Matematický software

Zápočtový dokument

Jméno: Pham Thanh Tung

Kontaktní email: tomas.phamt@gmail.com

Datum odevzdání: 27.9.2023

Odkaz na repozitář: https://github.com/Pham99/MSW

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:
Porovnával jsem klasický Python s modulem Numpy.
Vybral jsem tyhle matematické problémy:
1. Skalární součin
2. Determinant matice
3. Násobení matic
4. Transpozice matice
5. Určitý integrál

Malá ukázka kódu:

```
# vytvoříme 2 vektory náhodných čísel
pole_1_np = numpy.random.randint(1,9,(10_000_000))
pole_2_np = numpy.random.randint(1,9,(10_000_000))

# přetypujeme na standartní python list
list_1_pth = pole_1_np.tolist()
list_2_pth = pole_2_np.tolist()

# skalární součin běžným pythonem
def skalar_pth(seznam1, seznam2):
    return sum([i[0]*i[1] for i in zip(seznam1, seznam2)])

# porovnání
cas_np, out_np = timer(numpy.dot, pole_1_np, pole_2_np)
cas_pth, out_pth = timer(skalar_pth, list_1_pth, list_2_pth)

print(f"Výsledek numpy: {out_np}\t čas: {cas_np} sekund")
print(f"Výsledek python: {out_pth}\t čas: {cas_pth} sekund")
✓ 1.0s
```

Nejdřív se vytvoří data se kterými budeme pracovat. Numpy využije svoje vlastní pole, Python bude pracovat s běžnými listy.

O časování se postará funkce timer(), který funguje jako wrapper.

```python
# časovač
def timer(funkce, *argument):
    start_timer = time.perf_counter_ns()
    vysledek = funkce(*argument)
    end_timer = time.perf_counter_ns()
    return (end_timer - start_timer) / 1_000_000_000, vysledek
```

Další příklad implementace v běžném Pythonu

```python
# funkce na nasobení matic
def nasob_matic(matice_1, matice_2):
    sirka_matice_1 = len(matice_1[0])
    sirka_matice_2 = len(matice_2[0])
    vyska_matice_2 = len(matice_2)
    vyska_matice_1 = len(matice_1)

    if sirka_matice_1 != vyska_matice_2:
        raise ValueError("Šířka první matice musí být roven

    vysledek = []
    for n in range(vyska_matice_1):
        radek = []
        for y in range(sirka_matice_2):
            prvek = 0
            for i in range(sirka_matice_1):
                prvek += matice_1[n][i] * matice_2[i][y]
            radek.append(prvek)
        vysledek.append(radek)
    return vysledek
```

Zbytek kódu je samozřejmě dostupná v repositáři.

Výsledky:

Úloha	Počet prvků	Čas Python	Čas Numpy
Skalární součin	20000000	828.9215	4.084 ms
Determinant	100	1751.3888 ms	0.0983 ms
Násobení matic	20000	261.9368 ms	0.5916 ms
Transpozice	25000000	1965.656 ms	0.0092 ms
Určitý integrál	n/a	0.0213 ms	128.0721 ms

Závěr: Numpy byl ve většině případů rychlejší. Toto není překvapením, jelikož Numpy používá homogenní pole. Prvky pole jsou v paměti seřazeni vedle sebe a používá vektorizované operace, které provádí operace na celých polích. Výjimkou byl určitý integrál, kde byl python rychlejší. Numpy měl přesnější výsledek, takže asi tam došlo ke ztrátě rychlosti.

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:
doplňte

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:
Zvolené metody:
Přímá - Gaussova eliminační metoda (LU dekompozice)
Iterační – Jacobiho metoda

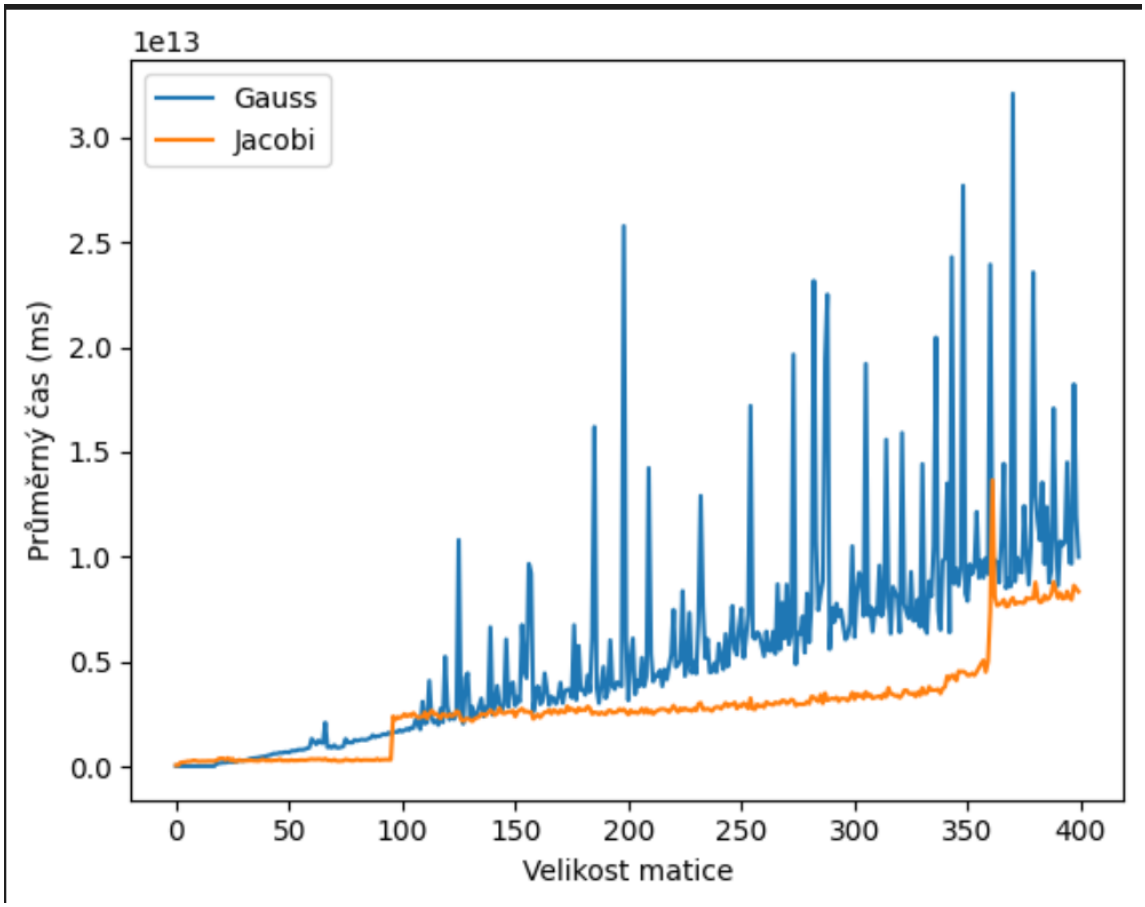
Napsal jsem Jacobiho metodu na hledání kořenů a generátor diagonálně dominantních matic.

```python
# Jacobiho iterační metoda
def jacobi(A, b, pocet_iteraci, tolerance):
    x = numpy.ones(len(A))
    D = numpy.diag(A)
    L = numpy.tril(A, k= -1)
    U = numpy.triu(A, k= 1)
    old_x = 0
    for i in range(pocet_iteraci):
        x = (b - numpy.matmul((L + U), x)) / D
        #pokud je rozdíl součtů současnýho a minulého výsledku menš
        if numpy.abs(numpy.sum(x) - numpy.sum(old_x)) < tolerance:
            return x
        old_x = x
    return x

# funkce na generování diagonálně dominantní matice
def diag_dom_matice(velikost):
        A = numpy.random.randint(1, 9, (velikost, velikost))
        b = numpy.random.randint(1, 9, velikost)
        A = A + ((numpy.eye(len(A))) * (velikost - 1) * 9)
        return A, b
```

Porovnal jsem to na maticích o velikostech 0 až 400 a každá matice se spočítala pět krát. Časy byli zprůměrovány a vykresleny na graf.

```python
velikost_matic = 400
prumer_casy_gauss = []
prumer_casy_jacobi = []
for i in range(velikost_matic):
    casy_gauss = []
    casy_jacobi = []
    for y in range(5):
        A, b = diag_dom_matice(i)
        cas_gauss, vysledek_gauss = timer(numpy.linalg.solve, A, b)
        cas_jacobi, vysledek_jacobi = timer(jacobi, A, b, 20, 0.0001)
        casy_gauss.append(cas_gauss)
        casy_jacobi.append(cas_jacobi)
    prumer_casy_gauss.append(numpy.mean(casy_gauss))
    prumer_casy_jacobi.append(numpy.mean(casy_jacobi))
```



Závěr: Přímá metoda je vhodná pro matice velikosti 1 až 25 a kolem 100. Pro všechny ostatní případy je lepší použít iterační metodu.

4. Interpolace a aproximace funkce jedné proměnné

Zadání:
Během měření v laboratoři získáte diskrétní sadu dat. Často potřebujete data i mezi těmito diskrétními hodnotami a to takové, které by nejpřesněji odpovídaly reálnému naměření. Proto je důležité využít vhodnou interpolační metodu. Cílem tohoto zadání je vybrat si 3 rozdílné funkce (např. polynom, harmonická funkce, logaritmus), přidat do nich šum (trošku je v každém z bodů rozkmitejte), a vyberte náhodně některé body. Poté proveďte interpolaci nebo aproximaci funkce pomocí alespoň 3 rozdílných metod a porovnejte, jak jsou přesné. Přesnost porovnáte s daty, které měly původně vyjít. Vhodnou metrikou pro porovnání přesnosti je součet čtverců (rozptylů), které vzniknou ze směrodatné odchylky mezi odhadnutou hodnotou a skutečnou hodnotou.

Řešení:
doplňte

5. Hledání kořenů rovnice

Zadání:
Vyhledávání hodnot, při kterých dosáhne zkoumaný signál vybrané hodnoty je důležitou součástí analýzy časových řad. Pro tento účel existuje spousta zajímavých metod. Jeden typ metod se nazývá ohraničené (například metoda půlení intervalu), při kterých je zaručeno nalezení kořenu, avšak metody typicky konvergují pomalu. Druhý typ metod se nazývá neohraničené, které konvergují rychle, avšak svojí povahou nemusí nalézt řešení (metody využívající derivace). Vaším úkolem je vybrat tři různorodé funkce (například polynomiální, exponenciální/logaritmickou, harmonickou se směrnicí, aj.), které mají alespoň jeden kořen a nalézt ho jednou uzavřenou a jednou otevřenou metodou. Porovnejte časovou náročnost nalezení kořene a přesnost nalezení.

Řešení:
Zvolené metody:
Ohraničená – Půlení intervalu
Neohraničená – Newtonova metoda

Zvolené funkce:

```python
# polynomiální funkce
def polynom(x):
    return 6*x**3 - x**2 - 1

# logaritmická funkce
def log(x):
    return 2*numpy.log2(x - 1) + 2

# goniometrická funkce
def goniometric(x):
    return 3*numpy.sin(x) + x/5
```

Výsledky:

Funkce	Metoda	Čas	Výsledek
Polynomiální	Bisekce	0.0001745 s	0.6118583185020725
	Newton	0.0006696 s	0.6118583185012075
Logaritmická	Bisekce	0.0001341 s	1.4999999999987725
	Newton	0.0003270 s	1.5
Goniometrická	Bisekce	0.0000838 s	3.3680609377279325
	Newton	0.0001513 s	3.3680609377262827

Závěr:
Metoda Bisekce byla ve všech případech rychlejší. Přesnost mají podobnou v řádu sto miliard^{-1}, ačkoliv Newtonova metoda našla přesnou hodnotu kořenu. I se špatným odhadem je metoda Bisekce rychlejší.

6. Generování náhodných čísel a testování generátorů

Zadání:
Tento úkol bude poněkud kreativnější charakteru. Vaším úkolem je vytvořit vlastní generátor semínka do pseudonáhodných algoritmů. Jazyk Python umí sbírat přes ovladače hardwarových zařízení různá fyzická a fyzikální data. Můžete i sbírat data z historie prohlížeče, snímání pohybu myší, vyzvání uživatele zadat náhodné úhozy do klávesnice a jiná unikátní data uživatelů.

Řešení:
Pomocí modulu time, pyautogui a psutil jsem načetl data z počítače jako jsou aktuální čas, pozice kurzoru a využití systému. Ty data jsem dal do cyklu, kde se po každý rundě načítají nová data a všechny se spojí matematickými operacemi. Cílem toho bylo, zvýšit počet proměnných, a tím pádem snížit jejich vliv na výsledku, abychom minimalizovali pravděpodobnost k dostání stejného semínka.

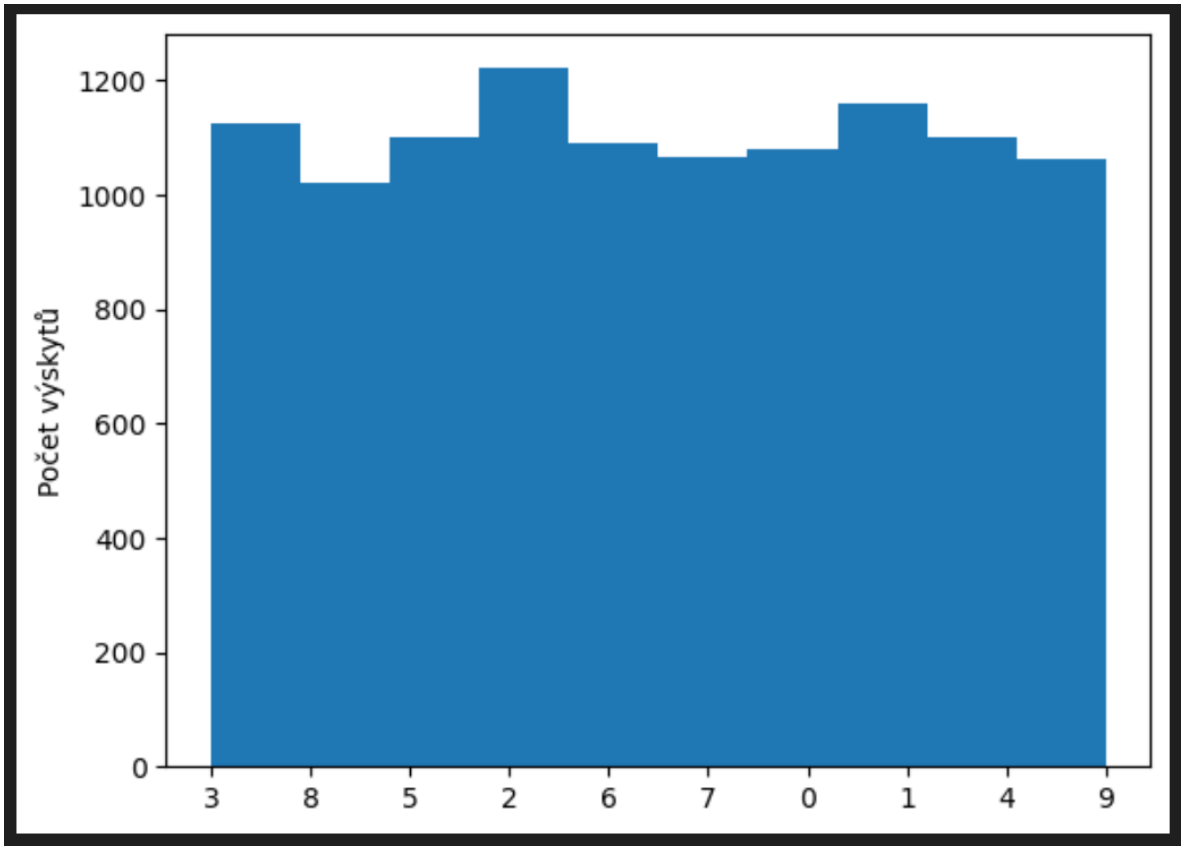
```
def seed_generator():
    # načtení času systému
    a = time.time()
    # vezmem jenom desetinnou část
    a = int((a - int(a)) * 1000000)

    # načteme souřadnice kurzoru
    x, y = pyautogui.position()

    # načteme data o využití systémových zdrojů, ale jen
    c = int(str(psutil.disk_io_counters()[3])[-4:])
    for _ in range(5):
        b = int(psutil.cpu_percent())
        d = int(str(psutil.net_io_counters()[1])[-4:])
        e = int(str(psutil.virtual_memory()[3])[-4:])
        c = (c * e + y - b + x * d)
    return c + a
```

Generátor jsem pak testoval pomocí frekvence výskytů čísel. Běžné generátory by měli mít, s velkým počtem hodnot, uniformní rozložení, což mému generátoru odpovídá.

```
# testování generátoru
cisla = ""
for _ in range(500):
    cisla = cisla + str(seed_generator())
print(cisla)
pyplot.hist(list(cisla))
pyplot.ylabel("Počet výskytů")
pyplot.show()
```


Počet výskytů
1200
1000
800
600
400
200
0
3
8
5
2
6
7
0
1
4
9

7. Metoda Monte Carlo

Zadání:
Metoda Monte Carlo představuje rodinu metod a filozofický přístup k modelování jevů, který využívá vzorkování prostoru (například prostor čísel na herní kostce, které mohou padnout) pomocí pseudonáhodného generátoru čísel. Jelikož se jedná spíše o filozofii řešení problému, tak využití je téměř neomezené. Na hodinách jste viděli několik aplikací (optimalizace portfolia aktiv, řešení Monty Hall problému, integrace funkce, aj.). Nalezněte nějaký zajímavý problém, který nebyl na hodině řešen, a získejte o jeho řešení informace pomocí metody Monte Carlo. Můžete využít kódy ze sešitu z hodin, ale kontext úlohy se musí lišit.

Řešení:
Jako problém jsem si vybral Narozeninový problém, je to úloha, kde se hledá počet lidí, tak aby byla 50% pravděpodobnost, že alespoň 2 z nich mají stejný den a měsíc narozenin. (za předpokladu, že pravděpodobnost narození je v každém dni stejný, že rok má 365 dní, a že dvojčata a další n-čata neexistují)

Napsal jsem funkci, která do množiny generuje náhodná čísla reprezentující dny v roce podle počtu lidí, které chceme. Pokud se délka množiny liší od počtu lidí, znamená to, že byli generovány duplicity, což simuluje shoda narozenin dvou lidí. Počet shod vydělíme počtem iterací a tím získáme pravděpodobnost, že alespoň 2 lidi mají stejné narozeniny pro daný počet lidí.

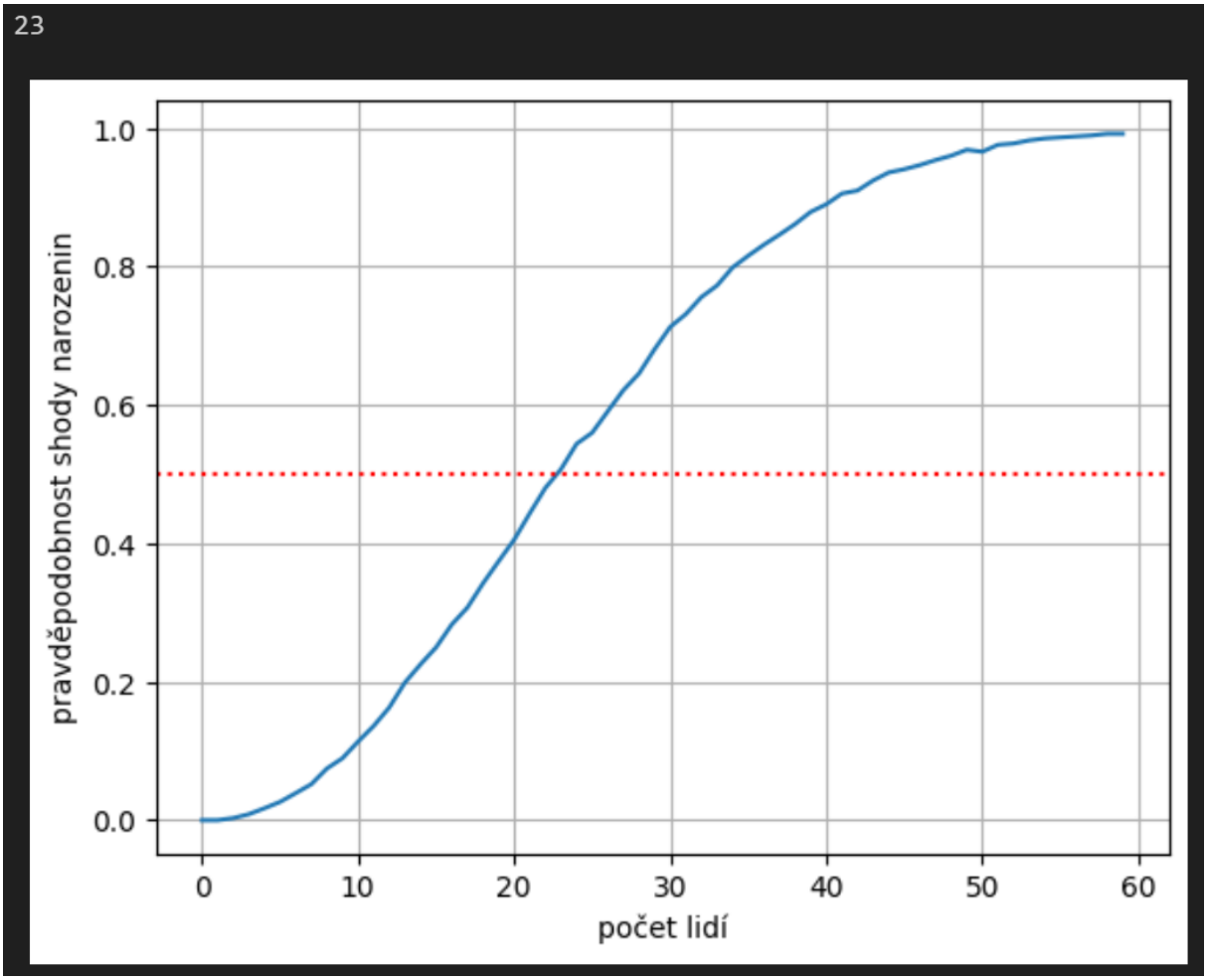
```
def narozeniny(pocet_iteraci: int, pocet_lidi: int) -> int:
    shoda_narozenin = 0
    pravdepodobnosti = []
    for iterace in range(1, pocet_iteraci + 1):
        # pomocí komprehenze generujeme n náhodných čísel, reprezentující dny v roce
        bdays = {random.randint(1, 365) for _ in range(pocet_lidi)}
        # nerovnost znamená ze v množině byli generovány duplicity a tudíž shoda narozenin
        if len(bdays) != pocet_lidi:
            shoda_narozenin += 1
        pravdepodobnosti.append(shoda_narozenin/iterace)
    return pravdepodobnosti
```

Tuto funkci zkusíme na skupiny 0 až 59 lidí. Najdeme počet lidí, jejichž pravděpodobnost se nejvíc blíží 50 %. Vykreslíme graf.

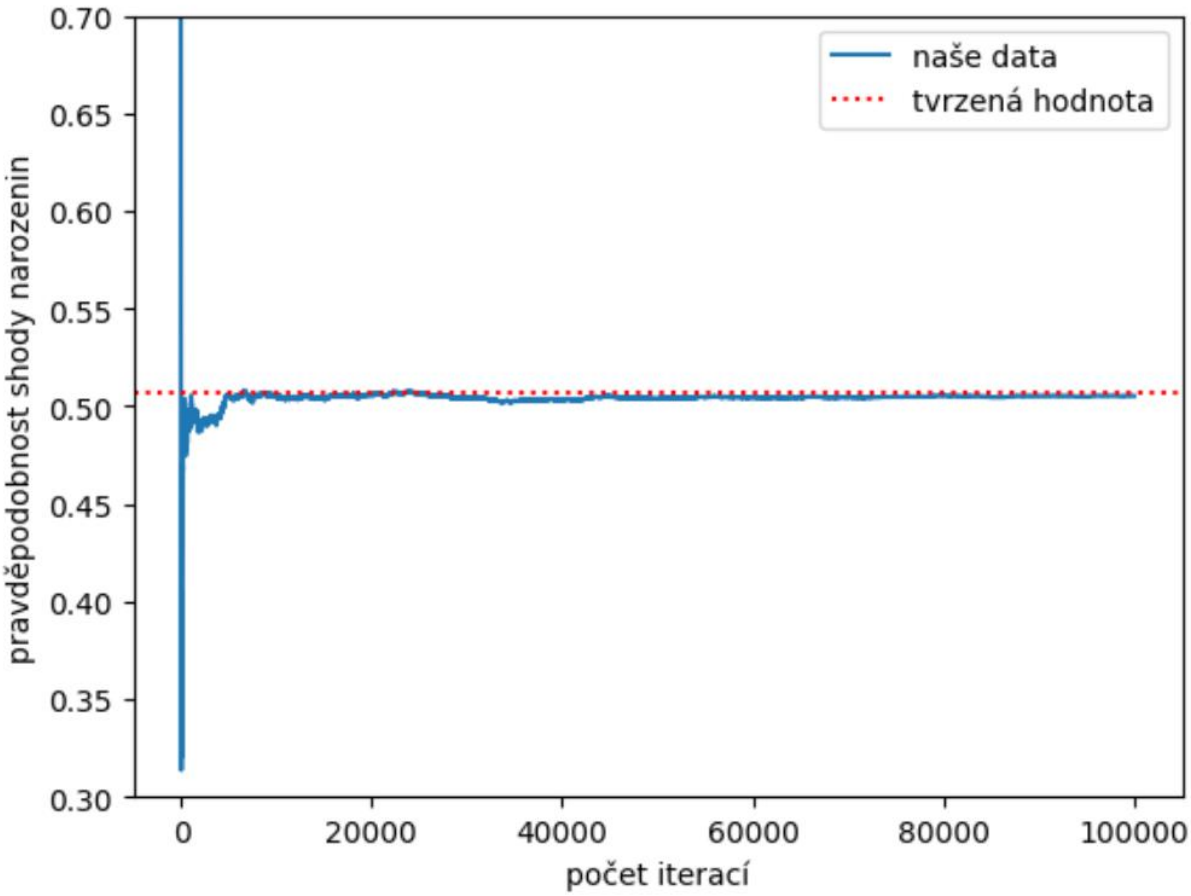
```
y = []
for pocet_lidi in range(60):
    x = narozeniny(10000, pocet_lidi)
    y.append(x[-1])

# hledání indexu (počtu lidí), která se nejvíc blíží 0.50
z = [abs(i - 0.50) for i in y]
print(z.index(min(z)))
```

```
23
```



Dostali jsme číslo 23, což může být překvapivé.
Z toho vychází Narozeninový paradox, říká, že pro skupinu 23 lidí je 50 % (přesněji 50,7) pravděpodobnost nalezení shody narozenin. Je to takzvaný „veridical paradox“. Jeho tvrzení se na první pohled jeví překvapivým nebo dokonce špatným, ale je to pravda.
To si ještě ověříme pomocí naší funkce. Zavoláme ho s počtem lidí 23 a počtem iterací 100000. Vykreslíme graf, kde uvidíme jak se pravděpodobnost mění v závislosti na počtu iterací.



Pro skupinu 23 je pravděpodobnost hodně blízká k 50.7 %.

Spočítáme to ještě jednou, ale 1000 krát a s průměrem.

```
x = []
for _ in range(1000):
    x.append(narozeniny(1000, 23)[-1])

print(f"průměr je: {sum(x)/1000 * 100}")
```

✓ 8.1s

```
průměr je: 50.73079999999993
```

Závěr: Pomocí metody Monte Carlo jsme našli řešení Narozeninového problému a potvrdili Narozeninový paradox.

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:
doplňte

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:
doplňte

10. Řešení obyčejných diferenciálních rovnic

Zadání:
Diferenciální rovnice představují jeden z nejdůležitějších nástrojů každého přírodovědně vzdělaného člověka pro modelování jevů kolem nás. Vaším úkolem je vybrat si nějakou zajímavou soustavu diferenciálních rovnic, která nebyla zmíněna v sešitech z hodin a pomocí vhodné numerické metody je vyřešit. Řešením se rozumí vizualizace jejich průběhu a jiných zajímavých informací, které lze z rovnic odvodit. Proveďte také slovní okomentování toho, co lze z grafu o modelovaném procesu vyčíst.

Řešení:
doplňte