KIV/PRO

2. Semestrální práce – počítání determinantu

16.10.2023 Michael Hladký

Obsah

1. Zadání

Prostudujte pro zvolený problém existující metody řešení. Vyberte jednu z nich nebo navrhněte vlastní, implementujte a ověřte na experimentech. Postup a výsledky popište ve zprávě.

Volba problému a metody: Můžete buď použít článek zpracovaný v práci 1 anebo si vybrat kterýkoliv z problémů popisovaných v kapitolách 13-18 knihy [1] s výjimkou podkapitol 13.3, 13.6, 13.10, 14.1 až 14.5, 15.3, 15.4, z kapitoly 18 nejsou povolena témata týkající se textů a znakových řetězců, ale témata týkající se množin ano. Důvodem těchto omezení je nasměrovat vás přednostně k tématům, která se neprobírají ani na PRO ani na PT.

2. Úvod

2.1 Popis problematiky

Jako úkol na řešení jsem si vybral z knížky [1] úkol popsaný v podkapitole 13.4 o výpočtu determinantu.

Na vstupu dostaneme čtvercovou matici M řádu n. Jaký je determinant |M| matice M?

2.2 Vysvětlení pojmů

2.2.1 Determinant

Definice: Determinant je číslo, které vypočteme sečtením všech možných permutací na množině {1, 2, 3, …, n–1, n}, kde n je počet řádků a sloupců (musí se rovnat) v matici.

$$\det(A) = \sum_{\pi} zn(\pi) a_{1\pi(1)} a_{2\pi(2)} a_{3\pi(3)} \dots a_{n\pi(n)}$$

Také lze definovat vzorcem:

$$\det(A) = \sum_{i=1}^{n} (-1)^{sign(\pi_i)} \prod_{j=1}^{n} A[j, \pi_i]$$

Operátor zn () / sign () vynásobí danou část vzorce 1 nebo -1 podle toho, jestli je permutace sudá nebo lichá (1 pokud je sudá, - 1 pokud je lichá).

Determinant leze vypočítat pouze pro čtvercovou matici (počet sloupců se rovná počtu řádků). Determinant lze například použít k zjištění, jestli je matice singulární, či ne.

Jedno z hlavních použití determinantu je zjištění, jestli je matice singulární nebo regulární. Pokud se determinant matice se rovná 0, je singulární, jestli ne tak je regulární.

2.2.2 Permutace

Definice: Permutace je vzájemně jednoznačné zobrazení konečné množiny na sebe.

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

$$1 \rightarrow 2$$

$$2 \rightarrow 3$$

$$3 \rightarrow 1$$

Permutaci lze také definovat jako uspořádanou n-tici prvků z množiny M o n prvcích, tak, že se v ní žádný prvek neopakuje. Počet různých permutací na množinu M je n!.

2.2.3 Algebraický doplněk

Definice: Máme čtvercovou matici A řádu n. Pak číslo $A_{ij} = (-1)^{i+j} \det(A[i/j])$ se nazývá algebraický doplněk prcku a_{ij}. A [i / j] znamená matice A bez i – tého řádku a j – tého sloupce.

Algebraický doplněk se používá při rozvoji podle i – tého řádku.

Např. pro matici řádu n = 2:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} * A_{11} + \ a_{11} * A_{11}$$

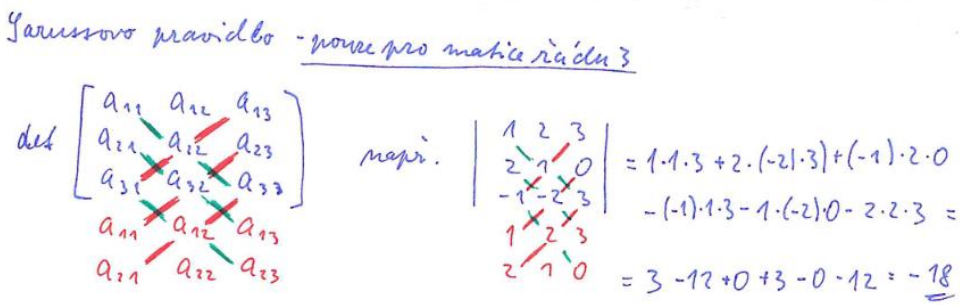
2.2.4 Rozvoj podle i – tého řádku

Věta: Máme matici řádu n a i ϵ {1, 2, …, n}. Pak platí:

$$\det(A) = \sum_{j=1}^{n} a_{ij}A_{ij} = a_{i1}A_{i1} + a_{i2}A_{i2} + \cdots + a_{in}A_{in}$$

2.2.5 Sarussovo pravidlo

Sarussovo pravidlo je způsob počítání determinantu matice. Sečte všechny permutace v čtvercové matici. Platí pouze pro matice řádu n = 3.



Obrázek 1 – použití Sarussova pravidla na matici 3x3 [2]

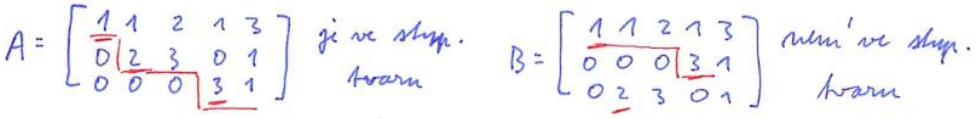
2.2.6 Čtvercová matice

Čtvercová matice je matice, kde se počet řádků a sloupců rovnají. Čtvercová matice typu n x n se nazývá matice řádu n.

2.2.7 Stupňovitý tvar matice

Definice: Máme matici A typu m/n. Pivot řádku i je první nenulový prvek v tomto řádku (z leva). Matice A je ve stupňovitém tvaru, jestliže pro každý řádek v matici platí:

- je-li pivot i – tého řádku na pozici j, tak pivoty ve následujících řádcích jsou na pozicích větších než j
- je-li i – tý řádek nulový, každý následující řádek také nulový



Obrázek 2 – Stupňovitý tvar matice [2]

2.2.8 Trojúhelníkový tvar matice

Matice, kde všechny prvky pod hlavní diagonálou jsou nulové se nazývá horní trojúhelníková matice.

Matice, kde všechny prvky nad hlavní diagonálou jsou nulové se nazývá dolní trojúhelníková matice.

Determinant čtvercové matice ve trojúhelníkovém tvaru je roven násobku členů na hlavní diagonále matice.

2.2.9 Gaussova eliminační metoda

Gaussova eliminační metoda je metoda většinou používaná k řešení soustav lineárních rovnic. Jedná se o způsob, jak převést matici do stupňovitého tvaru. Řádky matice nejdříve seřadíme podle pozice jejich pivotů vzestupně (od nejnižší pozice pivotů po nejvyšší). Metoda se provede tak, že vezmeme první řádek a podle pivotu v tomto řádku násobek daného řádku přičítáme, či odečítáme k ostatním řádkům v matici tak, aby pod pivotem byli nulové prvky. Toto opakujeme na další následující řádky, dokud matice není v stupňovitém tvaru.

$$\begin{pmatrix} 1 & 2 & -1 \\ 2 & 5 & 0 \\ 3 & 7 & -1 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

3. Způsob počítání determinantu a zvolené řešení

3.1 Počítání přes rozvoj podle i – tého řádku

Pokud determinant počítá člověk, tak většinou bude nejlepší použít rozklad matice podle rozvoje podle i – tého řádku. Člověk by většinou nejdřív v jednom řádku pomocí jednoho z prvků v tomto řádku odečetl násobky toho řádku od ostatních řádků, aby v řádce zbyl pouze jeden nenulový prvek a podle tohoto řádku by matici rozložil. Toto se dělá, dokud se matice nerozloží do tvaru, kde má počet řádků/sloupců 2 nebo 3 při použití Sarussova pravidla.

Pro počítání determinantu pro stroj je tento způsob také možný. Tento algoritmus by šel implementovat pomocí rekurze, kde program redukuje matici na menší matice do doby, než tyto matice mají počet řádků/sloupců 2 nebo 3 při implementaci Sarussova

pravidla. Spočítá determinant těchto menších matic a jednotlivé determinanty těchto menších matic sečte. Počítač ani nemusí danou matici upravovat do tvaru, kde je v matici v řádku pouze jeden nenulový prvek.

Avšak tento algoritmus pro výpočet determinantu má velmi vysokou výpočetní složitost pro matice většího řádu než 3 a to konkrétně $O(n! + n^2)$, což lze vyjádřit jako $O(n!)$. Tato vysoká výpočetní složitost zajistí že tento algoritmus nelze použít na matice s vysokým počtem řádků/sloupců, kvůli příliš dlouhému výpočtu. Pro matice řádu 1 až 3 má algoritmus konstantní složitost $O(1)$. Výpočetní složitost metody pro redukci matice je $O(n^2)$.

3.1.1 Pseudo kód

```
FUNCTION vypoctiDetrminat(matice: 2D array of floats) -> float
    velikost = length of matice
    navrat = 0

    IF velikost == 1 THEN
        navrat = matice[0][0]
    ELSE IF velikost == 2 THEN
        navrat = matice[0][0] * matice[1][1] - matice[0][1] * matice[1][0]
    ELSE IF velikost == 3 THEN
        FOR i = 0 TO 2
            navrat += matice[0][i] * matice[1][(i + 1) % 3] * matice[2][(i
+ 2) % 3]
        END FOR
        FOR i = 0 TO 2
            navrat -= matice[2][i] * matice[1][(i + 1) % 3] * matice[0][(i
+ 2) % 3]
        END FOR
    ELSE
        FOR i = 0 TO velikost
            mensiMatice = redukceMatice(matice, i)
            znamenko = zjistiZnam(i)
            navrat = navrat + znamenko * matice[0][i] *
vypoctiDetRek(mensiMatice)
```

```
        END FOR
    END IF

    RETURN navrat
END FUNCTION
```

3.2 Počítání přes horní trojúhelníkovou matici

Lepší způsob, který jsem zvolil pro výpočet determinantu pro počítač je přes Gaussovu eliminační metodu. Gaussovou eliminační metodou si převede matici do horního trojúhelníkového tvaru. Po převedení matice do tohoto tvaru vynásobíme prvky na diagonále matice a dostaneme její determinant. Tento algoritmus má mnohem menší výpočetní složitost $O(n^3)$ na rozdíl od O(n!) minulého algoritmu.

Chod algoritmu by se dal popsat tak, že nejdříve mu předáme matici, u které chceme vypočítat determinant. Dále algoritmus matici převede na trojúhelníkový tvar. Toto udělá tak, že nedříve vezme první prvek v první řádce (pivot) a pro každý prvek pod ním vypočítá kolikrát je nutné ho vynásobit/vydělit (koeficient), aby při odečtení prvku v sloupci, kde je pivot s násobkem pivotu horního řádku s koeficientem vyšla nula. Po odečtení všech prvků pod pivotem se algoritmus převede na další řádek a sloupce v matici, kde toho provede znova. Toto opakuje, dokud matice není v horním trojúhelníkovém tvaru. Po převedení matice vynásobí jednotlivé prvky na diagonále a vrátí vypočtenou hodnotu jako determinant.

Složitost algoritmu $O(n^3)$ vychází z výpočetní složitosti Gaussovy eliminační metody, konkrétně $O(n^3)$. Výpočetní složitost vynásobení diagonály je O(n), takže algoritmus má složitost $O(n^3 + n)$. Jelikož n^3 roste asymptoticky rychleji než n, tak můžeme říct, že složitost algoritmu je $O(n^3)$. Jeden menší problém této metody je, že kvůli způsobu, jak se vyhodnocují reálná čísla v mnoha programovacích jazycích, tak se může stát, že výsledek se bude lišit od správného výsledku ve velmi malých řádech (například v řádu deseti-tisícin, ne-li i sta-tisícin).

3.2.1 Pseudo kód

```
FUNCTION prevedDoTrojuhelnik(matice: 2D array of floats) -> 2D array of
floats:
    navrat = matice
```

```
    FOR i = 0 TO length of matice - 1:
        FOR j = i + 1 TO length of matice - 1:
            nasobitelOdecitatele = navrat[j][i] / navrat[i][i]
            FOR k = i TO length of matice - 1:
                      navrat[j][k] = navrat[j][k] - nasobitelOdecitatele *
      navrat[i][k]
            END FOR
        END FOR
    END FOR

    RETURN navrat
END FUNCTION

FUNCTION vysabDiag(matice: 2D array of floats) -> float:
    navrat = 1

    FOR i = 0 TO length of matice - 1:
        navrat = navrat * matice[i][i]

    RETURN navrat
END FUNCTION

FUNCTION vypoctiDet(matice: 2D array of floats) -> float:
    maticeTroj = prevedDoTrojuhelnik(matice)
    det = vysabDiag(maticeTroj)
    RETURN det
END FUNCTION
```

4. Experiment

V experimentu jsem porovnával mnou zvolené řešení problému (použitím Gaussovi eliminační metody), mnou implementovanou metodou přes rozvoj podle i – tého řádku a implantaci z volně dostupné knihovny do jazyka Java JAMA, která obsahuje funkce pro výpočet determinantu matice. [3]

4.1 Výsledky experimentu

V experimentu jsme používali náhodně generované matice, které byly naplněny náhodně generovanými hodnotami reálných čísel v rozsahu -100 až 100. Čas byl měřen v ns. Výsledná hodnota byla získána jako průměr z více vzorků.

Řád matice	Mé řešení	rozvoj podle i – tého řádku	JAMA
1	47081,82	21654,55	387945,50
2	47690,82	21054,55	379081,70
3	46518,18	22409,09	419872,70
5	46436,36	410709,10	44218,18
9	51763,64	15608455,00	371200,00
10	60163,55	105186973,00	421627,10
11	53236,36	950478309,10	379200,00
12	60972,73	10802818727,00	411854,50

Tabulka 1 – výsledky měření pro menší matice

Řád matice	Mé řešení	JAMA
15	216145,55	474272,64
20	241963,55	550763,82
25	278554,55	496563,64
50	1127036,36	1225554,55
90	3380281,82	2804572,73
100	4886336,36	3010572,73
180	6864681,82	6198800,00
250	11156536,36	13729818,18
500	32586872,73	48100372,73
1000	172974836,36	306608463,64
1500	569007500,00	1012305954,55

Tabulka 2 – výsledky měření pro větší matice

Z první tabulky lze vidět, že ze začátku je nejrychlejší metoda výpočtu požívající rozvoj podle i – tého řádku, ale jakmile začneme počítat determinanty matic pro matice řádu větší než pět, tak doba výpočtu pro metodu s rozvojem podle i – tého řádku začne být prudce delší než ostatní způsoby. U matic s řádem vyšším než 12 přestala být tato metoda měřitelná.

Dále můžeme vidět, že mé řešení může být v určitých situacích více efektivní než algoritmus ze knihovny JAMA. JAMA je více efektivní pro matice s řádem okolo 100–150.

Z případů, které jsem testoval v experimentu, všechny algoritmy vypočítaly téměř stejné hodnoty, s rozdíly až ve velmi malých číslech.

5. Operace s programem

Po spuštění se program zeptá, jestli si uživatel přeje matici vygenerovat, nebo zda si ji přeje zadat ručně. V každém případě se program zeptá na řád matice. Jestli uživatel zadal manuální zadání, program se bude ptát na prvky na jednotlivých místech v matici. Jestli uživatel zadá vygenerování matice, tak se program zeptá, v jakém intervalu reálných čísel mají být čísla generována.

Po zadání nebo vygenerování matice se vypočítá determinant nejdříve přes Gaussovu eliminační metodu, pak přes metodu v knihovně JAMA, a nakonec přes rozklad matice použitím rozkladu podle i – tého řádku.

6. Závěr

Implementoval jsem algoritmus pro výpočet determinantu čtvercové matice přes Gaussovu eliminační metodu a rozvoj podle i – tého řádku. Porovnal jsem tyto dva algoritmy mezi sebou a algoritmem použitým v knihovně JAMA. Z výsledků experimentu lze vidět, že algoritmus přes rozvoj podle i – tého řádku je silně neefektivní a Gaussova eliminační metoda je pro větší matice vhodnější. Oproti algoritmu v knihovně JAMA je mnou implementovaný algoritmus v některých maticích rychlejší, ale pro matice v řádech mezi 90–180 je algoritmus ze knihovny JAMA rychlejší.

Zdroje

[1] S. Skiena: The Algorithm Design Manual, Spring-Verlag, New York Berlin Heidelberg, 1998 nebo 2008 2.vydání lze najít např. na: http://mimoza.marmara.edu.tr/~msakalli/cse706_12/SkienaTheAlgorithmDesignManual.pdf

[2] Přednášky z lineární algebry docenta Přemysla Holuba. (2022).

[3] Knihovna JAMA dostupná z**:** https://math.nist.gov/javanumerics/jama/