ 321925  127475]
 [ 292850 1963800]
 [ 338900   88650]], time:0.0

numpy: 59362, time: 0.0
vanilla python: 59362, time: 0.0009984970092773438

sympy derivace: 5733067680, time:0.0009999275207519531
vanilla python: 5734740107.955933, time: 0.0

scipy: 392.25, time: 0.0
vanilla python:392.2500005326722, time: 0.05176973342895508
numba: 392.2500005326722
time: 0.714684009552002
```

Časy pro porovnání jsou bohužel moc malé na to, abych z nich vyvodil nějaké závěry, nehledě na to, že pro testování, který způsob je rychlejší, by bylo dobré projít více spuštění s různými startovními podmínkami (různé funkce)
Tak jako tak obecně pro práci s čísly a matematické operace, z toho bude built-in modul math a nebo numpy vycházet většinou, né-li vždy lépe, než nativní python.
Výsledek za numbu mě nepřekvapil, leč jsem funkci, kterou volám, obalil v numba dekorátoru, nevyužívám v ní žádných cyklů ani datových typů, které zlepšují/zrychlují chod kódu v numbě.
Jinak samozřejmě pro některé situace je použití JIT compileru (jako numba) klíčové.

2. Vizualizace dat

Zadání:
V jednom ze cvičení jste probírali práci s moduly pro vizualizaci dat. Mezi nejznámější moduly patří matplotlib (a jeho nadstavby jako seaborn), pillow, opencv, aj. Vyberte si nějakou zajímavou datovou sadu na webovém portále Kaggle a proveďte datovou analýzu datové sady. Využijte k tomu různé typy grafů a interpretujte je (minimálně alespoň 5 zajímavých grafů). Příklad interpretace: z datové sady pro počasí vyplynulo z liniového grafu, že v létě je vyšší rozptyl mezi minimální a maximální hodnotou teploty. Z jiného grafu vyplývá, že v létě je vyšší průměrná vlhkost vzduchu. Důvodem vyššího rozptylu může být absorpce záření vzduchem, který má v létě vyšší tepelnou kapacitu.

Řešení:
Z Kaggelu jsem si stáhl datovou sadu věnující se počtu nakažených covidem v jednotlivých amerických státech: [COVID-19 State Data]
Nahrál jsem jí skrze pandas:

```
1 import seaborn as sns
2 import pandas as pd
3 import matplotlib.pyplot as plt
```
✓ 0.0s

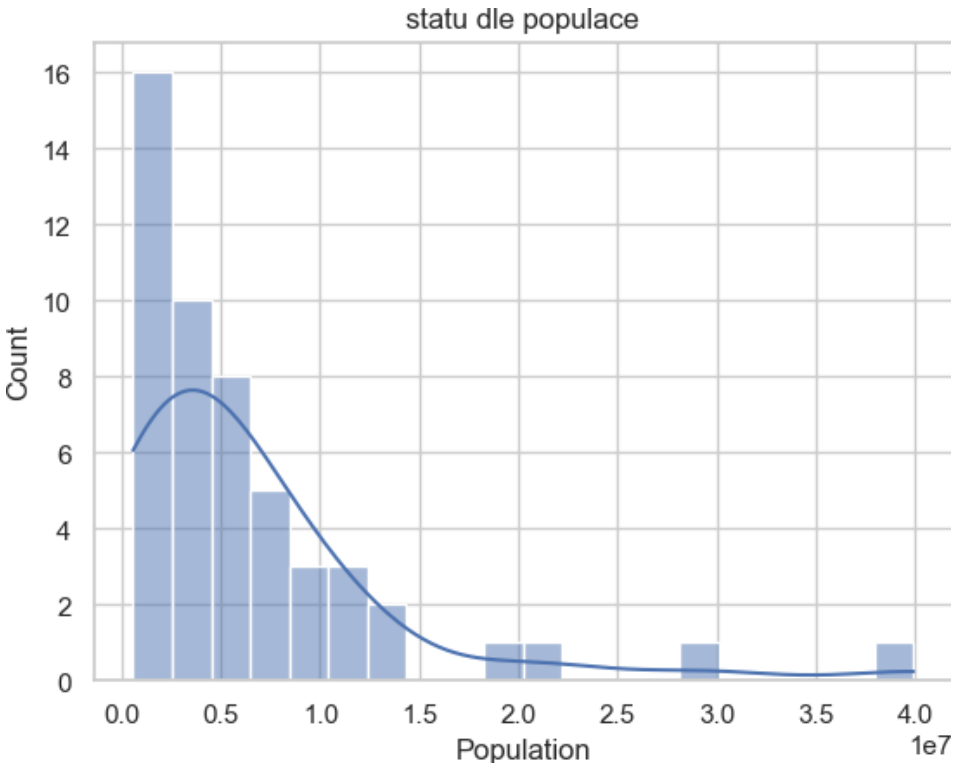
```
1 %matplotlib inline
```
✓ 0.0s

```
1 data = pd.read_csv('COVID19_state.csv')
```
✓ 0.0s

```
1 data.head()
```
✓ 0.0s

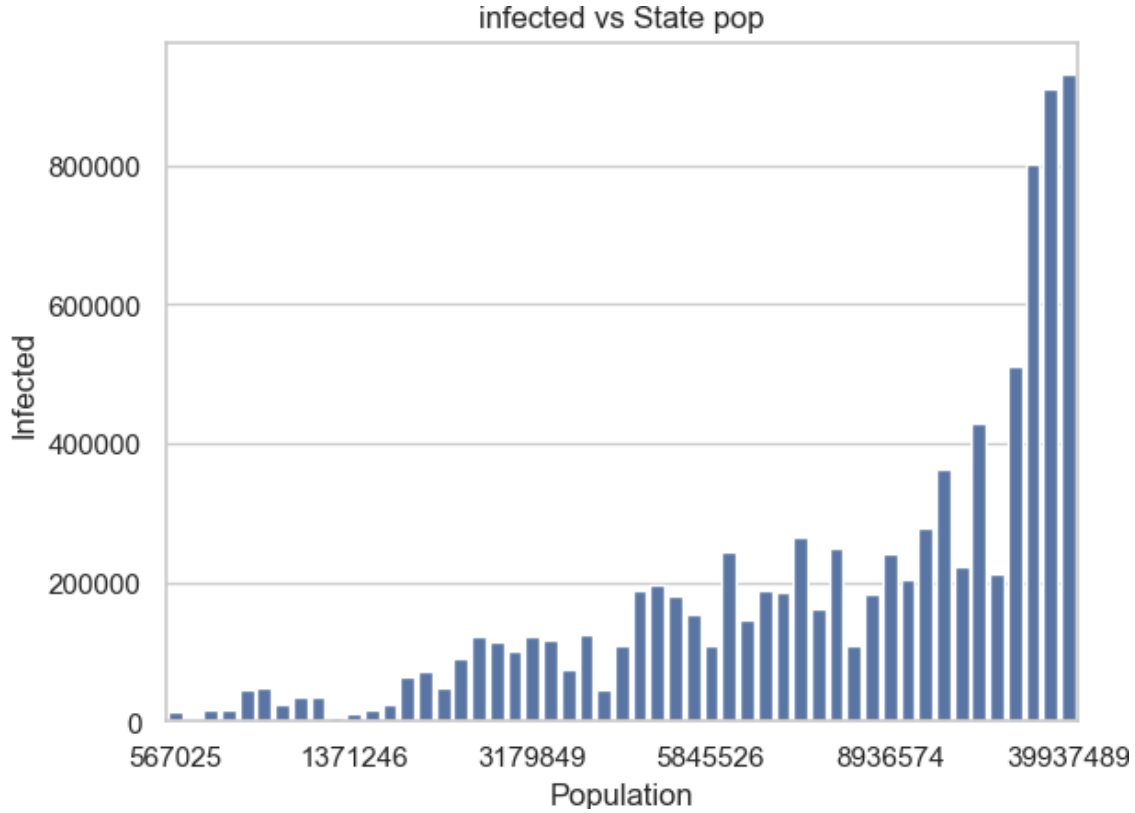
	State	Tested	Infected	Deaths	Population	Pop Density	Gini	ICU Beds	Income	GDP	...	Hospitals	Health Spend
0	Alaska	620170	17057	84	734002	1.2863	0.4081	119	59687	73205	...	21	11
1	Alabama	1356420	194892	2973	4908621	96.9221	0.4847	1533	42334	45219	...	101	7
2	Arkansas	1363429	113641	1985	3038999	58.4030	0.4719	732	42566	42454	...	88	7
3	Arizona	1792602	248139	5982	7378494	64.9550	0.4713	1559	43650	48055	...	83	6
4	California	18912501	930628	17672	39937489	256.3727	0.4899	7338	62586	74205	...	359	7

Poté jsem již začal s tvorbou grafů, při které jsem použil knihovnu seaborn.
Prvním grafem byl graf států dle populace respektive, kolik je států s jakým množstvím obyvatel:



Z grafu lze vyčíst, že je ve USA většina států, které nemají ani 10 milionů obyvatel

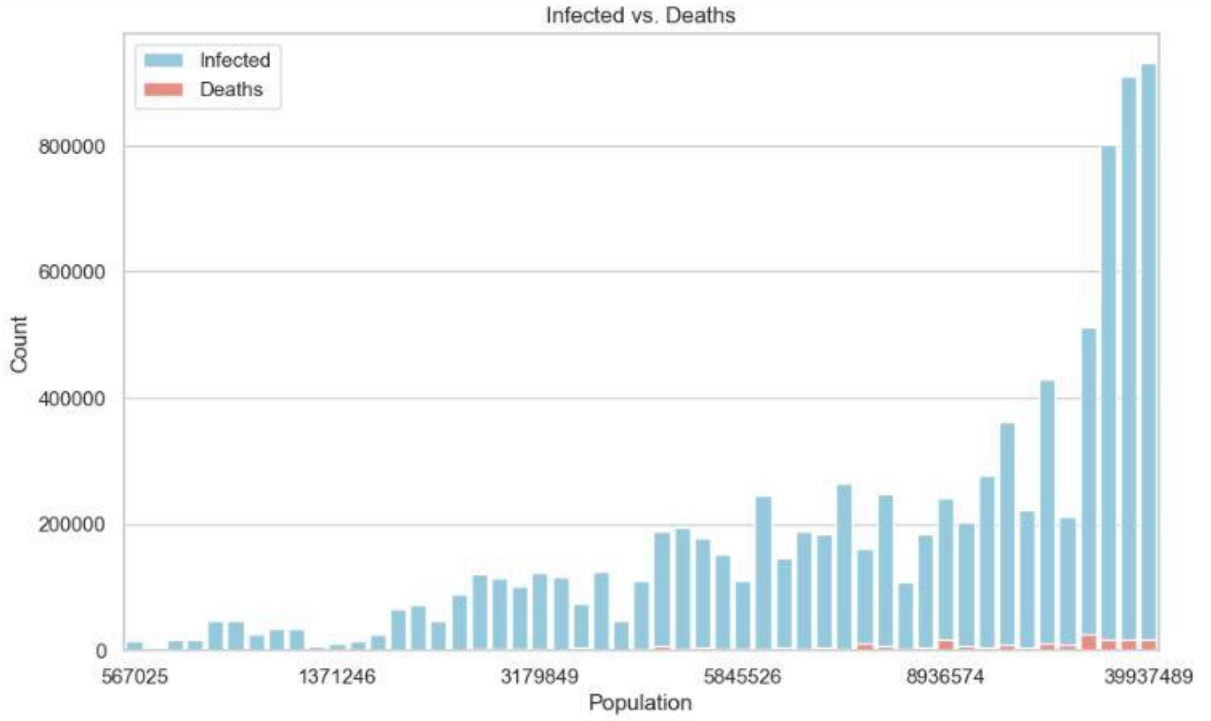
Dalším grafem je graf infikovaných COVIDEM-19 na počet obyvatel v průřezu státy. Jde si všimnout, že čím více obyvatel ve státu, tím více nakažených:



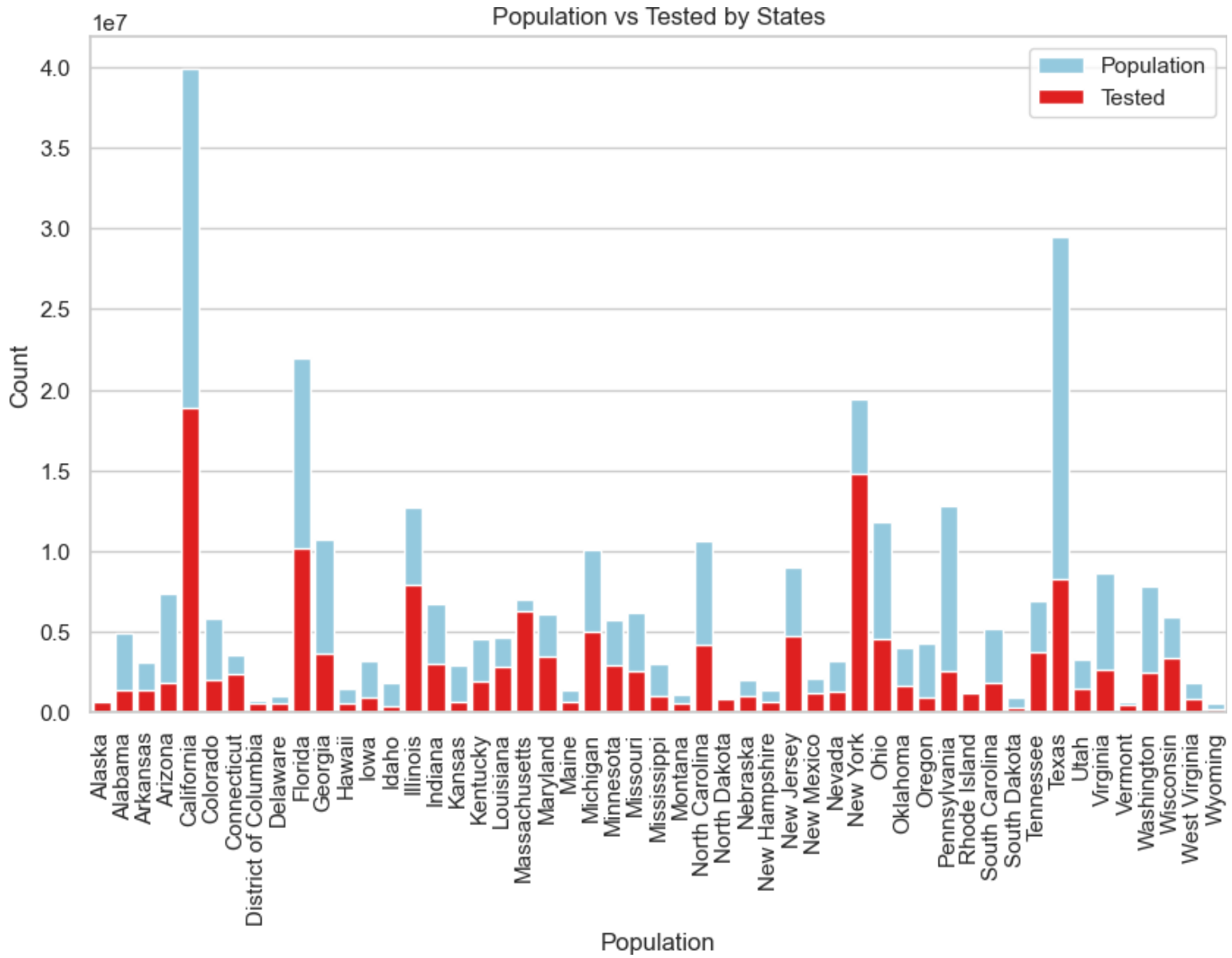
Jako další mě zajímalo, kolik mrtvých je na počet nakažených. což se v grafu objevilo, jako mizivá část, čili smrtnost na COVID-19 je podle toho grafu nízká:

```python
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))

graf =sns.barplot(x='Population', y='Infected', data=data, color='skyblue', label='Infected')
sns.barplot(x='Population', y='Deaths', data=data, color='salmon', label='Deaths')

for ind, label in enumerate(graf.get_xticklabels()):
    if ind % 10 == 0:  # every 10th label is kept
        label.set_visible(True)
    else:
        label.set_visible(False)

plt.title('Infected vs. Deaths')
plt.xlabel('Population')
plt.ylabel('Count')
plt.legend()
#plt.savefig('InfvsDeaths.png', bbox_inches='tight', pad_inches=0)
plt.show()
```

✓ 2.5s



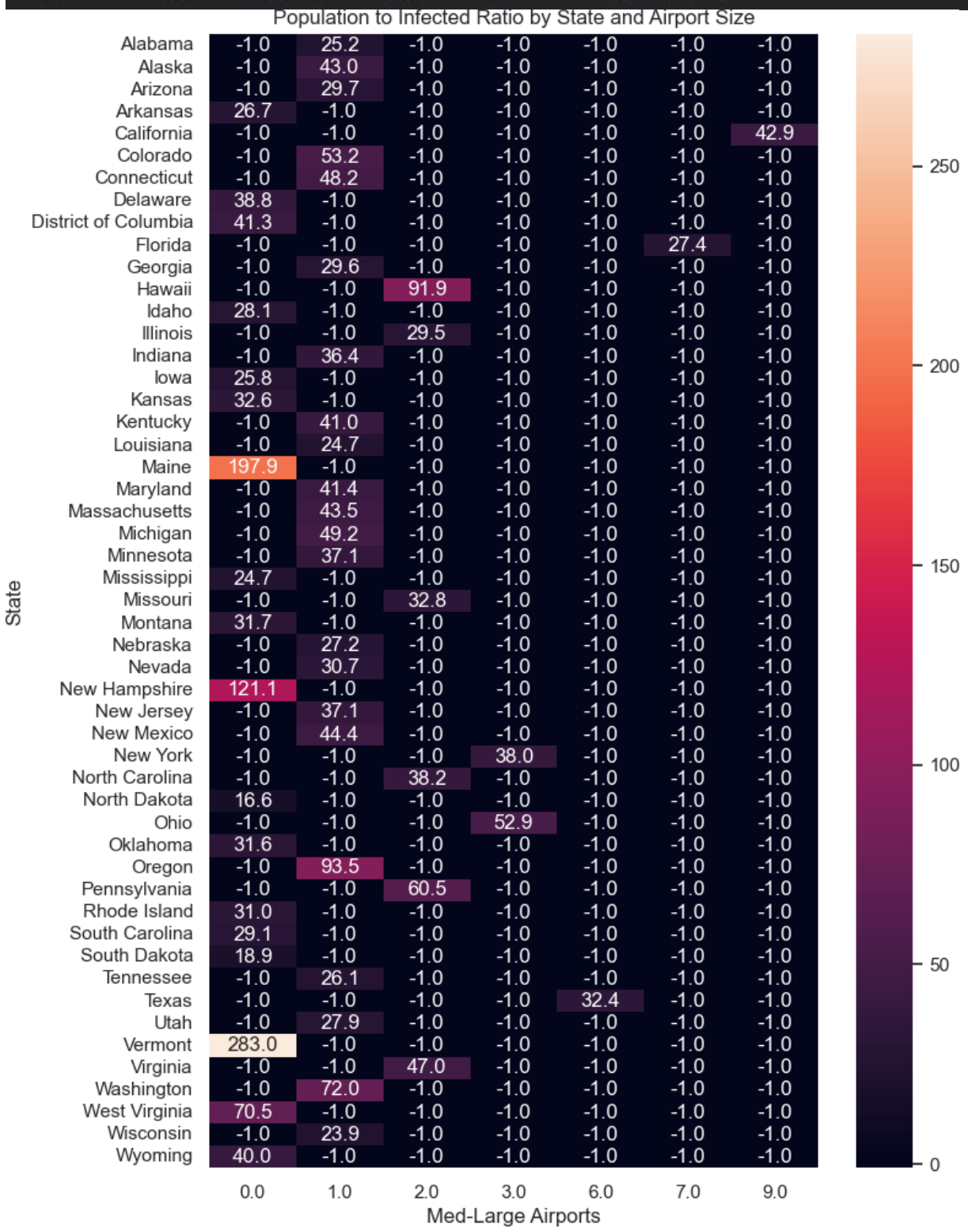
Další je graf populace v jednotlivých státech a počet testovaných lidí na COVID-19:



Lze si všimnout, že počet otestovaných lidí je zhruba v polovině celkového počtu lidí v daných státech. Co přesně je myšleno pojmem “otestovaný” jsem z grafu ani z datové sady nevyčetl. ale předpokládám že je to osoba, která byla někdy otestovaná na covid-19. V jakém časovém horizontu není udáváno.

Posledním grafem je graf, který mi měl pomoc odhalit, zda nějak souvisí počet nakažených lidí v jednotlivých státech s tím, kolik má daný stát středně velkých a velkých letišť:
Vyšel mi koeficient, který říká, kolik je zdravých lidí na počet nakažených. Čím menší je tedy toto číslo, tím více nakažených lidí v daném místě bylo. hodnoty s -1 jsou to pro neexistující letiště v dané oblasti.:

```python
data['Pop_Inf_Coef'] = data['Population'] / data['Infected']
plt.figure(figsize=(8, 12))
#tvorba matice z hlediska poctu nakazenych na pocet letist, cim nizsi cislo, tim vice nakažených v
Inf_to_Airport = data.pivot_table(index='State',columns='Med-Large Airports',values='Pop_Inf_Coef')
#hodnota -1 je pro stav, kdyz se v danem statu nenachazi zadne letiste
Inf_to_Airport.fillna(-1,inplace=True)
sns.heatmap(Inf_to_Airport, annot=True, fmt='.1f')
plt.title('Population to Infected Ratio by State and Airport Size')
#plt.savefig('Inf_to_Airport.png', bbox_inches='tight', pad_inches=0)
```



State	0.0	1.0	2.0	3.0	6.0	7.0	9.0
Alabama	-1.0	25.2	-1.0	-1.0	-1.0	-1.0	-1.0
Alaska	-1.0	43.0	-1.0	-1.0	-1.0	-1.0	-1.0
Arizona	-1.0	29.7	-1.0	-1.0	-1.0	-1.0	-1.0
Arkansas	26.7	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
California	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	42.9
Colorado	-1.0	53.2	-1.0	-1.0	-1.0	-1.0	-1.0
Connecticut	-1.0	48.2	-1.0	-1.0	-1.0	-1.0	-1.0
Delaware	38.8	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
District of Columbia	41.3	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Florida	-1.0	-1.0	-1.0	-1.0	-1.0	27.4	-1.0
Georgia	-1.0	29.6	-1.0	-1.0	-1.0	-1.0	-1.0
Hawaii	-1.0	-1.0	91.9	-1.0	-1.0	-1.0	-1.0
Idaho	28.1	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Illinois	-1.0	-1.0	29.5	-1.0	-1.0	-1.0	-1.0
Indiana	-1.0	36.4	-1.0	-1.0	-1.0	-1.0	-1.0
Iowa	25.8	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Kansas	32.6	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Kentucky	-1.0	41.0	-1.0	-1.0	-1.0	-1.0	-1.0
Louisiana	-1.0	24.7	-1.0	-1.0	-1.0	-1.0	-1.0
Maine	197.9	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Maryland	-1.0	41.4	-1.0	-1.0	-1.0	-1.0	-1.0
Massachusetts	-1.0	43.5	-1.0	-1.0	-1.0	-1.0	-1.0
Michigan	-1.0	49.2	-1.0	-1.0	-1.0	-1.0	-1.0
Minnesota	-1.0	37.1	-1.0	-1.0	-1.0	-1.0	-1.0
Mississippi	24.7	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Missouri	-1.0	-1.0	32.8	-1.0	-1.0	-1.0	-1.0
Montana	31.7	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Nebraska	-1.0	27.2	-1.0	-1.0	-1.0	-1.0	-1.0
Nevada	-1.0	30.7	-1.0	-1.0	-1.0	-1.0	-1.0
New Hampshire	121.1	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
New Jersey	-1.0	37.1	-1.0	-1.0	-1.0	-1.0	-1.0
New Mexico	-1.0	44.4	-1.0	-1.0	-1.0	-1.0	-1.0
New York	-1.0	-1.0	-1.0	38.0	-1.0	-1.0	-1.0
North Carolina	-1.0	-1.0	38.2	-1.0	-1.0	-1.0	-1.0
North Dakota	16.6	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Ohio	-1.0	-1.0	-1.0	52.9	-1.0	-1.0	-1.0
Oklahoma	31.6	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Oregon	-1.0	93.5	-1.0	-1.0	-1.0	-1.0	-1.0
Pennsylvania	-1.0	-1.0	60.5	-1.0	-1.0	-1.0	-1.0
Rhode Island	31.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
South Carolina	29.1	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
South Dakota	18.9	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Tennessee	-1.0	26.1	-1.0	-1.0	-1.0	-1.0	-1.0
Texas	-1.0	-1.0	-1.0	-1.0	32.4	-1.0	-1.0
Utah	-1.0	27.9	-1.0	-1.0	-1.0	-1.0	-1.0
Vermont	283.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Virginia	-1.0	-1.0	47.0	-1.0	-1.0	-1.0	-1.0
Washington	-1.0	72.0	-1.0	-1.0	-1.0	-1.0	-1.0
West Virginia	70.5	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
Wisconsin	-1.0	23.9	-1.0	-1.0	-1.0	-1.0	-1.0
Wyoming	40.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0

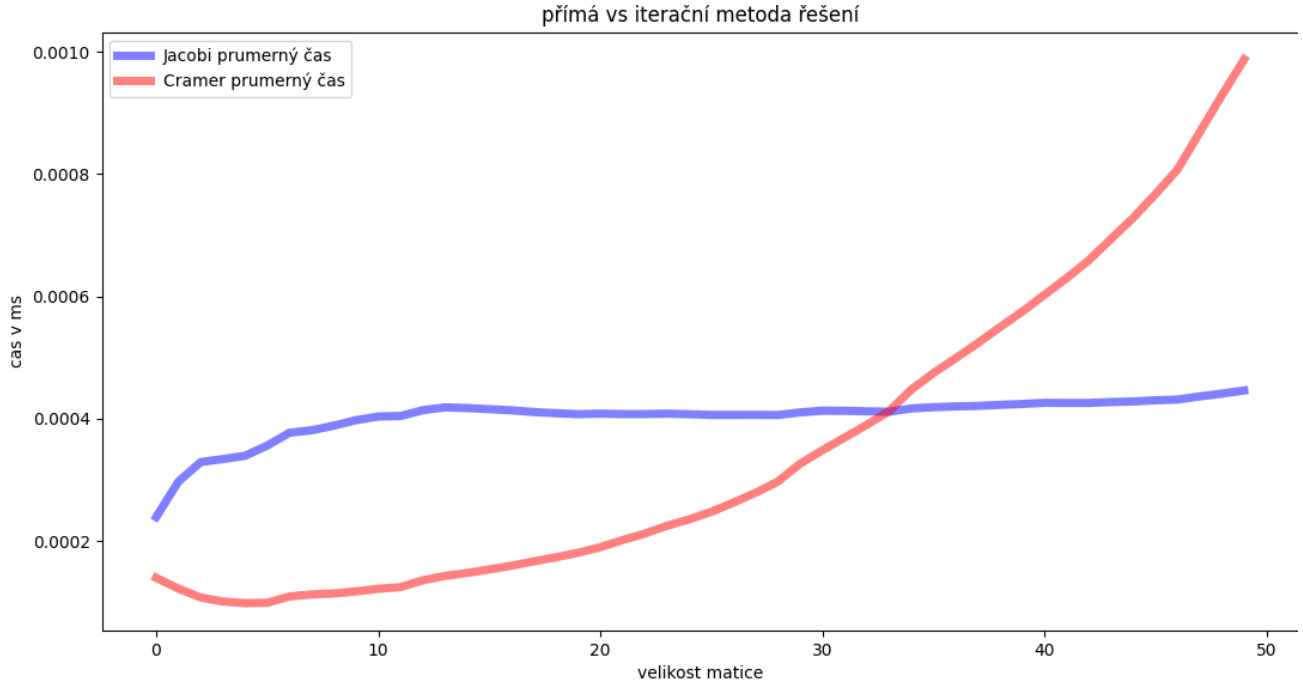
Bohužel bych řekl, že není z tohoto grafu patrné, že by na tom státy bez letiště, byly nějak

výrazně lépe z hlediska počtu nakažených. Jediné výjimky snad tvoří Vermont a Maine, kde je na jednoho nakaženého 283, respektive 198 zdravých lidí a zároveň v těchto státech není středně velké ani velké letiště.

3. Úvod do lineární algebry

Zadání:
Důležitou částí studia na přírodovědecké fakultě je podobor matematiky zvaný lineární algebra. Poznatky tohoto oboru jsou základem pro oblasti jako zpracování obrazu, strojové učení nebo návrh mechanických soustav s definovanou stabilitou. Základní úlohou v lineární algebře je nalezení neznámých v soustavě lineárních rovnic. Na hodinách jste byli obeznámeni s přímou a iterační metodou pro řešení soustav lineárních rovnic. Vaším úkolem je vytvořit graf, kde na ose x bude velikost čtvercové matice a na ose y průměrný čas potřebný k nalezení uspokojivého řešení. Cílem je nalézt takovou velikost matice, od které je výhodnější využít iterační metodu.

Řešení:
Jakobiho a Cramerovu metodu jsem převzal ze sešitu z hodin, cyklus a měření jejich výkonnosti jsem naprogramoval a vyšly mi tyto výsledky:



Probíhá tedy opakovaný výpočet hodnot pro Jakobiho a Cramera. Výpočet je opakován pro každou velikost matice 100 krát a zprůměrován.

Z grafu tedy vyplývá, že pro matici o velikosti zhruba N > 34 je výhodnější použít iterační metodu.

8. Derivace funkce jedné proměnné

Zadání:
Numerická derivace je velice krátké téma. V hodinách jste se dozvěděli o nejvyužívanějších typech numerické derivace (dopředná, zpětná, centrální). Jedno z neřešených témat na hodinách byl problém volby kroku. V praxi je vhodné mít krok dynamicky nastavitelný. Algoritmům tohoto typu se říká derivace s adaptabilním krokem. Cílem tohoto zadání je napsat program, který provede numerickou derivaci s adaptabilním krokem pro vámi vybranou funkci. Proveďte srovnání se statickým krokem a analytickým řešením.

Řešení:
První jsem si vytvořil potřebné funkce pro dopřednou, zpětnou a centrální derivaci:

```
4  def dopredna_der(f, x0, delta_x):
5      return (f(x0+delta_x) - f(x0))/delta_x
6
7  def zpetna_der(f, x0, delta_x):
8      return (f(x0) - f(x0-delta_x))/delta_x
9
10 def central_der(f, x0, delta_x):
11     return (f(x0+delta_x) - f(x0-delta_x))/2*delta_x
```

poté funkci, kterou budeme derivovat: $f(x) = x^3 + 5$

```
def f(x):
    return (x**3+5)
```

delta_x = 0.001, pokud tedy nevložíme v případě dynamické změny kroku jinou hodnotu
x0 = 5
pak už jen funkce volám a výsledek je:

```
Derivace funkce x**2:
pro x0 = 5, delta_x = 0.001:
dopredna: 75.01500100002545, zpetna: 74.98500100001593, centralni: 75.00000100002069
pro x0 = 5, delta_x = 0.5:
dynamicka dopredna: 82.7500000000, dynamicka zpetna: 67.7500000000, dynamicka centralni: 75.2500000000
```

Čím menší delta_x(krok), tím přesnější výsledek a to blíže si výsledky jsou v rámci jednotlivých druhů použitých numerických derivací.
výše jsou výsledy pro adaptivní krok = 0.5
níže je adaptivní krok = 1e-09

```
Derivace funkce x**2:
pro x0 = 5, delta_x = 0.001:
dopredna: 75.0150010000, zpetna: 74.9850010000, centralni: 75.0000010000
pro x0 = 5, delta_x = 1e-09:
dynamicka dopredna: 75.0000026528, dynamicka zpetna: 75.0000026528, dynamicka centralni: 75.0000026528
```

9. Integrace funkce jedné proměnné

Zadání:
V oblasti přírodních a sociálních věd je velice důležitým pojmem integrál, který představuje funkci součtů malých změn (počet nakažených covidem za čas, hustota monomerů daného typu při posouvání se v řetízku polymeru, aj.). Integraci lze provádět pro velmi jednoduché funkce prostou Riemannovým součtem, avšak pro složitější funkce je nutné využít pokročilé metody. Vaším úkolem je vybrat si 3 různorodé funkce (polynom, harmonická funkce, logaritmus/exponenciála) a vypočíst určitý integrál na dané funkci od nějakého počátku do nějakého konečného bodu. Porovnejte, jak si každá z metod poradila s vámi vybranou funkcí na základě přesnosti vůči analytickému řešení.

Řešení:
Vybrané funkce:
polynomiální: $3x^2 + 5x + 12$
harmonická: $3\sin^2(x)$
exponenciální: $3^{(6x)} + 2^x$

Použité metody:
obdélníková metoda, lichoběžníková metoda, rombergova metoda

```python
#numerické metody pro vypočet integralu:
def obdelnikova_metoda(fce, start, end):
    return integrate.quad(fce, start, end)

def lichobeznikova_metoda(fce, start, end, n = 100):
    return integrate.trapezoid(fce(np.linspace(start, end, n)), np.linspace(start, end, n))

def rombergova_metoda(fce, start, end):
    return integrate.romberg(fce, start, end)

#polynom fce
print(f"integrace obdelnikovou metodou: {obdelnikova_metoda(polyf, a, b)[0]}")
print(f"integrace lichobeznikovou metodou: {lichobeznikova_metoda(polyf, a, b)}")
print(f"integrace rombergovou metodou: {obdelnikova_metoda(polyf, a, b)[0]}\n")
#harmo fce
print(f"integrace obdelnikovou metodou: {obdelnikova_metoda(harmf, a, b)[0]}")
print(f"integrace lichobeznikovou metodou: {lichobeznikova_metoda(harmf, a, b)}")
print(f"integrace rombergovou metodou: {obdelnikova_metoda(harmf, a, b)[0]}\n")
#exp fce
print(f"integrace obdelnikovou metodou: {obdelnikova_metoda(exp, a, b)[0]}")
print(f"integrace lichobeznikovou metodou: {lichobeznikova_metoda(exp, a, b)}")
print(f"integrace rombergovou metodou: {obdelnikova_metoda(exp, a, b)[0]}\n")
```

✓ 0.0s

```
integrace obdelnikovou metodou: 15.5
integrace lichobeznikovou metodou: 15.50005101520253
integrace rombergovou metodou: 15.5

integrace obdelnikovou metodou: 2.1819730701192612
integrace lichobeznikovou metodou: 2.181949875965299
integrace rombergovou metodou: 2.1819730701192612

integrace obdelnikovou metodou: 1.7182818284590453
integrace lichobeznikovou metodou: 1.7182964381834482
integrace rombergovou metodou: 1.7182818284590453
```

Výsledky všech metod jsou velmi blízko k analytickému řešení, rozdílem jsou pouze malé odchylky způsobené nejspíše zaokrouhlováním a celkově tím, jakým způsobem zachází

python s float datovým typem/objectem. Bezpečně lze říci, že z naměřených výsledků lze pozorovat přesnost na 4 desetinná místa pro všechny metody.