Matematický software

Zápočtový dokument

**Jméno:** vyplňte

**Kontaktní email:** vyplňte​

**Datum odevzdání:** vyplňte

**Odkaz na repozitář:** vyplňte

# 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. Výpočet faktoriálu**

Tato úloha se zabývá výpočtem faktoriálu. Nejdříve bez pomoci knihoven a poté pomocí

knihovny *SymPy*. Časová náročnost je následující:

Výpočet (Python): 1.8854195330059156\\

Výpočet (SymPy): 0.1536074520117836

**2. Výpočet sin(x)**

Tentokrát porovnávám 3 knihovny – *Math*, *NumPy* a *SymPy*.

Čas výpočtu (Python): 0.00030469500052277\\

Čas výpočtu (NumPy): 0.0011716900044120848\\

Čas výpočtu (Sympy): 0.6211337630084017

Python s pomocí knihovny *math* je zde nejrychlejší. Po něm je *NumPy* a poté *SymPy*.

**3. Násobení matic**

Tato úloha ukazuje efektivitu násobení matic o velikosti 3x3 pomocí knihovny *NumPy*.

Čas výpočtu (Python): 0.00018488299974706024\\

Čas výpočtu (NumPy): 0.00012147700181230903

Výsledky ukazují, že je knihovna *NumPy* výrazně rychlejší. Python využívá tři vnořené smyčky k výpočtu, zatímco *NumPy* používá vektorové operace pro rychlejší výpočet.

**4. Výpočet faktoriálu**

Tato úloha se zabývá výpočtem faktoriálu. Nejdřív pomocí Pythonu a poté pomocí

knihovny *Math*.

Čas výpočtu (Python): 0.15173245201003738\\

Čas výpočtu (Math): 0.01574059799895622\\

**5. Aritmetický průměr**

Tato úloha ukazuje délku výpočtu aritmetického průměru seznamu čísel o délce 3 524 287 čísel.

Čas výpočtu (Python): 0.031385743990540504\\

Čas výpočtu (NumPy): 0.009323335994849913

Ve výsledcích můžeme vidět, že výpočet za pomoci *NumPy* je výrazně rychlejší. Python musí projít celý seznam a získat sumu prvků, zatímco *NumPy* vypočítá průměr pomocí své vestavěné funkce mean().

# 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 tuto úlohu jsem použil datovou sadu o diskografii kapely The Cure z portálu Kaggle. K analýze a vizualizaci dat využívám knihovnu *pandas* pro manipulaci s daty a *matplotlib* pro tvorbu grafů.

**1. Načtení dat a předzpracování**

Data se načtou z CSV souboru a celé datumy z *album\_release\_date* převedu na rok *album\_release\_year*. Pro mé účely je to dostačující přesnost a navíc se tím i zlepší čitelnost grafů.

**2. Vztah mezi tóninou (key) a popularitou skladeb**

Bar plot ukazuje průměrnou popularitu skladeb v různých tóninách. Tento graf nám umožňuje vidět, zda existuje vztah mezi tóninou skladby a její oblíbeností.

**3. Histogram délky skladeb**

Histogram zobrazuje rozložení délky skladeb v minutách. Z něj lze vyčíst, jaká je typická délka skladby.

**4. Podíl instrumentálních skladeb**

Koláčový graf ukazuje procentuální podíl instrumentálních skladeb oproti skladbám s vokály, což poskytuje přehled o tom, kolik skladeb je čistě instrumentálních.

**5. Vztah mezi hlasitostí a rokem vydání**

Bar plot zobrazuje průměrnou hlasitost skladeb v různých letech. Tento graf může naznačovat změny v produkčním stylu kapely během jejich kariéry. Zde předpokládám, že hlasitost skladeb v CSV souboru byla měřena na souborech s normalizovanou hlasitostí.

**6. Vztah mezi tanečností (danceability, jak dobře se na danou skladbu tancuje) a popularitou skladeb**

Scatter plot vizualizuje vztah mezi tanečností skladeb a jejich oblíbeností, což může poskytnout informace o tom, zda tanečnost skladeb ovlivňuje jejich popularitu.

# 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:**

Pro nalezení kořenů funkcí jsem využil ohraničenou metodu půlení intervalu a neohraničenou metodu Newtonovy metody. Tolerance chyby v mé implementaci je 10-10

Funkce

Metoda půlení intervalu x=-1.9999999999708961695432663  
Časová náročnost: 0.0000440880030510015785694 sekund

Newtonova metoda x=2.0000000000000000000000000

Časová náročnost: 0.0048414449993288144469261 sekund

Funkce

Metoda půlení intervalu x=1.2599210498156026005744934

Časová náročnost: 0.0000231319936574436724186 sekund

Newtonova metoda x=1.2599210498948731906665444

Časová náročnost: 0.0038661680009681731462479 sekund

Funkce

Metoda půlení intervalu x=2.9999999998253770172595978

Časová náročnost: 0.0000237130007008090615273 sekund

Newtonova metoda x=3.0000000000000000000000000

Časová náročnost: 0.0011777550025726668536663 sekund

Z výsledků je patrné, že ohraničená metoda půlení intervalu je pomalejší.

# 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:**

Můj generátor semínek do pseudonáhodných algoritmů využívá vložené WAV soubory. Nejdříve WAV soubory otevře a načte jejich rámce. Ty se pak převedou na numpy pole. Poté využije součet absolutních hodnot jejich rámců pro vygenerování semínka. V tomto příkladu používám dva WAV soubory, s jejichž semínky se pak provede následující operace:

seed1 + seed2 / np.pi \* seed2 / seed1 / random.randint(10, 785410)

která následně vrátí finální semínko použité ve funkci random.randint(), kde je vypsáno na standardní výstup. V tomto příkladu generuji pět čísel. Výsledek vypadá takto:

Semínko: 1947876781.708661\\

Náhodná čísla: [112417021688641, 328604045480135, 90813242632018, 200443802296886, 30682555226394]

Využil jsem dva přiložené WAV soubory, *smrt.wav* a *jubilejni\_den.wav*. Tento způsob generování by mohl mít dobré využití např. při použití živého zvuku z mikrofonu na hodně rušné ulici.

# 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:**

Pro demonstraci metody Monte Carlo jsem se rozhodnul odhadnout hodnotu čísla . K odhadu hodnoty používám generování náhodných bodů uvnitř jednotkového čtverce a poté určujeme, kolik z těchto bodů spadá do jednotkového kruhu.

**1. Definice počtu náhodných bodů**

*num\_samples = 1000000* určuje počet generovaných náhodných bodů. Čím více bodů, tím přesnější bude odhad hodnoty .

**2. Generování náhodných bodů**

a jsou pole náhodných čísel v rozmezí , která představují souřadnice bodů v jednotkovém čtverci.

**3. Určení bodů uvnitř jednotkového kruhu**

*inside\_circle = x\*\*2 + y\*\*2 <= 1* kontroluje, zda body leží uvnitř kruhu o poloměru 1 (jednotkový kruh).

**4. Výpočet přibližné hodnoty**

pi\_estimate = 4 \* np.sum(inside\_circle) / num\_samples vypočítává odhad hodnoty na základě poměru počtu bodů uvnitř kruhu k celkovému počtu bodů, vynásobeného 4 (protože plocha čtvrtkruhu je ).

# 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:**

Pro výpočet numerické derivace s adaptabilním krokem používám metodu dvojitého kroku. Funkce bude iterovat s krokem , poté s krokem a vypočítá numerickou derivaci pomocí centrální diference. Pokud je rozdíl mezi výsledky menší než zvolená tolerance, výpočet končí. Pokud není, funkce se opakuje s krokem a novým středovým bodem.

Dále mám funkci pro výpočet numerické derivace s pevným krokem. Použijeme opět centrální diferenci s krokem , kde je pevně zvolené číslo.

Zde je porovnání jednotlivých výsledků pro :

Analytická derivace: 0.7071067811865476

Numerická derivace s pevným krokem: 0.7059288589999413

Numerická derivace s adaptabilním krokem: 0.7071056302971712

Vidíme, že všechny tři výsledky jsou podobné. Numerická derivace s adaptabilním krokem je o něco přesnější než numerická derivace s pevným krokem, ale obě jsou velmi blízko analytické derivaci.