Matematický software

Zápočtový dokument

Jméno: Adam Budiš

Kontaktní email: budis.adam@seznam.cz

Datum odevzdání: 21.9.2023

Odkaz na repozitář: https://github.com/adambudis/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:
Pro vyřešení této úlohy jsem zvolil provést měření pro výpočet určitého integrálu, skalárního součinu, násobení matic, součtu matic a násobení matice konstantou.

Při měření výpočtu určitého integrálu jsem zvolil složitou funkci a porovnal ji s knihovnou numpy a scipy. Měření ukázalo, že numpy je nejrychlejší knihovna a že výpočet klasickým pythonem trval nejdéle.

Při řešení ostatních úloh jsem porovnával pouze klasický python s knihovnou numpy a zvolil jsem mnohem menší čísla pro měření. Volba mnohem menších čísel pro měření způsobila to, že klasický python byl rychlejší než modul numpy, který je vhodný pro práci s mnohem větším množstvím dat.

1. Určitý integrál

```
def vypocet_klasickym_pythonem(a, b, dx):
    start = time.perf_counter()
    integral = 0
    x = a
    while x < b:
        integral += dx * (f(x) + f(x+dx))/2
        x += dx

    end = time.perf_counter()
    return end - start
```

```
def vypocet_pomoci_numpy(a, b, dx):
    start = time.perf_counter()
    x = np.arange(a, b+dx, dx)
    np.trapz(f(x), x, dx)

    end = time.perf_counter()
    return end - start
```

```python
def vypocet_pomoci_scipy(a, b, dx):
    start = time.perf_counter()
    x = np.arange(a, b+dx, dx)
    scipy.integrate.trapezoid(f(x), x)

    end = time.perf_counter()
    return end - start
```

Výpočet klasickým pythonem: 2.99057680s
Výpočet pomocí knihovny Scipy: 0.51495910s
Výpočet pomocí knihovny Numpy: 0.25675430s

2. Skalární součin

```python
def skalarni_soucin_klasickym_pythonem(a, b):
    start = time.perf_counter()
    skalarni_soucin = 0
    for i in range(len(a)):
        skalarni_soucin += a[i] * b[i]

    end = time.perf_counter()
    #print(skalarni_soucin)
    return end - start
```

```python
def skalarni_soucin_pomoci_numpy(a, b):
    # Přetypování vektorů z objektu třídy list na objekt třídy numpy.ndarray
    a = np.array(a)
    b = np.array(b)

    start = time.perf_counter()
    skalarni_soucin = np.dot(a, b)

    end = time.perf_counter()
    #print(skalarni_soucin)
    return end - start
```

Výpočet klasickým pythonem trval: 0.00000310s
Výpočet pomocí knihovny Numpy trval: 0.00001790s

3. Násobení matic

```python
def nasobeni_matic_klasickym_pythonem(A, B):
    start = time.perf_counter()

    # Rozměry matic
    matice_A_radky = len(A)
    matice_A_sloupce = len(A[0])
    matice_B_radky = len(B)
    matice_B_sloupce = len(B[0])

    # Inicializace výsledné matice c
    matice_C = [[0 for _ in range(matice_B_sloupce)] for _ in range(matice_A_radky)]

    # Násobení matic
    for i in range(matice_A_radky):
        for j in range(matice_B_sloupce):
            for k in range(matice_A_sloupce):
                matice_C[i][j] += A[i][k] * B[k][j]

    end = time.perf_counter()
    #print(f"Výsledek v pythonu: {matice_C}")
    return end - start
```

```python
def nasobeni_matic_pomoci_numpy(A, B):
    # Přetypování matice z objektu třídy list na objekt třídy numpy.ndarray
    A = np.array(A)
    B = np.array(B)

    start = time.perf_counter()
    matice_C = np.matmul(A, B)
    end = time.perf_counter()
    #print(f"Výsledek v numpy: {matice_C}")
    return end - start
```

Výpočet klasickým pythonem trval: 0.00003300s
Výpočet pomocí knihovny Numpy trval: 0.00006190s

4. Součet matic

```python
def soucet_matic_klasickym_pythonem(A, B):
    start = time.perf_counter()

    matice_A_radky = len(A)
    matice_A_sloupce = len(A[0])

    # Inicializace výsledné matice C
    matice_C = [[0 for _ in range(matice_A_sloupce)] for _ in range(matice_A_radky)]

    # Součet matic
    for i in range(matice_A_radky):
        for j in range(matice_A_sloupce):
            matice_C[i][j] = A[i][j] + B[i][j]

    end = time.perf_counter()
    #print(f"Výsledek v pythonu: {matice_C}")
    return end - start
```

```python
def soucet_matic_pomoci_numpy(A, B):
    # Přetypování matic z objektu třídy list na objekt třídy numpy.ndarray
    A = np.array(A)
    B = np.array(B)

    start = time.perf_counter()
    matice_C = np.add(A, B)
    end = time.perf_counter()
    #print(f"Výsledek v numpy: {matice_C}")
    return end - start
```

Výpočet klasickým pythonem trval: 0.00001140s
Výpočet pomocí knihovny Numpy trval: 0.00002450s

5. Násobení matic číslem

```python
def nasobeni_matice_cislem_pythonem(A, c):
    start = time.perf_counter()

    matice_A_radky = len(A)
    matice_A_sloupce = len(A[0])

    # Inicializace výsledné matice C
    matice_C = [[0 for _ in range(matice_A_sloupce)] for _ in range(matice_A_radky)]

    # Násobení matice číslem
    for i in range(matice_A_radky):
        for j in range(matice_A_sloupce):
            matice_C[i][j] = A[i][j] * c

    end = time.perf_counter()
    #print(f"Výsledek v pythonu: {matice_C}")
    return end - start
```

```python
def nasobeni_matice_cislem_pomoci_numpy(A, c):
    # Přetypování matice z objektu třídy list na objekt třídy numpy.ndarray
    A = np.array(A)

    start = time.perf_counter()
    matice_C = A * c;
    end = time.perf_counter()
    #print(f"Výsledek v numpy: {matice_C}")
    return end - start
```

Výpočet klasickým pythonem trval: 0.00001970s
Výpočet pomocí knihovny Numpy trval: 0.00002200s

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:
Jako iterační metodu jsem zvolil Jacobiho iterační metodu, se kterou jsem byl seznámen na hodinách.

```
# funkce pro výpočet matice pomocí Jacobiho iterační metody
def jacobiho_iteracni_metoda(A, b, niteraci):
    x = np.ones(len(A))
    D = np.diag(A)
    L = np.tril(A, k = -1)
    U = np.triu(A, k = 1)
    for i in range(niteraci):
        x = (b - np.matmul((L + U), x)) / D
        #print("iterace:",i, "x=",x)
    return x
```

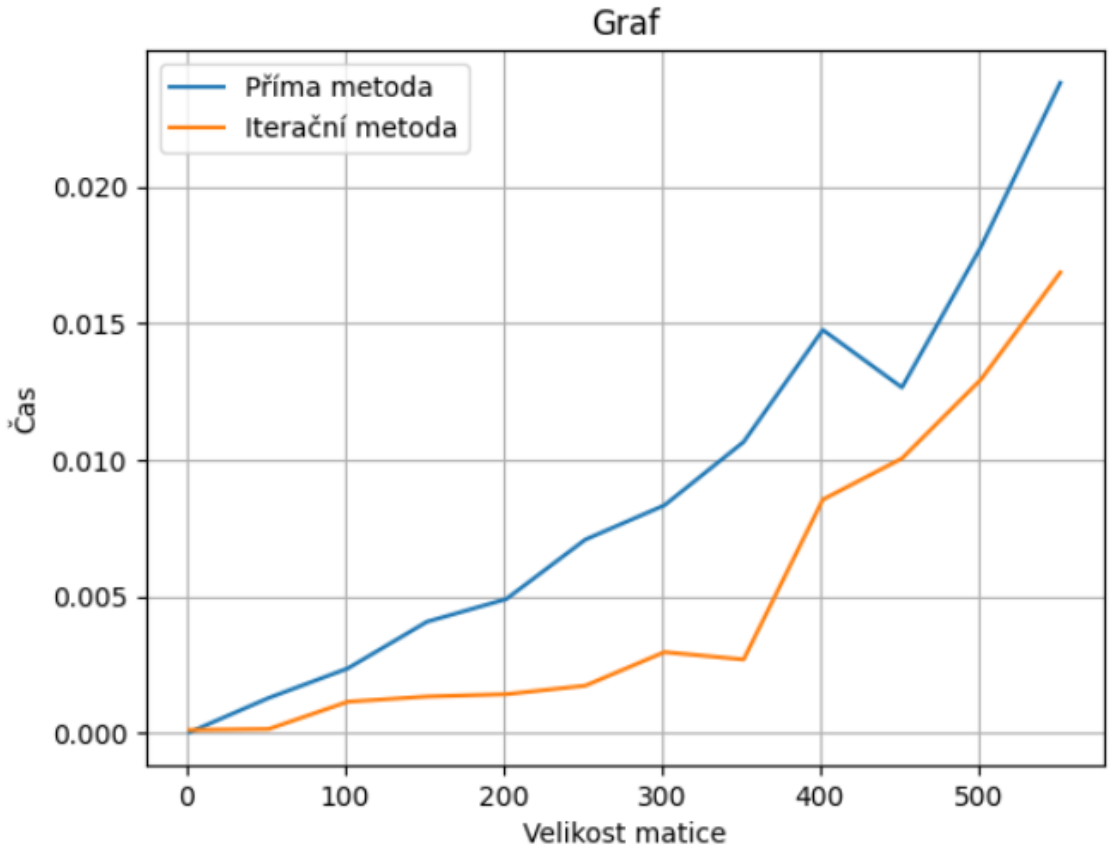
Následně jsem si vytvořil funkci pro generování vhodných náhodných lineárních soustav které budou mít řešení a budou konvergovat.

```
# funkce pro vygenerování matice
def vygeneruj_nahodnou_matici(rozmer):
    A = np.random.rand(rozmer, rozmer)
    np.fill_diagonal(A, np.sum(np.abs(A), axis=1))
    # zajištění že matice není singulární a správně konverguje
    while not np.linalg.matrix_rank(A) == rozmer:
        A = np.random.rand(rozmer, rozmer)
        np.fill_diagonal(A, np.sum(np.abs(A), axis=1))

    b = np.random.rand(rozmer)
    return [A, b]
```

Nakonec jsem vytvořil cyklus, ve kterém spočítám průměrný čas k nalezení řešení soustavy rovnic při různém počtu neznámých.

```
for velikost_matice in range(MIN_VELIKOST, MAX_VELIKOST, 50):
    velikosti.append(velikost_matice)

    casy_prime_metody = []
    casy_iteracni_metody = []

    for i in range(POCET):
        # vyenerování soustavy rovnic
        A, b = vygeneruj_nahodnou_matici(velikost_matice)

        # řešení přímou metodou
        start = time.perf_counter()
        np.linalg.solve(A, b)
        end = time.perf_counter()
        casy_prime_metody.append(end - start)

        # řešení iteracni metodou
        start = time.perf_counter()
        jacobiho_iteracni_metoda(A, b, N_ITERACI)
        end = time.perf_counter()
        casy_iteracni_metody.append(end - start)

    prumerne_casy_prime_metody.append(sum(casy_prime_metody) / POCET)
    prumerne_casy_iteracni_metody.append(sum(casy_iteracni_metody) / POCET)
```

Výsledný graf



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:
K vytvoření vlastního generátoru semínka jsem zvolil snímání pohybu myši. Ke snímání pohybu myši používám Listener, který po určitou dobu při každém pohybu kurzorem zavolá funkci, která přidá aktuální pozici myši do seznamu. Pro vytvoření semínka následně dojde k vypočtení hash hodnoty z nasbíraných dat.

```
# Doba po kterou probíhá snímání pohybu myše
DOBA_SNIMANI = 5

pozice_myse = []

# Funkce která se zavolá při každém pohybu myší a uloží pozici myši
def on_move(x, y):
    pozice_myse.append((x, y))

def generator():
    start = time.time()
    with Listener(on_move=on_move) as listener:
        while time.time() - start < DOBA_SNIMANI:
            pass
            #print("Probíhá snímání myše.")

    return hashlib.sha256(str(pozice_myse).encode()).digest()
```

Příklad vygenerovaného semínka a vygenerování náhodných čísel

```
# Vygenerování semínka z pozic myše a výpis
seminko = generator()
print(f"Vygenerované semínko z pohybu myší: {seminko}")

# Generování náhodných čísel ze semínka
random.seed(seminko)
print(random.randint(0, 100))
print(random.randint(0, 100))
```

Vygenerované semínko z pohybu myší:
b'T7\xe2*\x9e\x9f\xde?\x05E\x9bU\xb9\x84\xb7\xb9p\xe2\x984\x10\xa1\xc1\x80\xc5\xdd\x08\x17
\xaek\x9d\x16'
97
64

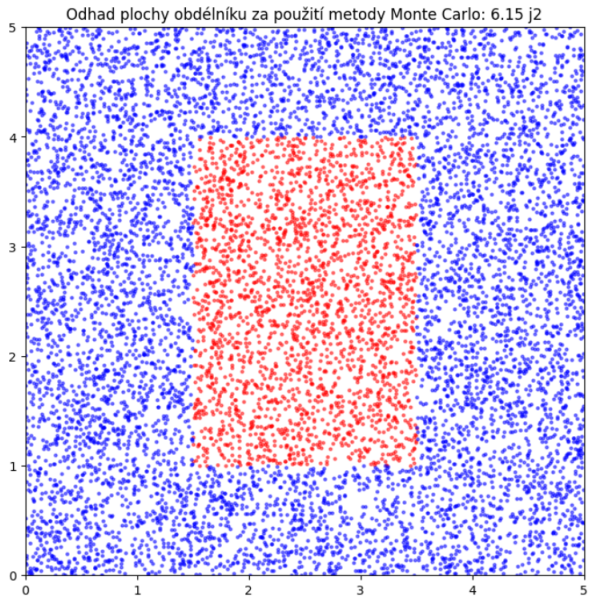
7. Metoda Monte Carlo

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 příklad využití metody Monte Carlo jsem si připravil odhad plochy obdélníku ve čtverci. Nejprve si zvolím souřadnice vrcholů obdélníku a rozměr čtverce. Následně v cyklu generuji náhodný bod v rámci čtverce o daném rozměru. Poté kontroluji zda vygenerovaný bod leží uvnitř obdélníku. Nakonec vypočítám odhad plochy pomocí poměru počtu bodů uvnitř obdélníku k celkovému počtu bodů a následně vynásobím plochou čtverce.

```
# Seznam vrcholů, které tvoří obdélník
VRCHOLY = [(1.5, 1), (1.5, 4), (3.5, 4), (3.5, 1)]
POCET_POKUSU = 10000
ROZMER_CTVERCE = 5;

zasahy = []
minuti = []

for _ in range(POCET_POKUSU):
    novy_x, novy_y = np.random.uniform(0, ROZMER_CTVERCE), np.random.uniform(0, ROZMER_CTVERCE)

    # Podmínka, která kontroluje jestli vygenerovaný bod leží v obdélníku
    if min(VRCHOLY, key=lambda x: x[0])[0] <= novy_x <= max(VRCHOLY, key=lambda x: x[0])[0] and min(VRCHOLY, key=lambda x: x[1])[
        zasahy.append((novy_x, novy_y))
    else:
        minuti.append((novy_x, novy_y))

# Výpočet odhadu plochy čtverce
odhad_plochy = (len(zasahy) / POCET_POKUSU) * ROZMER_CTVERCE**2
```

Výsledný graf



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:
Při řešení této úlohy jsem si nejprve zvolil integrační meze a krok. Následně jsem vytvořil jednotlivé funkce na integrování a funkci na výpis výsledku integrací.

```
# vytvoření integračních mezí a kroku
a = 0
b = 3
dx = 0.1

x = np.arange(a, b+dx, dx)

# integrace pomocí funkce trapezoid
def integrate_trapezoid(y, x):
    return scipy.integrate.trapezoid(y, x)

# integrace pomocí funkce simpson
def integrate_simpson(y, x):
    return scipy.integrate.simpson(y, x)

# integrace pomocí funkce romberg
def integrate_romberg(f, a, b):
    return scipy.integrate.romberg(f, a, b)

# funkce na vypis integraci
def vypis_vysledky_integraci(trapezoid, simpson, romberg):
    print(f"Výsledek integrálu pomocí funkce trapezoid: {trapezoid}")
    print(f"Výsledek integrálu pomocí funkce simpson: {simpson}")
    print(f"Výsledek integrálu pomocí funkce romberg: {romberg}")
```

Řešení polynomické funkce

```
# Analytické řešení polynomické funkce
# I(-3x2 + 4x + 5)(a, b) = [-x3 + 2x2 + 5x](0,3) = -27 + 18 + 15 = 6

# definice polynomické funkce
def f(x):
    return -3*x**2 + 4*x + 5

# výpis výsledků
vypis_vysledky_integraci(integrate_trapezoid(f(x), x), integrate_simpson(f(x), x), integrate_romberg(f, a, b))
```

```
Výsledek integrálu pomocí funkce trapezoid: 5.985
Výsledek integrálu pomocí funkce simpson: 6.0000000000000036
Výsledek integrálu pomocí funkce romberg: 6.0
```

Řešení harmonické funkce

```
# Analytické řešení harmonické funkce
# I(2sinx)(a, b) = [-2cosx](0, 3) = 2 - 2cos(3) = 3.97

# definice harmonické funkce
def f(x):
    return 2*np.sin(x)

# výpis výsledků
vypis_vysledky_integraci(integrate_trapezoid(f(x), x), integrate_simpson(f(x), x), integrate_romberg(f, a, b))
```

```
Výsledek integrálu pomocí funkce trapezoid: 3.9766677861325515
Výsledek integrálu pomocí funkce simpson: 3.979987206938697
Výsledek integrálu pomocí funkce romberg: 3.9799849932023488
```

Řešení exponenciální funkce

```
# Analytické řešení exponenciální funkce
# I(2^x)(a, b) = [2^x / log(2)](0, 3) =  10.099

# definice exponenciální funkce
def f(x):
    return 2**x

# výpis výsledků
vypis_vysledky_integraci(integrate_trapezoid(f(x), x), integrate_simpson(f(x), x), integrate_romberg(f, a, b))
```

```
Výsledek integrálu pomocí funkce trapezoid: 10.102908321039077
Výsledek integrálu pomocí funkce simpson: 10.098866580578232
Výsledek integrálu pomocí funkce romberg: 10.098865286222773
```