Matematický software

Zápočtový dokument

Jméno: Kateryna Leonova

Kontaktní email: leonova.k@seznam.cz

Datum odevzdání: 25.09.2024

Odkaz na repozitář: github.com/CptLeoxo/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 více než polovinu z probraných témat. To prokážete vyřešením vybraných úkolů. V tomto dokumentu naleznete celkem 10 zadání, která odpovídají probíraným tématům. Vyberte si 6 zadání, vypracujte je a odevzdejte. Pokud bude všech 6 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 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:

1) **Skalární součin**

Standardní Python:

```
def skalar_python(a, b):
    return sum(x * y for x, y in zip(a, b))
```

NumPy:

```
def skalar_np(a, b):
    return np.dot(a, b)
```

2) **Integrál**

Standardní Python:

```
def integral_cp(n):
    dx = 1 / n
    return sum((i * dx) ** 2 for i in range(n)) * dx
```

NumPy:

```
def integral_scipy():
    return quad(lambda x: x**2, 0, 1)[0]
```

3) **Násobení matic**

Standardní Python:

```
def matice_nas_cp(A, B):
```

```
    return [[sum(a * b for a, b in zip(A_row, B_col)) for B_col in zip(*B)] for A_row in A]
```

NumPy:

```
def matice_nas_np(A, B):
    return np.dot(A, B)
```

4) Generování náhodných čísel

Standardní Python (knihovna random):

```
def random_cp(n):
    return sum(random.random() for _ in range(n)) / n
```

NumPy (take s použitím knihovny random):

```
def random_mean_np(n):
    return np.mean(np.random.rand(n))
```

5) Výpočet faktorialu

Standardní Python:

```
def faktorial_cp(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

NumPy:

```
def faktorial_math(n):
    return math.factorial(n)
```

Ve všech případech knihovna NumPy byla rychlejší jak standardní Python. **Výsledky:**

Skalární součin čistý Python: 0.0010149478912353516 vteřin.
Skalární součin NumPy: 0.0 vteřin.
Vypočet určitého integrálu s pomocí čistého Pythonu: 0.39205503463745117 vteřin.
Vypočet určitého integrálu s pomocí SciPy: 0.0010106563568115234 vteřin.
Vypočet násobení matic s pomocí čistého Pythonu: 18.055609941482544 vteřin.
Vypočet násobení matic s pomocí NumPy: 0.025007247924804688 vteřin.
Generování nahodných čísel s pomocí čistého Pythonu: 0.14590764045715332 vteřin.
Generování nahodných čísel s pomocí NumPy: 0.016010522842407227 vteřin.
Vypocet faktorialu s pomoci cisteho Pythonu: 5.535025 vteřin.
Vypocet faktorialu s pomoci knihovny math: 0.337012 vteřin.

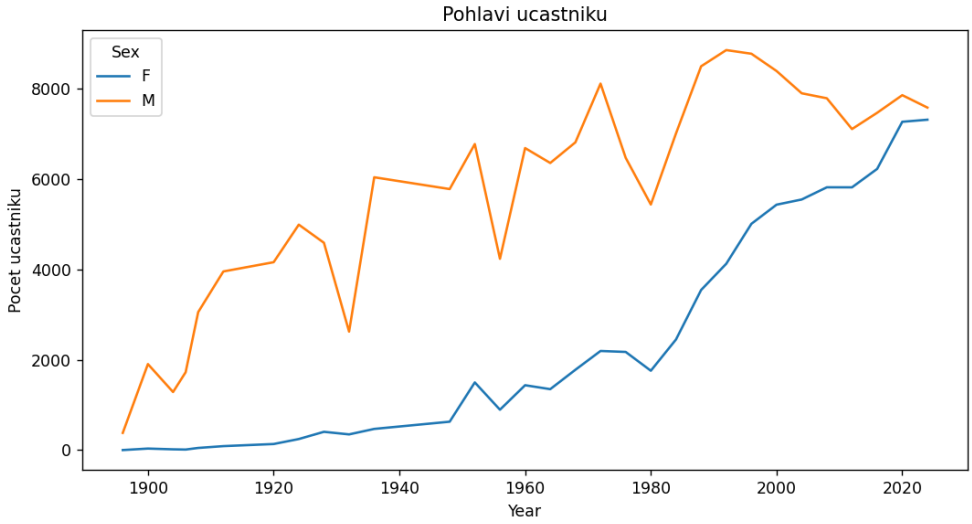
2. Vizualizace dat

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:
Pro řešení druhého zadání jsem si vzala dataset s informací o Olympijských hrách.

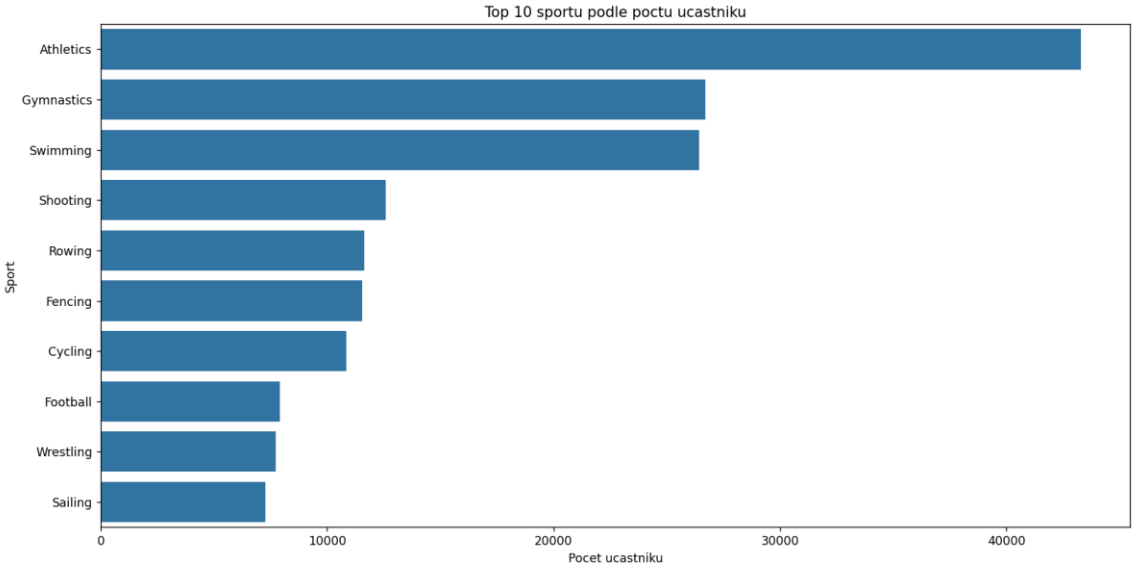
Rozdíl počtu účastníků v závislostí na roce a pohlavi

```
gender = df.groupby(['Year', 'Sex']).size().unstack(fill_value=0)
gender.plot(kind='line', figsize=(10,5))
plt.title('Pohlavi ucastniku')
plt.ylabel('Pocet ucastniku')
plt.show()
```



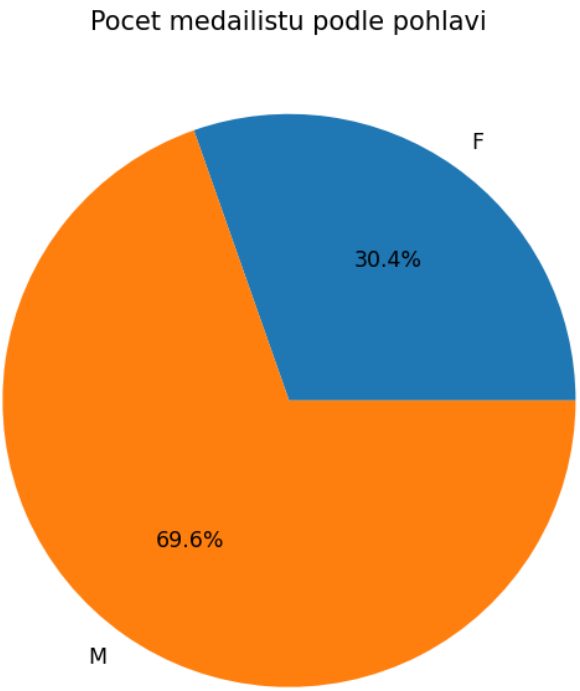
Top 10 sportů podle čísla účastníků

```
sport_count = df['Sport'].value_counts().head(10)
sns.barplot(x=sport_count.values, y=sport_count.index)
plt.title('Top 10 sportu podle poctu ucastniku')
plt.xlabel('Pocet ucastniku')
plt.show()
```



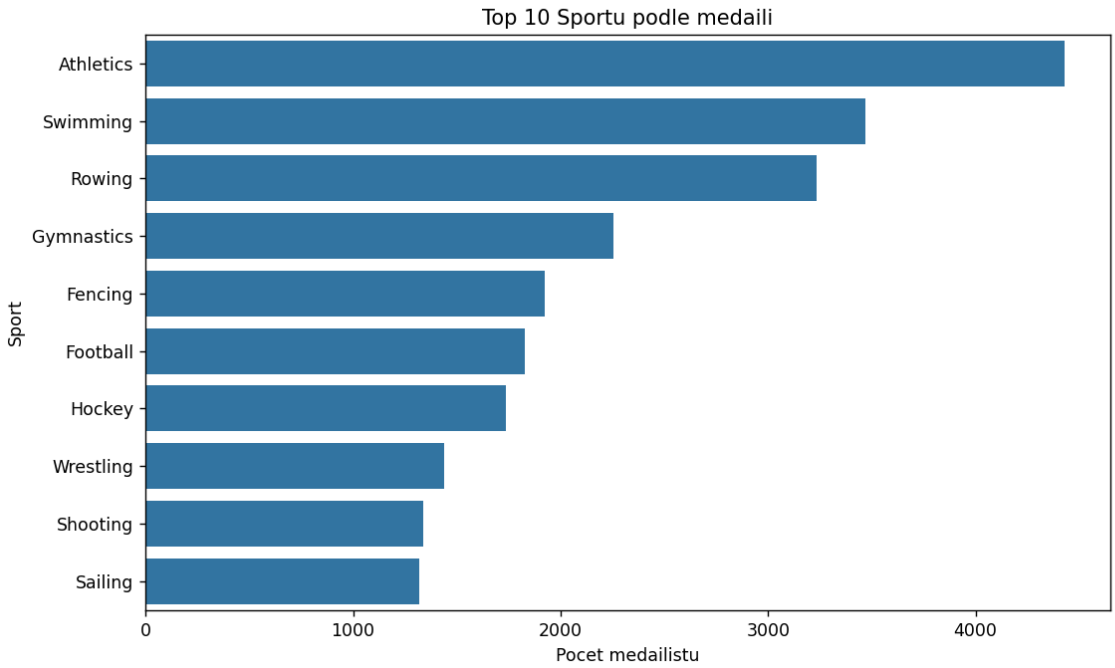
Počet medailistu podle pohlaví

```
medaile_podle_pohlavi = df[df['Medal'] != 'No medal'].groupby('Sex').size()
medaile_podle_pohlavi.plot(kind='pie', autopct='%1.1f%%', figsize=(6,6))
plt.title('Pocet medailistu podle pohlavi')
plt.show()
```



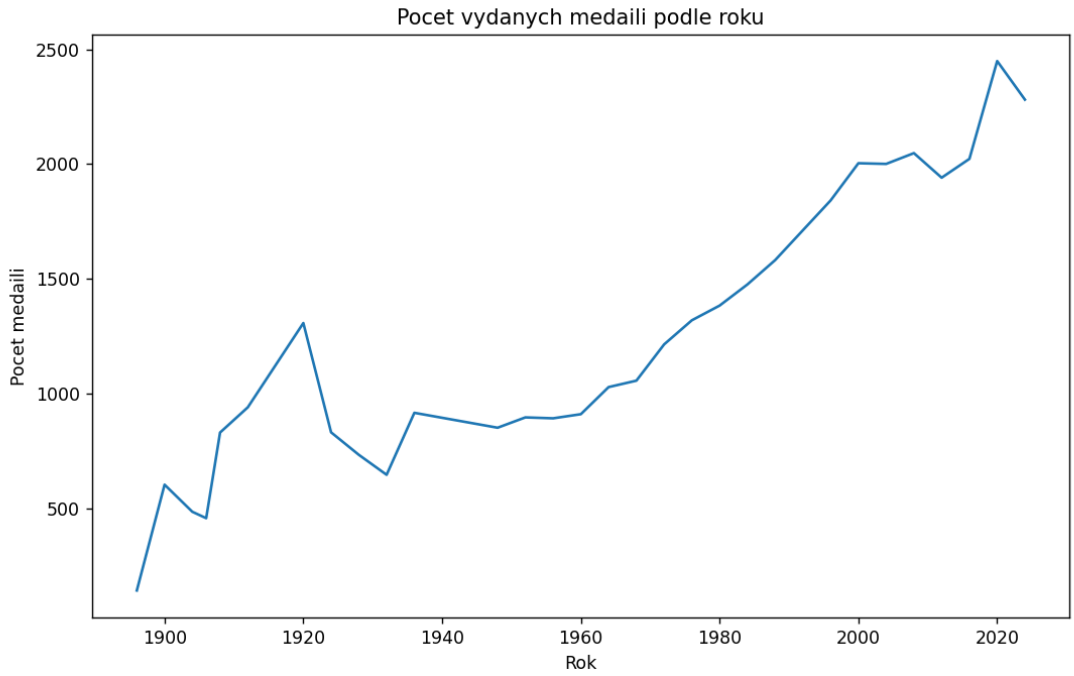
Top 10 sportů podle počtu medailistů

```
top_sporty = df[df['Medal'] != 'No medal']['Sport'].value_counts().head(10)
plt.figure(figsize=(10,6))
sns.barplot(x=top_sporty.values, y=top_sporty.index)
plt.title('Top 10 Sportu podle medaili')
plt.xlabel('Pocet medailistu')
plt.ylabel('Sport')
plt.show()
```



Počet vydaných medaili podle roku

```
plt.figure(figsize=(10,6))
sns.lineplot(x=medals_by_year.index, y=medals_by_year.values)
plt.title('Pocet vydanych medaili podle roku')
plt.xlabel('Rok')
plt.ylabel('Pocet medaili')
plt.show()
```


Pocet vydanych medaili podle roku
Pocet medaili
2500
2000
1500
1000
500
1900
1920
1940
1960
1980
2000
2020
Rok

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:

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:

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:

Vybrala jsem si funkce:

1) Polynomiální x**3 - x - 2
2) Exponenciální np.exp(x) – 2
3) Harmonická funkce np.sin(x) - 0.5

Bisekcni metoda:

```
def bisekce(func, a, b, tol=1e-6):
    if np.sign(func(a)) == np.sign(func(b)):
        raise ValueError(f"f(a) a f(b) musi mit ruzna znamenka: f({a})={func(a)},
f({b})={func(b)}")
    koren = optimize.bisect(func, a, b, xtol=tol)
    return koren
```

Newtonova metoda:

```
def newton(func, x0, tol=1e-6):
    koren = optimize.newton(func, x0, tol=tol)
    return koren
```

Newtonová metoda mi vychazela rychlejí. **Výsledky:**

Polynomiální funkce (bisekce): 1.5213804244995117, čas: 0.0
Polynomiální funkce (Newton): 1.521379706804575, čas: 0.0009999275207519531
Exponencialní funkce (bisekce): 0.6931467056274414, čas: 0.0
Exponencialní funkce (Newton): 0.6931471805597554, čas: 0.0
Harmonicka funkce (bisekce): 0.5235990252696511, čas: 0.0
Harmonicka funkce (Newton): 0.5235987755983077, čas: 0.0010001659393310547

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:

Generátor náhodného čísla podle pohybu mýši po dobu n vteřin. (S tím, že časovou hodnotu sbíraní dat se dá změnit.)

Pozice myši jsou v listu x, y.

Program sbíra data s pomocí funkce:

```
def sber_dat(doba=5):
    print(f"Hybejte myši po dobu {doba} vteřin.")

    with Listener(on_move=pozice) as listener:
        time.sleep(doba)
        listener.stop()

    print(f"Nasbíráno {len(mys_pozice)} pozic myši.")
    # print(mys_pozice)
    return generator_seminka()
```

Potom je potřeba vygenerovat seminko:

```
def generator_seminka():
    seminko = sum([x + y for x, y in mys_pozice]) % (2**32)
    return seminko
```

Funkčnost kódu:

```
if __name__ == "__main__":
    seminko = sber_dat(5)

    random.seed(seminko)
    print(f"Vygenerované semínko: {seminko}")
    print(f"Náhodné číslo: {random.randint(1, 100)}")
```

Výsledek:

Hybejte myši po dobu 5 vteřin.

Nasbíráno 349 pozic myši.
Vygenerované semínko: 494372
Náhodné číslo: 36

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:
Tento úkol jsem chtěla pojat kreativněji. Proto jsem si vzpoměla na Jackpot. Hazardní hra, kde pro výhru je potřeba získat 3 stejné znaky v jednom řadku.

```
def slot(n):
    jackpot_vyhry = 0

    for _ in range(n):
        # simulace otacek valcu se tremi symboly
        tah = [random.choice(znaky) for _ in range(3)]

        # kontrola, zda hrac vyhral jackpot
        if tah[0] == tah[1] == tah[2]:
            jackpot_vyhry += 1

    return jackpot_vyhry

n = 10000
jackpot_vyhry = slot(n)

jackpot_pr = jackpot_vyhry / n
```

Výsledek:

Vyhráli jackpot: 1147 krát.
Pravděpodobnost výhry jackpotu: 11.47%

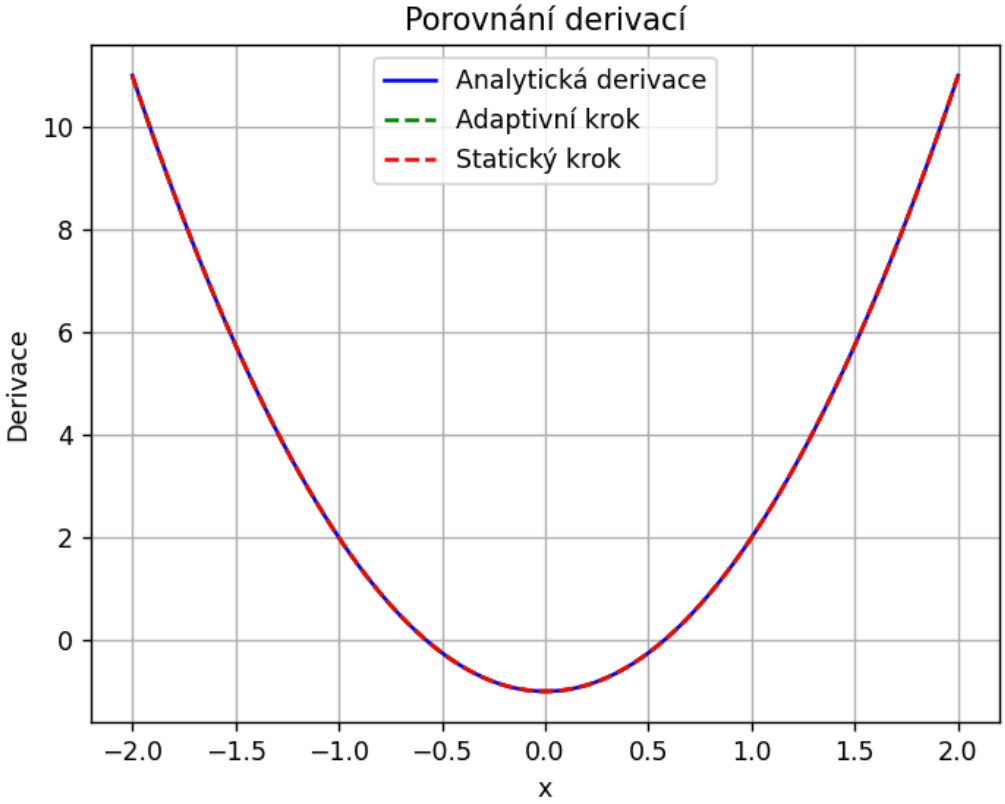
8. Derivace funkce jedné proměnné

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:

Fce : x**3 - x - 2

```
x = sp.symbols('x')
analyticka_vyraz = f(x)
analyticka_derivace = sp.diff(analyticka_vyraz, x)

def analyticka_derivace_fce(hodnota):
    return float(analyticka_derivace.subs(x, hodnota))

def staticka_derivace(f, x, h=0.01):
    return (f(x + h) - f(x - h)) / (2 * h)

def adaptivni_derivace(f, x, tol=1e-6):
    h = 0.1
    diff = (f(x + h) - f(x - h)) / (2 * h)

    while True:
        h_new = h / 2
        diff_new = (f(x + h_new) - f(x - h_new)) / (2 * h_new)

        if np.abs(diff_new - diff) < tol:
            return diff_new
        diff = diff_new
        h = h_new
```


Porovnání derivací
Analytická derivace
Adaptivní krok
Statický krok
Derivace
x
10
8
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

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:

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: