Matematický software

Zápočtový dokument

Jméno: Ondřej Švorc

Kontaktní email: ondrejsvorc@email.cz

Datum odevzdání: 31. 5. 2024

Odkaz na repozitář: https://github.com/ondrejsvorc/UJEP/tree/main/MSW/zapocet

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.

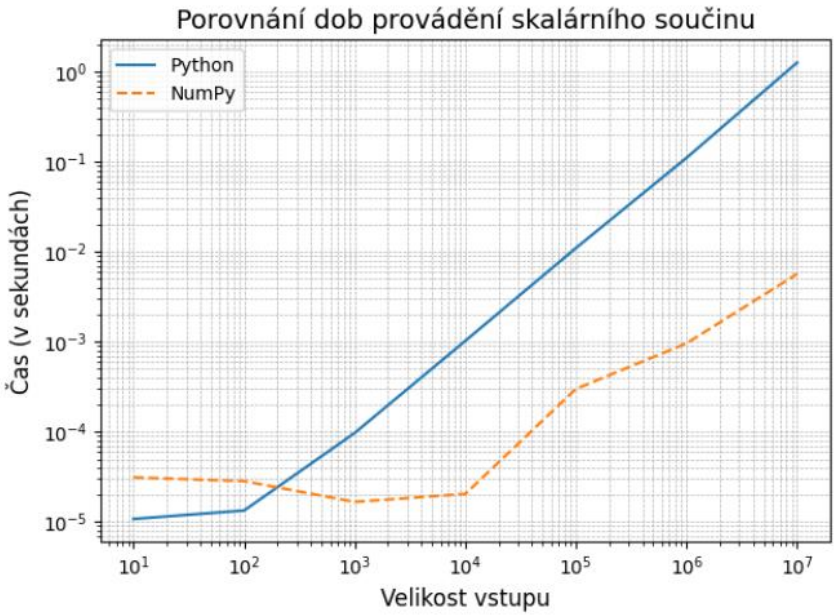
Řešení:

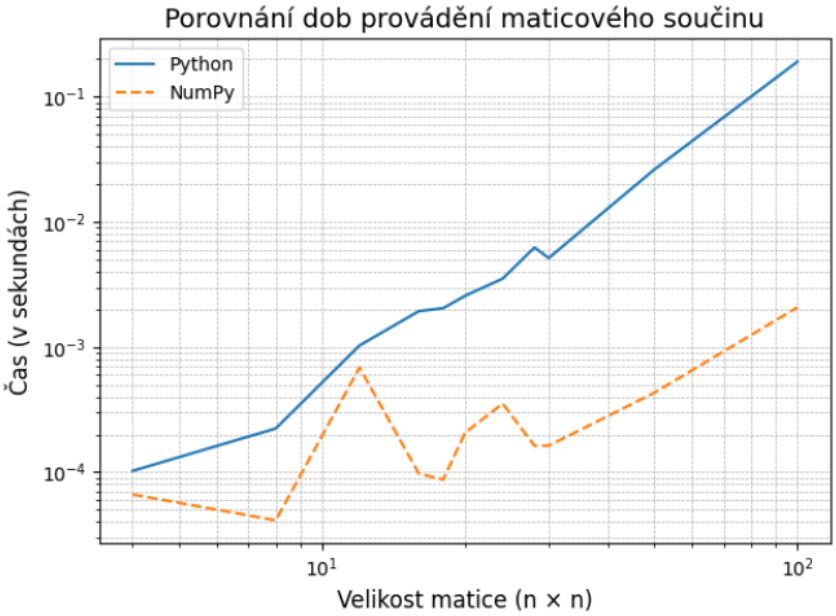
Ve svém řešení jsem provedl měření na **5** různých úlohách:

1. Porovnání dob provádění **skalárního součinu**
2. Porovnání dob provádění **výpočtu derivace**
3. Porovnání dob provádění **maticového součinu**
4. Porovnání dob výpočtu **faktoriálu**
5. Porovnání dob výpočtu **statistických funkcí**

V měřeních je primárně porovnáván standardní **Python** s knihovnou **NumPy**. V případě 2. měření je porovnáván **Python** s knihovnou **SymPy**. Všechna měření jsou doplněna o **grafy**, z nichž lze lehce vydedukovat rychlost Pythonu a porovnávané knihovny při různě velkých vstupech.

U téměř všech měření rychlostně zvítězila daná knihovna nad Pythonem. Při velmi malých vstupech (cca do velikosti 100) byla u 1. a 4. měření porovnávaná knihovna pomalejší, ale to se při překročení zmíněné hranice změnilo, a to již nastálo. Zajímavé bylo porovnání výpočtu statistických funkcí (tedy měření č. 5), kdy byl standardní Python při jakémkoliv vstupu nejrychlejší; pouze v případě kumulativní sumy od vstupu 1000 začala být knihovna NumPy nepatrně rychlejší. Další zajímavý poznatek se objevil při měření č. 4 (výpočet faktoriálu), kdy bylo rekurzivní řešení pomalejší než iterativní řešení; knihovna NumPy byla pomalejší než obě zmíněné metody pouze do zmíněného vstupu 100.





Další grafy jsou k dispozici ve vypracovaném řešení.

Vypracované řešení je k dispozici zde.

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:

Ve svém řešení jsem vizualizovat **7** různých situací:

1. Poměr výskytu seriálů a filmů na platformě Netflix
2. 10 nejvíce produkujících zemí na platformě Netflix
3. Počet filmů, vydaných v daném roce, na platformě Netflix
4. Režiséři s největším počtem titulů na platformě Netflix
5. Herci vyskytující se v největším počtu titulů na platformě Netflix
6. 50 filmů na platformě Netflix začínajících na písmeno 'A' vydaných mezi lety 2000–2020
7. 50 seriálů na platformě Netflix začínajících na písmeno 'A' vydaných mezi lety 2000–2020

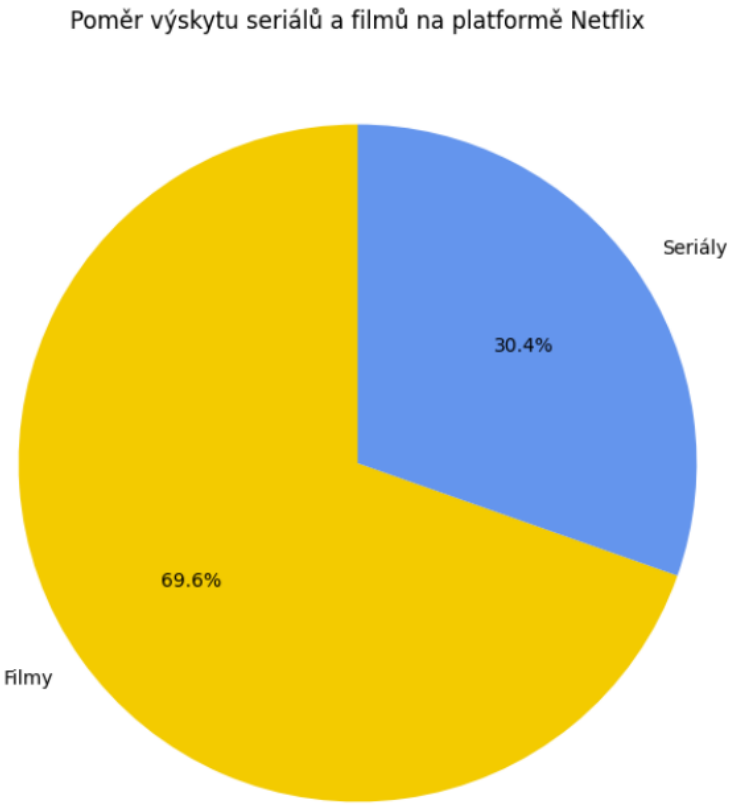
Pro vizualizaci situací jsem použil **5** různých grafů:

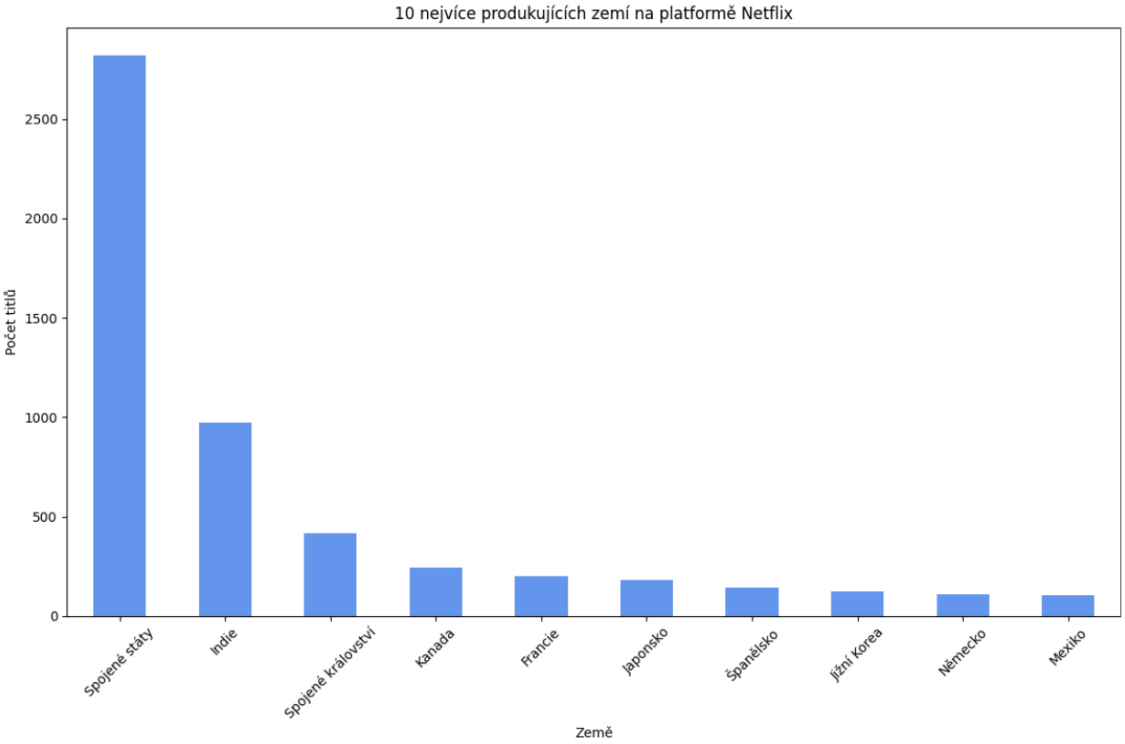
1. **koláčový graf**
2. **sloupcový graf**
3. **histogram**
4. – 5. **vodorovný pruhový graf**
6. – 7. **spojnicový graf**

Vybraná datová sada se zabývá filmy a seriály dostupnými na platformě Netflix. Obsahuje různé informace, jako je typ titulu, režisér, obsazení, země výroby, rok vydání, hodnocení, délka trvání apod. Textová data této datové sady jsou primárně v angličtině, a proto jsem je pro lepší interpretaci česky mluvícím člověk přeložil při vizualizacích do češtiny.

Z grafů vyplívá hned několik zajímavých faktů, a těmi jsou např.:

- cca 70 % obsahu na platformě Netflix jsou filmy; zbývajících 30 % pak seriály
- nejvíce titulů pochází ze Spojených států amerických (více než 2500)
- nejvíce filmů na platformě Netflix pochází z období okolo roku 2020
- režisér s největším počtem titulů je Ind, Rajiv Chilaka (18+ titulů)
- herec vyskytující se v největším počtu titulů je též Ind, Anupam Kher (40+ titulů)





Další grafy jsou k dispozici ve vypracovaném řešení.

Vypracované řešení je k dispozici zde.

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:

Mé řešení se skládá z **5** kroků:

1. Vybrání konkrétních funkcí a definování jejich derivace
2. Vizualizace funkcí v grafu
3. Implementace metody půlení intervalů
4. Implementace Newtonovy metody
5. Porovnání metod

Zvolené matematické funkce:
- polynomiální: $x^3 - 2x^2 - 5x + 6$
- exponenciální: $e^x - 4$
- harmonická: $\sin x$

Jako ohraničenou metodu jsem zvolil metodu půlení intervalů (též *metoda bisekce*) a jako neohraničenou metodu jsem zvolil Newtonovu metodu (též *Newton-Raphsonova metoda*).

Metoda bisekce byla pomalejší u nalezení kořene u polynomiální a exponenciální funkce; metoda byla pak rychlejší u nalezení kořene u harmonické funkce. Newtonsonova metoda byla naopak rychlejší u nalezení kořene u polynomiální a exponenciální funkce, ale pomalejší u nalezení kořene u harmonické funkce.

Metoda bisekce, jak již z jejího alternativního názvu metoda půlení intervalů vyplívá, je metoda, založená na půlení intervalů. Jedná se o iterativní metodu, která postupně zmenšuje interval, ve kterém se předpokládá, že je kořen funkce. Na začátku je určen interval, ve kterém se předpokládá, že se kořen nachází. Interval je v mém řešení definován pomocí počátečního a koncového bodu. Principiálně to funguje tak, že metoda rozdělí interval na dva menší intervaly a zkontroluje, ve kterém z těchto intervalů se nachází kořen. Pro pokračování je vybrán ten interval, ve kterém je kořen. Tento postup se opakuje, dokud se délka intervalu nepřiblíží požadované přesnosti (epsilon).

```python
# Metoda půlení intervalů (bisekce) - ohraničená metoda
def bisection(function, interval_start=-100, interval_end=100, epsilon=0.000001):
    start = time()
    while abs(interval_end - interval_start) > epsilon:
        interval_middle = (interval_start + interval_end) / 2
        found_root = function(interval_middle) == 0
        if found_root:
            break
        is_root_between_start_and_end = (
            function(interval_start) * function(interval_middle) < 0
        )
        if is_root_between_start_and_end:
            interval_end = interval_middle
        else:
            interval_start = interval_middle
    end = time()
    total_time = end - start
    return (interval_middle, total_time)
```

Newtonova metoda je též iterativní metoda, která využívá derivaci funkce, jejíž kořen se snažíme nalézt. Začíná se s odhadem kořene (x_0) a tento odhad může být zvolen libovolně, ale čím blíže je k pravému kořeni, tím rychleji metoda konverguje. Iterativní proces (tělo while cyklu) aktualizuje tento odhad tak, aby se přibližoval skutečné hodnotě kořenu funkce. Klíčový je tedy řádek **x0 = x0 - function(x0) / derivation_function(x0)**, kdy tato operace aktualizuje hodnotu x0 podle vzorce Newtonovy metody.

```python
# Newton-Raphsonova metoda - neohraničená metoda
def newton_raphson(function, x0, derivation_function, epsilon=0.000001):
    start = time()
    while abs(function(x0)) > epsilon:
        x0 = x0 - function(x0) / derivation_function(x0)
    end = time()
    total_time = end - start
    return (x0, total_time)
```

Vypracované řešení je k dispozici zde.

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:

Výsledný **seed** (semínko) se generuje na základě následujících **5** bodech:

1. Získávání dat z vestavěného generátoru pseudonáhodných čísel
2. Získávání dat z momentálních statistik HW komponent
3. Získávání dat z informací o počítači v síti
4. Získávání dat z informací o HW specifikací počítače
5. Získávání momentálního času v milisekundách

Pro získání daných dat jsem využil následující **moduly**:
1. random = vygeneruje pseudonáhodné číslo (vestavěné)
2. psutil = získá data v reálném čase o vytíženosti procesoru, paměti a disku
3. socket = získá data o názvu zařízení a jeho IP adresu
4. platform = získá data o HW architektuře (32/64bit, OS, GPU, ...)
5. hashlib = hashuje získaná data

Mnou vytvořený generátor semínka sbírá pro jeho vygenerování data z různých zdrojů. Těmito zdroji jsou myšleny data vygenerovaná z vestavěného generátoru pseudonáhodných čísel (modul random) nebo třeba statistik hardwarových komponent v reálném čase apod. Tyto data jsou dále zpracovány a sloučeny do jednoho řetězce, který je dále zahashován, a tím vznikne semínko. Tímto způsobem je zajištěna vysoká míra náhodnosti při generování čísel, a to jest klíčové.

Informace o síti ani HW specifikace počítače sice generují na stejném zařízení stejná data, ale i přesto mohou být někdy v budoucnu, náhodně obměněny (např. přidělení nové síťové konfigurace službou DHCP, výměna HW komponent). Dané informace jsou také na každém zařízení různá. Nejdůležitějšími, dynamicky získávanými daty jsou tedy data o vytíženosti procesoru, paměti a disku.

Vypracované řešení je k dispozici zde.

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:

Dle nalezených skript a dalších zdrojů jsem naimplementoval **4** způsoby derivace:

1. Dopředná numerická derivace
2. Zpětná numerická derivace
3. Centrální numerická derivace
4. Derivace s adaptabilním krokem

Derivovaná funkce: $x^2 + 2x + 2$
Derivace: $2x + 2$

Analytická derivace	**4.43999999**9999995
Numerická derivace s adaptabilním krokem	**4.43999999**012798

Derivace s adaptabilním krokem využívá všechny předchozí způsoby derivace. V porovnání s analytickou derivací se liší až od 10. desetinného místa, kdy je méně přesnější. Výsledkem derivace s adaptabilním krokem je též o 1 desetinné místo kratší. Oba způsoby dávají správný a relativně dostatečně přesný výsledek.

Vypracované řešení je k dispozici zde.

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:

1. Gaussova metoda numerické integrace
2. Obdélníková metoda
3. Lichoběžníková metoda

Zvolené matematické funkce:

- polynomiální: $x^3 - 2x^2 - 5x + 6$
- exponenciální: $e^x - 4$
- harmonická: $\sin x$

Integrace polynomiální funkce

Gausova kvadratura	**3.083**33333333334
Obdelníková metoda	**3.083**33333333333
Lichoběžníková metoda	**3.0833**248307995786

Nejpřesnější je Gausova metoda.

Integrace exponenciální funkce

Gausova kvadratura	**-2.2817**181715416086
Obdelníková metoda	**-2.2817**181715409545
Lichoběžníková metoda	**-2.2817**035618165513

Nejpřesnější je Gausova metoda.

Integrace harmonické funkce

Gausova kvadratura	**0.45969**76941320484
Obdelníková metoda	**0.45969**769413186023
Lichoběžníková metoda	**0.45969**37855300522

Nejpřesnější je Gausova metoda.

Vypracované řešení je k dispozici zde.