Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Výuková aplikace pro lineární algebru

Bakalářská práce

2020 Vítek Rais

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Vítek Rais**
Osobní číslo:	**I17131**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Výuková aplikace pro lineární algebru**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské prace je vytvořit aplikaci pro generování příkladů z vybraných oblastí lineární algebry. Aplikace bude schopna animovat postup řešení příkladů. Aplikace bude vytvořena v programovacím jazyce Java.

V teoretické části práce budou představena vybraná témata a typy úloh z lineární algebry. Dále bude proveden návrh výukové aplikace.

V praktické části bude vytvořena aplikace pro podporu výuky lineární algebry. Aplikace umožní generování úloh z vybraných oblastí lineární algebry. Složitost úloh bude definována podle vstupních parametrů, aplikace umožní generování různých druhů řešení pro specifické typy úloh. Aplikace dále umožní pro zadané příklady vypočítat řešení a animovat elementární kroky výpočtu simulující postup lidského řešitele.

Rozsah pracovní zprávy: **40 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

BEČVÁŘ, Jindřich. *Lineární algebra*. Vydání páté. Praha: Matfyzpress, 2019. ISBN 978-80-7378-378-5.

ŠIMSOVÁ, Jana. *Sbírka úloh z matematiky: lineární algebra*. 2. vyd. Ústí nad Labem: Univerzita J.E. Purkyně v Ústí nad Labem, 2009. ISBN 978-80-7414-184-3.

Vedoucí bakalářské práce: **Ing. Marie Nedvědová**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **15. listopadu 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 17. prosince 2019

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 19. 8. 2020

Vítek Rais

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Marii Nedvědové za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnovala.

ANOTACE

Bakalářská práce se zabývá tvorbou výukové aplikace pro vybrané oblasti lineární algebry v jazyce Java. V první části této práce jsou představena vybraná témata a typy úloh z lineární algebry. Také jsou popsány možnosti řešení těchto úloh. V další části práce je proveden návrh výukové aplikace. Aplikace umožňuje generování úloh z vybraných oblastí lineární algebry, které jsou vhodné pro procvičování. Tyto úlohy jsou následně vypočteny a je možné zobrazit jednotlivé kroky výpočtu simulující postup lidského řešitele.

KLÍČOVÁ SLOVA

výuková aplikace, lineární algebra, matice, Java, soustava lineárních rovnic, determinant

TITLE

Educational application for linear algebra

ANNOTATION

This bachelor thesis deals with the creation of an educational application in Java for selected areas of linear algebra. The first part of this work summarizes selected topics and types of exercises from linear algebra. There are also described possible solutions of these exercises. The second part of the thesis explains design of the application. The application enables to generate exercises from the selected areas of linear algebra which are suitable for practice. These exercises are calculated by the application and its user can view particular steps of the solution, simulating calculation procedure of a human solver.

KEYWORDS

educational application, linear algebra, matrix, Java, system of linear equations, determinant

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

IT – informační technologie

SE – Standard Edition

JRE – Java SE Runtime Edition

JVM – Java Virtual Machine

JDK – Java SE Development Kit

AWT – Abstract Window Toolkit

EJML – Efficient Java Matrix Library

ÚVOD

Lineární algebra je jednou ze základních disciplín matematiky, která se věnuje vektorům, vektorovým prostorům, soustavám lineárních rovnic a lineárním transformacím, a je nedílnou součástí mnoha vědních oborů. V IT nalezne široké uplatnění například v počítačové grafice či algoritmizaci.

Autor si zvolil dané téma, protože se o oblast lineární algebry zajímá, především pak o její uplatnění při řešení algoritmických problémů. Autor je také aktivní při doučování studentů, a výstupy této práce budou při této činnosti velmi užitečné.

Cílem této práce je vytvořit výukovou aplikaci pro vybrané oblasti lineární algebry. Jedná se o desktopovou aplikaci vytvořenou v jazyce Java, která umožňuje generovat příklady různé složitosti, a pro tyto příklady vypočítat řešení a animovat elementární kroky výpočtu tak, aby byl simulován postup lidského řešitele. Aplikace klade velký důraz na to, aby byly vygenerované příklady vhodné k procvičování. To znamená, že výsledky příkladů jsou ve většině případech celočíselné a v průběhu výpočtu není obvykle potřeba pracovat s velkými ani desetinnými čísly.

Teoretická část se zaměřuje na představení vybraných témat z lineární algebry, která jsou obsažena ve výsledné aplikaci. Jsou zde popsány nejen potřebné teoretické základy, ale i jednotlivé typy úloh a možné postupy jejich řešení.

Součástí teoretické části je také představení návrhu výsledné aplikace. Tato část se zaměřuje na popis použitých knihoven, datových struktur a také logiky všech hlavních částí aplikace. Dále je představeno ovládání a nastavení důležitých komponent výsledné aplikace, jako je generátor příkladů nebo modul pro simulaci výpočtu lidským řešitelem.

Tato bakalářská práce využívá metodu analýzy pro určení důležitých oblastí lineární algebry, které jsou následně popisovány. Další metodou, kterou tato práce využívá, je metoda komparace pro vybrání nejvhodnějšího způsobu řešení daných problémů, které je následně implementováno v aplikaci.

1 ZÁKLADNÍ POJMY LINEÁRNÍ ALGEBRY

V této kapitole budou představeny základní stavební kameny lineární algebry. Bez znalosti a porozumění těchto základů je velice obtížné, ne-li přímo nemožné, správně pochopit pojmy a postupy řešení v dalších kapitolách.

1.1 Matice

Nechť X je neprázdná množina prvků nebo čísel a zároveň m, n jsou přirozená čísla. Maticí typu $m \times n$ nad množinou X potom rozumíme obdélníkové schéma

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix},$$

kde $a_{ij} \in X$ pro každé $i = 1, \ldots, n$ a každé $j = 1, \ldots, m$. Takovou matici potom značíme $(a_{ij})_{m \times n}$ nebo jednodušeji (a_{ij}). O prvcích matice potom říkáme, že každý prvek a_{ij} stojí na místě *ij*. [2], [5]

Hlavní diagonála je označení pro posloupnost prvků matice $a_{11}, a_{22}, \ldots, a_{nn}$.

Vedlejší diagonála označuje posloupnost prvků $a_{n1}, a_{n-1,2}, \ldots, a_{1n}$.

1.1.1 Základní typy matic

Pokud se $m = n$, tedy matice obsahuje stejný počet řádků a sloupců, tak ji označujeme jako čtvercovou matici řádu n. V opačném případě používáme termín obdélníková matice. Příklad čtvercové matice (vlevo) a obdélníkové matice (vpravo):

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Nulová matice je taková matice, ve které každý prvek $a_{ij} = 0$. Příklad nulové matice:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Jednotková matice je čtvercová matice, ve které jsou všechny prvky hlavní diagonály rovné 1 a všechny ostatní prvky jsou nulové neboli pro každé $i, j = 1, \ldots, n,\ i \neq j$ je $a_{ij} = 0$ a $a\_ii = 1$. Obvykle se jednotková matice označuje písmenem E nebo I, resp. E_n nebo I_n, kde n je rozměr matice. Příklad jednotkové matice I_3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Diagonální (též skalární) matice je čtvercová matice, která obsahuje nenulové prvky pouze na hlavní diagonále neboli pro každé $i,j = 1,\dots,n,\ i \neq j$ je $a_{ij} = 0$ a $a_{ii} = c \in R$. Z této podmínky nevyplývá, jaké prvky mají být na hlavní diagonále, takže na ní mohou být i nuly. Příklady diagonálních matic:

$$\begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Horní trojúhelníková matice je taková matice, jejíž prvky pod hlavní diagonálou jsou nulové, neboli pro každé $i,j = 1,\dots,n,\ i > j$ je $a_{ij} = 0$. Příklad horní trojúhelníkové matice:

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

Dolní trojúhelníková matice je definována velmi podobně jako horní trojúhelníková matice. Rozdíl je v tom, že dolní trojúhelníková matice obsahuje nulové prvky nad hlavní diagonálou. Matematický zápis je následující: pro každé $ij = 1,\dots,n,\ i < j$ je $a_{ij} = 0$. Příklad horní trojúhelníkové matice:

$$\begin{pmatrix} 7 & 0 & 0 \\ 1 & 2 & 0 \\ 8 & 5 & 1 \end{pmatrix}$$

Schodovitá matice je taková matice, ve které každý řádek začíná větším počtem nul než řádek předchozí. Pokud matice obsahuje nulové řádky a ty jsou umístěné až pod všemi nenulovými, pak je i tato matice považována za schodovitou. Zvláštními typy této matice jsou i nulová, jednotková, diagonální a horní trojúhelníková matice. Taktéž libovolná matice obsahující pouze jeden řádek je schodovitá. Příklady schodovitých matic:

$$\begin{pmatrix} 3 & 7 & 6 & 1 \\ 0 & 8 & 7 & 2 \\ 0 & 0 & 0 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 0 \\ 0 & 7 & 3 \\ 0 & 0 & 5 \end{pmatrix}$$

Transponovaná matice k matici A (značí se A^T) je taková matice typu $m \times n$, pro kterou platí $A^T = (b_{ji})$, kde pro každé $i = 1,\dots,n,\ j = 1,\dots,m$ je $b_{ji} = a_{ij}$. To znamená, že transponovaná matice k původní matici vznikne „vzájemným prohozením“ řádků a sloupců,

které mají stejný index (prohozením prvního řádku a prvního sloupce, druhého řádku a druhého sloupce, …). Příklad transpozice matice A:

$$A = \begin{pmatrix} 3 & 2 & 1 \\ 7 & 8 & 4 \\ 6 & 7 & 1 \\ 1 & 2 & 5 \end{pmatrix}, \qquad A^T = \begin{pmatrix} 3 & 7 & 6 & 1 \\ 2 & 8 & 7 & 2 \\ 1 & 4 & 1 & 5 \end{pmatrix}$$

Symetrická matice je čtvercová matice, jejíž transponovaná matice je totožná s původní maticí: $A = A^T$. Matematicky zapsáno: pro každé $i, j = 1, \dots, n$ je $a_{ij} = a_{ji}$. Důsledkem je symetrie matice podle hlavní diagonály. Příklad symetrické matice:

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 3 \\ 0 & 3 & 5 \end{pmatrix}$$

Antisymetrická matice je taková matice, pro níž platí $A = -A^T$, neboli pro každé $i, j = 1, \dots, n$ je $a_{ij} = -a_{ji}$. Pro splnění této podmínky je nezbytné, aby všechny prvky na hlavní diagonále byly nulové. [1], [3], [5]. Příklad antisymetrické matice:

$$\begin{pmatrix} 0 & 2 & -4 \\ -2 & 0 & 1 \\ 4 & -1 & 0 \end{pmatrix}$$

1.2 Aritmetický vektor

Aritmetickým vektorem (zkráceně vektorem) rozumíme uspořádanou n-tici reálných čísel $(a_1, a_2, \dots, a_n)$. Čísla $a_i, i \in \{1, \dots, n\}$ nazýváme souřadnice vektoru. Číslo n označuje rozměr neboli dimenzi vektoru. Vektor, který má dimenzi 1, se označuje jako skalár. Alternativně lze vektor definovat jako matici, jejíž počet řádků nebo sloupců je roven jedné. Vektory se obvykle zapisují do sloupce a pro označení vektoru se používají malá písmena abecedy se šipkou nad písmenem.

$$\vec{x} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

Vektor lze psát i v řádku, operace převedení vektoru z řádkového na sloupcový (i naopak) se nazývá transpozice.

$$\vec{x} = (a_1, \dots, a_n)^T$$

V dalších částech této práce budou vektory zapisované v řádkové podobě bez znaků šipky nad písmenem označujícím název vektoru a bez znaménka transpozice, tedy např. $x = (1, 2, 3)$.

Vektor libovolné dimenze, jehož všechny prvky se rovnají 0, se nazývá nulový vektor. [2], [8]

1.3 Grupa

Libovolná množina G, na které je definována binární operace „$\otimes$“ se nazývá grupa, jestliže pro tuto operaci platí:

a. asociativní zákon – pro každé $a, b, c \in G$ platí $(a \otimes b) \otimes c = a \otimes (b \otimes c)$,
b. existence neutrálního nebo jednotkového prvku – existuje $e \in G$ takové, že pro každé $a \in G$ platí $a \otimes e = e \otimes a = a$,
c. existence inverzního prvku – pro každé $a \in G$ existuje $x \in G$, pro které platí $a \otimes x = x \otimes a = e$,
d. komutativní zákon – pro každé $a, b \in G$ platí $a \otimes b = b \otimes a$.

Poslední pravidlo není povinné, pokud platí, tak hovoříme o komutativní grupě. Na počest norského matematika Nielse Abela se komutativní grupa někdy též označuje jako Abelova grupa.

Příklady: Množina R s operací sčítání, množina $R \setminus \{0\}$ s operací násobení, dvouprvková množina $\{1, -1\}$. [1], [9]

1.4 Těleso

Libovolná alespoň dvouprvková množina T se dvěma binárními operacemi „+“, „$\cdot$“ se nazývá těleso, pokud splňuje následující podmínky:

a. asociativita sčítání – pro libovolné prvky $a, b, c \in T$ platí $(a + b) + c = a + (b + c)$,
b. komutativita sčítání – pro libovolné prvky $a, b \in T$ platí $a + b = b + a$,
c. existence nulového prvku – existuje prvek $0 \in T$ takový, že pro každé $a \in T$ platí $a + 0 = a$,
d. existence opačného prvku – pro každé $a \in T$ existuje $-a \in T$, pro které platí $a + (-a) = 0$,
e. asociativita násobení – pro libovolné prvky $a, b, c \in T$ platí $(a \cdot b) \cdot c = a \cdot (b \cdot c)$,
f. existence jednotkového prvku – existuje prvek $1 \in T$ a platí $a \cdot 1 = a$,
g. existence inverzního prvku – pro každé $a \in T, a \neq 0$ existuje $a^{-1} \in T$, pro které platí $a \cdot a^{-1} = 1$,
h. distributivita – pro libovolné prvky $a, b, c \in T$ platí $a \cdot (b + c) = a \cdot b + b \cdot c$,

i. netrivialita – nulový prvek ≠ jednotkový prvek,
j. komutativita násobení – pro libovolné prvky $a, b \in T$ platí $a \cdot b = b \cdot a$.

Poslední podmínka není povinná, pokud je splněna, nazýváme těleso komutativní. [1], [4]

Můžeme si všimnout, že některé podmínky jsou velice podobné podmínkám definujícím grupu. Díky nim je možné definici tělesa značně zkrátit. Množina T se dvěma binárními operacemi „+“, „$\cdot$“ se nazývá těleso, jestliže

a. T s operací „+“ je komutativní grupa,
b. $T \setminus \{0\}$ s operací „$\cdot$“ je grupa,
c. Operace „+“ a „$\cdot$“ splňují distributivní zákon – pro libovolné prvky $a, b, c \in T$ platí $a \cdot (b + c) = a \cdot b + b \cdot c$.

Pokud je $T \setminus \{0\}$ s operací „$\cdot$“ komutativní grupa, pak hovoříme o komutativním tělese.

Příklady: Množiny všech racionálních, celých i komplexních čísel se standardním sčítáním i násobením jsou komutativní tělesa. Množiny přirozených ani celých čísel se standardním sčítáním i násobením nejsou tělesa. [4]

1.5 Vektorový prostor

Nechť T je těleso a V je množina s binární operací sčítání, které označíme symbolem „+“. Musí existovat zobrazení kartézského součinu $T \times V$ do množiny V. Pro dvojici prvků $a \in T$ a $v \in V$ přiřazuje toto zobrazení prvek, který značíme $a \cdot v$ nebo av. O tomto přiřazení hovoříme jako o násobení prvků množiny V prvky množiny T a značíme ho symbolem „$\cdot$“. Prvky tělesa T budeme označovat jako skaláry a prvky množiny V jako vektory. Dále musí platit následující podmínky:

a. Asociativní zákon sčítání vektorů – pro libovolné prvky $u, v, w \in V$ platí $(u + v) + w = u + (v + w)$,
b. komutativní zákon sčítání vektorů – pro libovolné prvky $u, v \in V$ platí $u + v = v + u$,
c. existence nulového vektoru – existuje prvek $o, u \in V$ a platí $u + o = u$,
d. existence opačného vektoru – pro každé $u \in V$ existuje $-u \in V$, pro které platí $u + (-u) = o$,
e. distributivita násobení vektorů skalárem – pro libovolné prvky $u, v \in V$ a $a \in T$ platí $a \cdot (u + v) = a \cdot u + a \cdot c$),

f. distributivita násobení skalárů vektorem – pro libovolné prvky $u \in V$ a $a, b \in T$ platí $(a + b) \cdot u = a \cdot u + b \cdot u$,
g. asociativita násobení vektoru skaláry – pro libovolné prvky $u \in V$ a $a, b \in T$ platí $(a + b) \cdot u = a \cdot (b \cdot u)$,
h. existence jednotkového vektoru – existuje prvek $1, u \in V$ platí $1 \cdot u = u$.

Při splnění všech těchto podmínek říkáme, že V je vektorový prostor nad tělesem T. Skaláry obvykle značíme písmeny ze začátku a vektory z konce abecedy. Násobení vektorů skaláry vždy zapisujeme tak, aby byly skaláry vlevo a vektory vpravo. [1]

2 VYBRANÉ KAPITOLY LINEÁRNÍ ALGEBRY

2.1 Gaussova metoda řešení soustav lineárních rovnic

S některými případy soustav lineárních rovnic a jejich řešením se pravděpodobně každý z nás setkal již na střední škole. Gaussova eliminační metoda je jednou z nejpoužívanějších metod pro řešení obecných soustav lineárních rovnic.

2.1.1 Definice soustavy lineárních rovnic

Soustavou n lineárních rovnic, která má m neznámých nad tělesem T, se rozumí soustava lineárních rovnic ve tvaru

$$\begin{matrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = y_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = y_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = y_n \end{matrix},$$

kde $a_{ij} \in T$ nazýváme koeficienty a $y_1, y_2, \ldots, y_n \in T$ nazýváme absolutními členy. Řešením takovéto soustavy je uspořádaná n-tice $(x_1, x_2, \ldots, x_{n)}$ prvků tělesa T, pro které platí všechny výše uvedené vztahy.

Z této soustavy rovnic lze vytvořit matici soustavy, resp. rozšířenou matici soustavy:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \quad resp. \quad \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1m} & y_1 \\ a_{21} & a_{22} & \cdots & a_{2m} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} & y_n \end{array}\right)$$

Svislá čára oddělující sloupec absolutních členů v rozšířené matici není povinná, ale pro lepší orientaci se obvykle uvádí. Matici soustavy označujeme A a rozšířenou matici $A|b$. [1], [5]

2.1.2 Homogenní soustavy lineárních rovnic

Soustava lineárních rovnic

$$\begin{matrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 0 \\ \vdots \\ a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n = 0 \end{matrix}$$

kde $a_{ij} \in T$, se označuje jako homogenní soustava k lineárních rovnic o n neznámých nad T. Soustava lineárních rovnic se tedy nazývá homogenní, pokud je vektor pravých stran roven nulovému vektoru. V ostatních případech se soustava nazývá nehomogenní. [5]

2.1.3 Ekvivalentní úpravy soustavy rovnic

Je dána soustava lineárních rovnic nad tělesem T. Následující operace jsou ekvivalentními úpravami této soustavy:

a. libovolná změna pořadí rovnic,
b. vynásobení kterékoliv rovnice nenulovým číslem z T,
c. přičtení jakéhokoli násobku rovnice k jiné rovnici (násobek je z T).

Při provádění ekvivalentních úprav není nutné pracovat s celou soustavou rovnic, mnohem výhodnější je pracovat pouze s rozšířenou maticí soustavy. Při práci s rozšířenou maticí soustavy se místo rovnic pracuje se řádky matice. Takové operace se potom nazývají řádkové elementární úpravy. Výše uvedené úpravy jsou poté ještě doplněny o vypuštění nulového řádku matice.[5]

2.1.4 Gaussův eliminační algoritmus

1) Je dána soustava rovnic.
2) Tuto soustavu převedeme do podoby rozšířené matice soustavy.
3) Postupně pomocí řádkových elementárních úprav vytvoříme matici ve schodovitém tvaru.

Můžeme si všimnout, že pomocí tohoto algoritmu získáme matici ve schodovitém tvaru, ale řešení soustavy rovnic nezískáme. To musíme dopočítat jiným způsobem. Než začneme počítat řešení soustavy, je ovšem vhodné ověřit její řešitelnost. To se provádí pomocí Frobeniovy věty popsané v následující kapitole (2.1.5). Nejběžnějšími způsoby výpočtu řešení jsou Gauss-Jordanova eliminace, které se budeme věnovat v podkapitole 2.1.6, a postupné dosazení proměnných.

Metoda postupného dosazení je velice jednoduchá, pokud existuje jen jedno řešení soustavy. V případě, že je řešení nekonečně mnoho, určíme některé proměnné jako parametry, se kterými dále pracujeme jako s čísly. Počet parametrů se rovná počtu proměnných dané soustavy rovnic mínus počtu řádků schodovité matice. Parametr je libovolné číslo $\in T$. [5]

2.1.5 Hodnost matice a Frobeniova věta

Frobeniova věta je velmi známá a důležitá věta, která se týká řešitelnosti soustav lineárních rovnic. Pro její vysvětlení je ovšem důležité chápat pojem hodnost matice.

Hodnost matice je počet lineárně nezávislých řádků nebo sloupců matice. Lze dokázat, že počet lineárně nezávislých řádků i sloupců bude stejný (Důkaz viz [1]). Prozatím ovšem nevíme, co

to je lineární nezávislost. Tomuto tématu se budeme věnovat později (viz podkapitola 2.2). Pro určení hodnosti matice nám postačí vědět, že je-li matice ve schodovitém tvaru, pak hodnost matice je rovna počtu jejích nenulových řádků. Hodnost matice A budeme označovat jako $r(A)$ nebo $rank(A)$. [1], [5]

Frobeniova věta říká, že soustava lineárních rovnic (nad T) je řešitelná právě tehdy, když je hodnost matice této soustavy rovnic rovna hodnosti rozšířené matice téže soustavy, neboli musí platit rovnost $rank(A) = rank(A|b)$. [5], [8]

Můžeme si všimnout, že Frobeniova věta nám neříká nic o počtu řešení dané soustavy rovnic. Pro zjednodušení se však tato problematika k Frobeniově větě připojuje.

Soustava má právě jedno řešení, pokud hodnost matice soustavy je rovna počtu proměnných soustavy. Soustava má nekonečně mnoho řešení, pokud hodnost matice soustavy je menší než počet proměnných této soustavy. [7]

Pokud je soustava rovnic homogenní (vektor pravých stran je roven nulovému vektoru), je vždy řešitelná. Existuje-li právě jedno řešení, pak je jím n-tice $x_1 = 0, x_2 = 0, \dots, x_n = 0$. Toto řešení nazýváme triviální. U homogenních soustav tedy existuje buď pouze triviální řešení, nebo mají nekonečně mnoho řešení [5]

2.1.6 Gauss-Jordanova eliminace

Gauss-Jordanova eliminace je rozšíření Gaussova eliminačního algoritmu, převádí rozšířenou matici do tzv. redukovaného řádkově odstupňovaného tvaru. Matice je v redukovaném řádkově odstupňovaném tvaru, jestliže:

a. je ve schodovitém tvaru,
b. každý pivot (první nenulový prvek v řádku) se rovná 1,
c. všechny prvky nad každým pivotem (prvky ve stejném sloupci) se rovnají 0.

Taková matice vypadá takto:

$$\left(\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & e_1 \\ 0 & 1 & \cdots & 0 & e_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & e_n \end{array}\right)$$

Z takové matice již rovnou vyčteme řešení ($x_i = e_i$ pro $i = 1,2, \dots, n$). V případě, že má matice nekonečně mnoho řešení, tak proměnné, které jsou parametry, nalezneme ve sloupcích neobsahujících žádného pivota. [6]

Nyní si názorně ukážeme výpočet soustavy rovnic pomocí Gaussovy eliminační metody a dokončení řešení jak pomocí postupného dosazení proměnných, tak i pomocí Gauss-Jordanovy eliminace.

2.1.7 Příklady:

Příklad 1: Řešte následující soustavu rovnic (nad R):

$$\begin{aligned} x + 2y + 3z &= 2 \\ 2x - 2y + z &= 0 \\ 3x - 6y - z &= 5 \end{aligned}$$

Řešení: Přepíšeme soustavu rovnic do rozšířené matice, a pomocí řádkových elementárních úprav dostaneme matici do schodovitého tvaru.

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 2 & -2 & 1 & 0 \\ 3 & -6 & 1 & 5 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 0 & -6 & -5 & -4 \\ 3 & -6 & 1 & 5 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 0 & -6 & -5 & -4 \\ 0 & -12 & -10 & -1 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 2 \\ 0 & -6 & -5 & -4 \\ 0 & 0 & 0 & 7 \end{array}\right)$$

Nyní si můžeme všimnout, že na posledním řádku upravené matice jsou všechny koeficienty nulové a absolutní člen se nerovná nule. To znamená, že hodnost matice soustavy se nerovná hodnosti matice rozšířené soustavy neboli $rank(A) \neq rank(A|b)$. To podle Frobeniovy věty znamená, že daná soustava rovnic nemá řešení.

Příklad 2: Řešte následující soustavu rovnic (nad R):

$$\begin{aligned} x - 2y + z &= 0 \\ 2x + 2y - 3z &= -3 \\ -3x + 3y + 2z &= 9 \end{aligned}$$

Řešíme stejně jako předchozí příklad:

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 2 & 2 & -3 & -3 \\ -3 & 3 & 2 & 9 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 6 & -5 & -3 \\ -3 & 3 & 2 & 9 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 6 & -5 & -3 \\ 0 & -3 & -5 & 9 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 6 & -5 & -3 \\ 0 & 0 & 5 & 15 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 6 & -5 & -3 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Matice je ve schodovitém tvaru a vidíme, že $rank(A) = rank(A|b) = n$, kde n je počet proměnných dané soustavy. To podle Frobeniovy věty znamená, že soustava je řešitelná a má

právě jedno řešení. Máme několik možností, jak toto řešení získat. Buď použijeme postupné dosazení proměnných, nebo využijeme principu Gauss-Jordanovy eliminace a budeme pokračovat v řádkových elementárních úpravách.

Dokončení řešení postupným dosazením proměnných:

$$\begin{matrix} x-2y+z=0 \\ 6y-5z=-3 \\ z=3 \end{matrix} \sim \begin{matrix} x-2y=-3 \\ 6y=12 \\ z=3 \end{matrix} \sim \begin{matrix} x=1x=1 \\ y=2y=2 \\ z=3z=3 \end{matrix}$$

Dokončení řešení pomocí Gauss-Jordanovy eliminační metody:

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 6 & 0 & 12 \\ 0 & 0 & 1 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -2 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Tato soustava má tedy jediné řešení, a to $(1, 2, 3)$.

Příklad 3: Řešte následující soustavu rovnic (nad R):

$$\begin{matrix} 2x+3y-3z=2 \\ -x-2y+3z=1 \\ x+3z=7 \end{matrix}$$

Řešení:

$$\left(\begin{array}{ccc|c} 2 & 3 & -3 & 2 \\ -1 & -2 & 3 & 1 \\ 1 & 0 & 3 & 7 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 3 & -3 & 2 \\ 0 & -1 & 3 & 4 \\ 1 & 0 & 3 & 7 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 3 & -3 & 2 \\ 0 & -1 & 3 & 4 \\ 0 & -3 & 9 & 12 \end{array}\right) \sim$$

$$\left(\begin{array}{ccc|c} 2 & 3 & -3 & 2 \\ 0 & -1 & 3 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 3 & -3 & 2 \\ 0 & -1 & 3 & 4 \end{array}\right)$$

Matice je ve schodovitém tvaru a vidíme, že $rank(A) = rank(A|b) < n$, kde n je počet proměnných dané soustavy. To podle Frobeniovy věty znamená, že soustava je řešitelná a má nekonečně mnoho řešení.

Dokončení řešení pomocí dosazení proměnných:

Určíme proměnnou z jako parametr $t \in R$. Tento parametr postupně dosadíme do ostatních rovnic a dokončíme řešení

$$\begin{matrix} 2x+3y-3z=2 \\ -1y+3z=4 \\ z=t \end{matrix} \sim \begin{matrix} 2x+3y-3t=2 \\ -1y+3t=4 \\ z=t \end{matrix} \sim \begin{matrix} 2x+3y=2+3t \\ y=-4+3t \\ z=t \end{matrix} \sim$$

$$\begin{matrix} 2x - 12 + 9t = 2 + 3t \\ y = -4 + 3t \\ z = t \end{matrix} \quad \sim \quad \begin{matrix} x = 7 - 3t \\ y = -4 + 3t \\ z = t \end{matrix}$$

Dokončení řešení pomocí Gauss-Jordanovy eliminace:

$$\left(\begin{matrix} 2 & 3 & -3 \\ 0 & -1 & 3 \end{matrix}\middle|\begin{matrix} 2 \\ 4 \end{matrix}\right) \sim \left(\begin{matrix} 2 & 3 & -3 \\ 0 & 1 & -3 \end{matrix}\middle|\begin{matrix} 2 \\ -4 \end{matrix}\right) \sim \left(\begin{matrix} 2 & 0 & 6 \\ 0 & 1 & -3 \end{matrix}\middle|\begin{matrix} 14 \\ -4 \end{matrix}\right) \sim \left(\begin{matrix} 1 & 0 & 3 \\ 0 & 1 & -3 \end{matrix}\middle|\begin{matrix} 7 \\ -4 \end{matrix}\right)$$

Třetí sloupec upravené matice neobsahuje pivot, proto je určen za parametr, takže můžeme rovnou vyčíst řešení $z = t,\ y = -4 + 3t, z = 7 - 3t$, kde $t \in R$. Nezapomeneme, že v matici jsou parametry na levé straně, proto se u nich při přepisu do řešení se změní znaménko.

2.2 Lineární kombinace, závislost a nezávislost vektorů

Pojem lineární závislosti a nezávislosti vektorů z vektorového prostoru má v lineární algebře zásadní důležitost, jelikož je využit v celé řadě dalších definic a vět. Při definici těchto pojmů se využívá termínu lineární kombinace vektorů, který si objasníme v následující kapitole.

2.2.1 Lineární kombinace vektorů

Nechť V je vektorový prostor nad tělesem T, $v_1, v_2, \dots, v_k$ jsou vektory prostoru V a $a_1, a_2, \dots, a_k$ jsou prvky tělesa prostoru T. Lineární kombinací vektorů $v_1, v_2, \dots, v_k$ potom rozumíme vektor

$$a_1 v_1 + a_2 v_2 + \cdots + a_k v_k.$$

Alternativně lze tento vektor zapsat pomocí sumy:

$$\sum_{i=1}^{k} a_i v_i$$

Prvky $a_1, a_2, \dots, a_k$ potom nazýváme koeficienty. Jestliže se počet vektorů rovná nule (neboli $k = 0$), potom takovou kombinaci nazýváme prázdná. Prázdná kombinace je rovna nulovému vektoru. Jestliže kombinace není prázdná ($k \geq 1$), potom v případě, že jsou všechny koeficienty nulové ($a_1 = a_2 = \cdots = a_k = 0$), hovoříme o triviální lineární kombinaci (takový vektor je opět nulový). V opačném případě, tedy když alespoň jeden z koeficientů $a_1, a_2, \dots, a_k$ je nenulový, tak takovou kombinaci nazýváme netriviální. [1], [9]

2.2.2 Lineární závislost a nezávislost

Nechť V je vektorový prostor nad tělesem T a $v_1, v_2, \dots, v_k$ je množina vektorů prostoru V. Množina vektorů $v_1, v_2, \dots, v_k$ je lineárně závislá, pokud je možno některý vektor této skupiny

vyjádřit jako lineární kombinaci ostatních vektorů. Alternativně lze říci, že množina vektorů je lineárně závislá, pokud existuje netriviální lineární kombinace těchto vektorů $(v_1, v_2, \dots, v_k)$, která je rovna nulovému vektoru. Podle definic lineární kombinace můžeme říct, že množina vektorů $v_1, v_2, \dots, v_k$ je lineárně závislá, jestliže existují reálná čísla $\alpha_1, \alpha_2, \dots, \alpha_k$ taková, že alespoň jedno z nich je nenulové, a zároveň platí rovnice

$$a_1 v_1 + a_2 v_2 + \cdots + a_k v_k = o.$$

Množinu vektorů nazýváme lineárně nezávislou, pokud není lineárně závislá. [1], [9]

Obvykle se místo formulace množina vektorů $v_1, v_2, \dots, v_k$ je lineárně závislá či nezávislá používá kratší formulace, a to vektory $v_1, v_2, \dots, v_k$ jsou závislé či nezávislé. Při používání této formulace je potřeba si uvědomit, co je myšleno. Rozhodně se tím nemyslí, že jednotlivé vektory jsou lineárně závislé/nezávislé (např. v_1 je lineárně nezávislý, v_2 je lineárně závislý atd.), ale vždy je myšlena vlastnost celé množiny vektorů. [9]

Nyní si uvedeme několik faktů týkajících se této problematiky. Důkazy následujících tvrzení lze snadno odvodit z předešlých definic. Prázdná množina je lineárně nezávislá. Jestliže množina vektorů obsahuje nulový vektor, je lineárně závislá. Jednoprvková množina $\{v\}$, je lineárně nezávislá, pokud je v nenulový vektor. Jestliže je některý vektor množiny $v_1, v_2, \dots, v_k$ násobkem jiného vektoru, pak je množina lineárně závislá. Neplatí to ale naopak. I množina $v_1, v_2, \dots, v_k$ taková, že žádný vektor není násobkem jiného, může být lineárně závislá. Každá podmnožina lineárně nezávislé množiny je lineárně nezávislá. Každá „nadmnožina“ lineárně závislé množiny je lineárně závislá. Nadmnožinou je myšlena množina obsahující všechny vektory původní množiny a libovolný počet dalších vektorů ze stejného vektorového prostoru. [1], [8]

2.2.3 Příklady

Příklad 1: Určete, zda je vektor $u = (5, 2, -4)$ lineární kombinací vektorů $v = (2, 0, -1)$, $w = (-1, -4, 8)$ a $x = (1, 4, 1)$.

Řešení: Pokud je vektor u lineární kombinací vektorů v, w a x, pak musí platit rovnice

$$a_1 v + a_2 w + a_3 x = u$$

Dosazením do této rovnice získáme rovnici ve tvaru

$$a_1(2, 0, -1) + a_2(-1, -4, 8) + a_3(1, 4, 1) = (5, 2, -4)$$

Tuto rovnici můžeme pomocí násobení vektorů skalárem a sčítání vektorů upravit do následující podoby:

$$(2a_1 - a_2 + a_3, -4a_2 + 4a_3, -1a_1 + 8a_2 + a_3) = (5, 2, -4)$$

Dva vektory se rovnají, pokud se rovnají jejich složky. V tom případě je možné rovnici přepsat do této soustavy rovnic:

$$\begin{aligned} 2a_1 - a_2 + a_3 &= 5 \\ -4a_2 + 4a_3 &= 2 \\ -a_1 + 8a_2 + a_3 &= -4 \end{aligned}$$

Nyní vyřešíme tuto soustavu rovnic např. pomocí Gaussovy eliminační metody (viz podkapitola 2.1). Tato soustava rovnic má pouze jedno řešení, a to $a_1 = 4$, $a_2 = -9$, $a_1 = 1$. To, že je soustava řešitelná, znamená, že vektor u je lineární kombinací vektorů $v, w,$ a x, konkrétně [9]

$$u = 4v - 9w + x \text{ resp. } (5, 2, -4) = 4(2, 0, -1) - 9(-1, -4, 8) + (1, 4, 1).$$

Příklad 2: Určete, zda je vektor $u = (0, 5, 2)$ lineární kombinací vektorů $v = (-4, 5, 0)$, $w = (-5, 1, 2)$ a $x = (1, 4, -2)$.

Řešení: Řešíme obdobně jako předchozí příklad. Pokud je vektor u lineární kombinací vektorů v, w a x, pak musí platit rovnice

$$a_1 v + a_2 w + a_3 x = u$$

Dosazením a následnými úpravami této rovnice dostaneme soustavu rovnic ve tvaru

$$\begin{aligned} -4a_1 - 5a_2 + a_3 &= 0 \\ 5a_1 + a_2 + 4a_3 &= 5 \\ 2a_2 - 2a_3 &= 2 \end{aligned}$$

Řešením této soustavy rovnic pomocí Gaussovy eliminační metody dostaneme rozšířenou matici soustavy ve tvaru

$$\left(\begin{array}{ccc|c} 4 & 5 & -1 & 2 \\ 0 & 21 & -21 & -4 \\ 0 & 0 & 0 & 41 \end{array}\right)$$

která podle Frobeniovy věty (viz podkapitola 2.1.5) nemá řešení. To znamená, že vektor u není lineární kombinací vektorů v, w a x.

Příklad 3: Určete, zda jsou vektory $x = (2, 1, 2)$, $y = (-6, -3, 4)$ a $z = (-2, -1, 1)$ lineárně závislé.

Řešení: Z definice je zřejmé, že musí platit následující rovnice:

$$a_1 x + a_2 y + a_3 z = o$$

Provedením stejných úprav jako v předchozích příkladech dostaneme následující homogenní soustavu rovnic:

$$\begin{aligned} 2a_1 - 6a_2 - 2a_3 &= 0 \\ a_1 - 3a_2 - a_3 &= 0 \\ 2a_1 + 4a_2 + a_3 &= 0 \end{aligned}$$

Tuto soustavu vyřešíme např. pomocí Gaussovy eliminační metody. Řešení této soustavy je $a_1 = -t/2$, $a_2 = t$, $a_3 = 0$, kde $t \in R$. Byly nalezeny nenulové koeficienty úvodní rovnice, tudíž jsou vektory x, y, z lineárně závislé.

Výše uvedené řešení je správné, ale je možné si ho i výrazně zkrátit. Jelikož je daná soustava rovnic homogenní a my nepotřebujeme znát její kompletní řešení, vystačíme si i s pouhým vědomím, zda existují nějaká nenulová řešení této soustavy. Tyto podmínky jsou uvedeny v podkapitole 2.1.5. V průběhu řešení této soustavy pomocí Gaussovy eliminační metody do stavu

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 0 \\ 2 & 4 & 1 & 0 \end{array}\right)$$

Je zřejmé, že hodnost této matice je maximálně dvě, ale neznámé jsou tři. Z vlastností homogenních soustav rovnic a Frobeniovy věty tedy plyne, že daná soustava má i nenulová řešení, a můžeme tedy říct, že vektory x, y, z jsou lineárně závislé.

Příklad 4: Určete, zda jsou vektory $x = (-3, 1, 1)$, $y = (1, 2, 0)$ a $z = (-4, 2, 2)$ lineárně závislé.

Řešení: Opět musí platit rovnice

$$a_1 x + a_2 y + a_3 z = o,$$

ze které úpravami získáme následující soustavu rovnic:

$$\begin{aligned} -3a_1 + a_2 - 4a_3 &= 0 \\ a_1 + 2a_2 + 2a_3 &= 0 \\ a_1 + 2a_3 &= 0 \end{aligned}$$

Soustavu rovnic budeme řešit pomocí Gaussovy eliminační metody, dokud nedostaneme matici ve schodovitém tvaru.

$$\left(\begin{array}{ccc|c} 3 & -1 & 4 & 0 \\ 0 & 7 & 2 & 0 \\ 0 & 0 & 2 & 0 \end{array}\right)$$

Z této matice je zřejmé, že její hodnost je rovna počtu proměnných (obě hodnoty se rovnají 3). Můžeme tedy prohlásit, že daná soustava má pouze triviální řešení, takže neexistuje žádné nenulové řešení původní rovnice, a tudíž jsou vektory lineárně nezávislé.

2.3 Determinanty

Determinant je číslo, které určitým způsobem charakterizuje čtvercovou matici. Toto číslo je velmi významné, setkáme se s ním nejen v mnoha oblastech lineární algebry, ale i v dalších matematických disciplínách. Pro definici determinantu je však třeba rozumět permutacím. Proto se tomuto pojmu budeme věnovat v následující podkapitole. [9]

2.3.1 Permutace

Nechť M je libovolná konečná množina o n prvcích. Permutací množiny M rozumíme každé vzájemně jednoznačné zobrazení (bijekci) množiny M na množinu M.Jinými slovy permutací prvků množiny M rozumíme uspořádanou n-tici prvků množiny M, ve které se žádný prvek neopakuje. Celkový počet permutací na množině M, kde n je počet prvků množiny, je $n!$ [1], [9]

Při vyšetřování permutací není důležité, jak označíme prvky množiny M. Proto můžeme bez újmy na obecnosti předpokládat, že $M = \{1,2, \dots, n\}$, kde n je libovolné přirozené číslo. [1]

Permutaci P množiny M obvykle zapisujeme tabulkou, kdy do horního řádku napíšeme v libovolném pořadí prvky množiny M a pod každý prvek $m \in M$ napíšeme jeho obraz. Ten budeme značit $P(m)$. [1] Příklad permutace P pěti prvků:

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 5 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 1 & 4 & 3 & 2 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}$$

Permutace množiny M skládáme jako zobrazení, složením permutací vznikne opět permutace. Složení permutací P a Q značíme QP a každému číslu $i \in M$ přiřazujeme číslo $Q(P(i))$. Řekněme, že P a Q jsou permutace množiny $M = \{1, 2, 3, 4, 5\}$

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 5 & 4 & 2 \end{pmatrix} \text{ a } Q = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix}$$

$$QP = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 5 & 3 \end{pmatrix} \text{ a } PQ = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 3 & 2 & 4 \end{pmatrix}$$

Z příkladu je patrné, že skládání permutací není komutativní. Přesto lze nalézt permutace P a Q množiny M takové, že $PQ = QP$. Takové permutace potom označujeme jako záměnné či komutující. [1], [8]

Speciálním typem permutace je identická permutace. Pro tuto permutaci platí $P(m) = m$ pro každé $m \in M$. Příklad identické permutace na $M = \{1, 2, 3, 4\}$:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Ke každé permutaci množiny M existuje inverzní permutace. Ta se obvykle označuje P^{-1}. Složením původní a inverzní permutace získáme identickou permutaci. [1] Mějme

$$P = \begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix},$$

potom inverzní permutace k P je

$$P^{-1} = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ 1 & 2 & \cdots & n \end{pmatrix},$$

Nyní bych rád upozornil na velmi podobně nazvaný termín, kterým je inverze permutace. Tyto termíny neoznačují stejné skutečnosti, a proto je důležité je vzájemně nezaměňovat. Inverzí permutace P rozumíme každou dvouprvkovou množinu $\{i, j\}$, pro kterou platí $i < j$ a zároveň $P(i) > P(j)$. Permutaci P nazýváme sudou (lichou), pokud obsahuje sudý (lichý) počet inverzí. Identická permutace neobsahuje žádnou inverzi, a proto je sudá [1]

Celkový počet inverzí permutace snadno spočítáme z tabulkového zápisu tak, že pro každý prvek zkontrolujeme jeho obraz, jestliže je obraz větší než prvek, potom k celkovému počtu inverzí přičteme jejich rozdíl.

S inverzemi permutace úzce souvisí znaménko neboli signum permutace, které značíme $sgn\ P$. To je definováno rovností

$$sgn\ P = (-1)^{in\ P},$$

Kde $in\ P$ je počet všech inverzí permutace P. Z této rovnosti je patrné, že jestliže je permutace P sudá, pak $sgn\ P = 1$, jestliže je lichá, potom $sgn\ P = -1$.[1], [9]

Dalším specifickým typem permutace je transpozice. Permutace T množiny M se nazývá transpozice, jestliže existují indexy $i, j \in M$ a zároveň $i \neq j$ takové, že $T(i) = j$ a $T(j) = i$. Dále musí platit $T(k) = k$ pro všechna $k \in M \setminus \{i, j\}$. Tabulkový zápis této permutace je:

$$T = \begin{pmatrix} 1 & 2 & \cdots & i & \cdots & j & \cdots & n \\ 1 & 2 & \cdots & j & \cdots & i & \cdots & n \end{pmatrix},$$

jedná se tedy o identickou permutaci, ve které jsou pouze dva prvky navzájem prohozené.

Mějme permutaci P n prvkové množiny M. Permutaci P', kterou dostaneme vzájemným prohozením dvou prvků permutace P, nazýváme transpozicí permutace P. Provedením transpozice se změní znaménko permutace (důkaz viz [5])

$$P = \begin{pmatrix} 1 & 2 & \cdots & i & \cdots & j & \cdots & n \\ a_1 & a_2 & \cdots & a_i & \cdots & a_j & \cdots & a_n \end{pmatrix}, \quad P' = \begin{pmatrix} 1 & 2 & \cdots & i & \cdots & j & \cdots & n \\ a_1 & a_2 & \cdots & a_j & \cdots & a_i & \cdots & a_n \end{pmatrix}.$$

P' můžeme také symbolicky zapsat jako složení permutací TP, kde T je příslušná transpozice. Z tohoto zápisu je zřejmé, že libovolnou permutaci můžeme získat složením konečného počtu transpozic. [5], [10]

2.3.2 Definice a vlastnosti determinantů

Nechť A je čtvercová matice řádu n nad R. Determinant matice A značíme $det\, A$ a definujeme ho rovností

$$\det A = \sum_{P=(i_1, i_2, \ldots, i_n)} sgn\, P \cdot a_{1,i_1} a_{2,i_2} \cdots a_{n,i_n}.$$

Podle uvedeného vzorce se sčítá přes všechny permutace n prvků, je to tedy součet $n!$ součinů. Jako řád determinantu označujeme řád původní matice. Determinant také někdy značíme zápisem

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

Obzvláště ve spojení s výše uvedeným zápisem může slovo determinant nabývat i trochu jiný smysl. Může se jím označovat samotná matice, ze které je determinant počítán. V tomto smyslu potom hovoříme o prvcích, řádcích, sloupcích či hlavní a vedlejší diagonále. [1], [9]

Výpočet determinantu podle definice je zjevně složitý a zdlouhavý proces. Takový výpočet má smysl pro matice, jejichž řád není větší než 3. Již pro matice čtvrtého řádu dle definice existuje

4! neboli 24 sčítanců. Pokud toto číslo ještě někoho neodrazuje, pak můžeme uvést např. matice desátého řádu. Pro ně už by bylo potřeba spočítat a sečíst 10! neboli 3 628 800 desetičlenných součinů.

Níže je uveden výpočet determinantu pro matici prvního, resp. druhého řádu:

$$|a_{11}| = a_{11} \quad \text{resp.} \quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$$

Výpočet determinantu matice třetího řádu již není tak triviální, ale existuje pravidlo, podle kterého je jeho výpočet poměrně rychlý. Toto pravidlo se jmenuje Sarrusovo a budeme se mu věnovat v samostatné podkapitole. [5]

Nyní si uvedeme několik základních vlastností determinantů, které využijeme při výpočtech determinantů vyšších řádů:

a. determinant matice, která obsahuje nulový řádek, je roven nule,
b. determinant matice, která obsahuje dva shodné řádky, je roven nule,
c. determinant jakékoliv trojúhelníkové matice je roven součinu všech prvků na hlavní diagonále,
d. determinanty navzájem transponovaných matic se rovnají,
e. vynásobíme-li řádek matice A číslem $c \in R$, potom determinant vzniklé matice se rovná $c \cdot \det A$,
f. přičteme-li k řádku matice libovolný násobek jiného řádku téže matice, determinant se nezmění,
g. prohodíme-li v matici A libovolné dva řádky, potom determinant vzniklé matice je roven $-\det A$,
h. Jestliže je A matice řádu n a $c \in R \setminus \{0\}$, potom $\det(cA) = c^n \cdot \det A$.

U všech těchto vlastností lze bez změny vlastnosti uvažovat místo řádků sloupce. [1] Důkazy těchto tvrzení se nebudeme zabývat, ale jsou k dispozici v [1].

2.3.3 Výpočet determinantu s využitím Gaussovy eliminace

Vraťme se k vlastnostem determinantu z předchozí podkapitoly, především k vlastnosti c. Ta nám říká, že je velice jednoduché vypočítat determinant trojúhelníkové matice. My již ovšem známe metodu, díky které efektivně převedeme libovolnou čtvercovou matici na horní trojúhelníkovou (eventuelně schodovitou s nulovými řádky). Tato metoda se nazývá Gaussova eliminace (viz podkapitola 2.1.4).

Je ovšem důležité nezapomínat, že některé řádkové elementární úpravy mění hodnotu determinantu matice. Tyto změny je vhodné psát před upravenou matici, aby nebyly zapomenuty. Pro připomenutí uveďme změny determinantu, které nastávají při provádění řádkových elementárních úprav. Prohozením řádků se změní znaménko determinantu (vlastnost g). Vynásobením řádku nenulovým číslem α zvětší výsledný determinant α-krát (vlastnost e). Pro získání determinantu původní matice je potřeba výsledný determinant vydělit číslem α. Přičtení libovolného násobku řádku k řádku jinému determinant nezmění (vlastnost f). Dále je vhodné připomenout, že algoritmus lze ukončit i dříve, pokud najdeme dva shodné řádky, resp. nulový řádek (vlastnost a resp. b). V tom případně je determinant matice roven nule. [9 s. 45–46]

2.3.4 Sarrusovo pravidlo

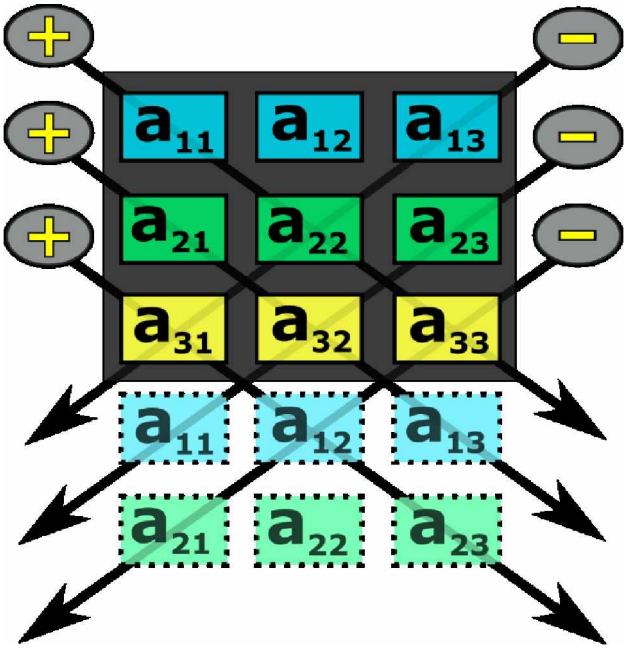
Vraťme se k výpočtu determinantu pro matice třetího řádu. Ten se podle definice vypočte následovně:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{21}a_{32}a_{13} + a_{31}a_{12}a_{23} -$$

$$- a_{13}a_{22}a_{31} - a_{23}a_{32}a_{11} - a_{33}a_{12}a_{21}$$

Tento vzorec je již poměrně dlouhý a hůře zapamatovatelný. Sarrusovo pravidlo je potom mnemotechnická pomůcka, která nám slouží pro snadné rozepsání výpočtu takového determinantu.

Podle tohoto pravidla k matici připíšeme jako čtvrtý a pátý řádek její první a druhý řádek. Poté vytvoříme součin prvků na hlavní diagonále, dále vytvoříme součiny z prvků v rovnoběžných liniích s hlavní diagonálou a tyto tři součiny sečteme. Poté vytvoříme součiny z prvků na vedlejší diagonále a na jejích dvou rovnoběžných liniích. Tyto součiny odečítáme. [1], [9] Situaci lépe popisuje obrázek 1.



Obrázek 1 – Sarrusovo pravidlo. Zdroj vlastní

Po získání praxe ani není nutné uvažované řádky opravdu přepisovat. Dále je dobré poznamenat, že Sarrusovo pravidlo je možné obdobně formulovat i pomocí sloupců matice.

2.3.5 Příklady

Příklad 1: Určete determinant matice A

$$A = \begin{pmatrix} 4 & 2 & -2 \\ 1 & -1 & -2 \\ -1 & 1 & 3 \end{pmatrix}$$

Řešení: Jelikož A je matice třetího řádu, můžeme použít Sarrusovo pravidlo i Gaussovu eliminaci. Nejprve si ukážeme řešení pomocí Sarrusova pravidla:

$$\begin{vmatrix} 4 & 2 & -2 \\ 1 & -1 & -2 \\ -1 & 1 & 3 \\ 4 & 2 & -2 \\ 1 & -1 & -2 \end{vmatrix}$$

Přepíšeme první dva řádky matice jako čtvrtý a pátý řádek. Nyní vezmeme prvky na hlavní diagonále a uděláme z nich součin. Totéž opakujeme pro dvě rovnoběžné linie pod ní. Tyto součiny sečteme a obdobně vytvoříme součiny z prvků hlavní diagonály a linií pod ní, které odečteme.

$$\begin{aligned} det\, A = {} & 4 \cdot (-1) \cdot 3 + 1 \cdot 1 \cdot (-2) + (-1) \cdot 2 \cdot (-2) - \\ & -(-2) \cdot (-1) \cdot (-1) - (-2) \cdot 1 \cdot 4 - 3 \cdot 2 \cdot 1 \end{aligned}$$

Nyní už jen dopočítáme výsledek:

$$det\, A = -12 - 2 + 4 + 2 + 8 - 6 = -6$$

Zkusme ještě vypočítat ten samý determinant pomocí Gaussovy eliminace:

$$\begin{vmatrix} 4 & 2 & -2 \\ 1 & -1 & -2 \\ -1 & 1 & 3 \end{vmatrix} \sim - \begin{vmatrix} 1 & -1 & -2 \\ 4 & 2 & -2 \\ -1 & 1 & 3 \end{vmatrix} \sim - \begin{vmatrix} 1 & -1 & -2 \\ 0 & 6 & 6 \\ -1 & 1 & 3 \end{vmatrix} \sim - \begin{vmatrix} 1 & -1 & -2 \\ 0 & 6 & 6 \\ 0 & 0 & 1 \end{vmatrix}$$

Teď již pouze vytvoříme součin ze všech prvků na hlavní diagonále a koeficientu vzniklého úpravami matice.

$$det\, A = -1 \cdot 1 \cdot 6 \cdot 1 = -6$$

Příklad 2: Určete determinant matice A

$$A = \begin{pmatrix} 2 & 8 & -2 & 1 \\ -1 & 6 & -4 & 3 \\ 0 & -4 & 2 & -3 \\ -1 & -6 & 2 & -2 \end{pmatrix}$$

Řešení:

Jelikož se jedná o matici čtvrtého řádu, využijeme metodu výpočtu pomocí Gaussovy eliminace.

$$\begin{vmatrix} 2 & 8 & -2 & 1 \\ -1 & 6 & -4 & 3 \\ 0 & -4 & 2 & -3 \\ -1 & -6 & 2 & -2 \end{vmatrix} \sim \begin{vmatrix} 2 & 8 & -2 & 1 \\ -1 & 6 & -4 & 3 \\ 0 & -4 & 2 & -3 \\ 0 & -12 & 6 & -5 \end{vmatrix} \sim \frac{1}{2} \begin{vmatrix} 2 & 8 & -2 & 1 \\ 0 & 20 & -10 & 7 \\ 0 & -4 & 2 & -3 \\ 0 & -12 & 6 & -5 \end{vmatrix} \sim$$

$$\sim \frac{1}{2} \begin{vmatrix} 2 & 8 & -2 & 1 \\ 0 & 20 & -10 & 7 \\ 0 & -4 & 2 & -3 \\ 0 & 0 & 0 & 4 \end{vmatrix} \sim \frac{1}{10} \begin{vmatrix} 2 & 8 & -2 & 1 \\ 0 & 20 & -10 & 7 \\ 0 & 0 & 0 & -8 \\ 0 & 0 & 0 & 4 \end{vmatrix} \sim \frac{1}{10} \begin{vmatrix} 2 & 8 & -2 & 1 \\ 0 & 20 & -10 & 7 \\ 0 & 0 & 0 & -8 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

Upravená matice obsahuje nulový řádek, proto $det\, A = 0$.

2.4 Inverzní matice

Dříve než začneme definovat inverzní matice, je vhodné znát pojem regulární, resp. singulární matice. Nechť A je čtvercová matice řádu n. Matice A se nazývá regulární, jestliže $rank(A) = n$. To znamená, že všechny řádky i sloupce regulární matice jsou lineárně nezávislé. Alternativně lze regulární matici definovat podle determinantu. Matice A se nazývá

regulární, jestliže $det\, A \neq 0$. V případě, že $rank(A) < n$, resp. $det\, A = 0$, matice se nazývá singulární. [5], [9]

Nechť A je čtvercová matice řádu n nad R. Inverzní matice k matici A, kterou budeme značit A^{-1}, je taková matice, pro kterou platí následující rovnost:

$$AA^{-1} = A^{-1}A = I$$

Inverzní matice ovšem neexistuje pro všechny čtvercové matice. Lze dokázat, že inverzní matice existuje pouze pro regulární matice (důkaz viz [5]). Dále platí, že pokud inverzní matice existuje, tak je určena jednoznačně, tedy ke každé matici existuje maximálně jedna matice inverzní.[1], [5], [8]

Pro určení inverzní matice (případně dokázání neexistence inverzní matice) k matici A je vhodné využít Gauss-Jordanův eliminační algoritmus (viz podkapitola 2.1.6). K matici A připíšeme na pravou stranu jednotkovou matici, a potom provádíme standardní řádkové úpravy. Pracujeme tedy s řádky o délce $2n$, kde n je řád matice A. Pokud narazíme na levé straně na nulový řádek, potom můžeme výpočet ukončit, jelikož víme, že matice je singulární, a tudíž k ní neexistuje inverzní matice. Postup výpočtu si názorně ukážeme na příkladech. [9]

2.4.1 Příklady

Příklad 1: Určete inverzní matici k matici A

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & -4 & 5 \\ 1 & 0 & 3 \end{pmatrix}$$

Řešení:

Nejprve vytvoříme rozšířenou matici – k matici A připíšeme na pravou stranu jednotkovou matici odpovídajícího řádu.

$$A = \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 5 & 0 & 1 & 0 \\ 1 & 0 & 3 & 0 & 0 & 1 \end{array}\right)$$

Nyní začneme provádět řádkové elementární úpravy:

$$\left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 5 & 0 & 1 & 0 \\ 1 & 0 & 3 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 5 & 0 & 1 & 0 \\ 0 & -1 & 1 & -1 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 5 & 0 & 1 & 0 \\ 0 & 0 & -1 & -4 & -1 & 4 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 5 & 0 & 1 & 0 \\ 0 & 0 & 1 & 4 & 1 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & -4 & 0 & -20 & -4 & 20 \\ 0 & 0 & 1 & 4 & 1 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 5 & 1 & -5 \\ 0 & 0 & 1 & 4 & 1 & -4 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|ccc} 1 & 1 & 0 & -7 & -2 & 8 \\ 0 & 1 & 0 & 5 & 1 & -5 \\ 0 & 0 & 1 & 4 & 1 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -12 & -3 & 13 \\ 0 & 1 & 0 & 5 & 1 & -5 \\ 0 & 0 & 1 & 4 & -1 & -4 \end{array}\right)$$

Na pravé straně již vidíme výslednou inverzní matici k matici A.

$$A^{-1} = \begin{pmatrix} -12 & -3 & 13 \\ 5 & 1 & -5 \\ 4 & -1 & -4 \end{pmatrix}$$

Příklad 2: Určete inverzní matici k matici A

$$A = \begin{pmatrix} -8 & -2 & -1 \\ 0 & 0 & 4 \\ -4 & -1 & 1 \end{pmatrix}$$

Řešení:

Postupujeme stejně jako v předchozím příkladu:

$$\left(\begin{array}{ccc|ccc} -8 & -2 & -1 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 \\ -4 & -1 & 1 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} -8 & -2 & -1 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 3 & -1 & 0 & 2 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} -8 & -2 & -1 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 & 3 & -8 \end{array}\right)$$

V poslední matici vidíme na levé straně nulový řádek, takže matice A je singulární a neexistuje k ní inverzní matice.

2.5 Vlastní čísla a vektory

Nechť A je čtvercová matice nad tělesem T. Jako charakteristickou matici matice A označujeme matici $A - \lambda I$, kde matice λI je jednotková matice vynásobena proměnnou λ. Jestliže A je matice řádu n, potom charakteristická matice k matici A je:

$$A - \lambda I = \begin{pmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{pmatrix}$$

Charakteristickým polynomem matice A potom rozumíme determinant její charakteristické matice neboli $det(A - \lambda I)$. Kořeny tohoto polynomu nazýváme vlastní čísla matice. Z této definice vyplývá, že i matice obsahující pouze reálná čísla mohou mít komplexní vlastní čísla. Jelikož polynomy mohou mít vícenásobné kořeny, tak násobnost tohoto kořene budeme označovat jako násobnost vlastního čísla. Dalším často využívaným pojmem je spektrum

matice. Jako spektrum matice A označujeme soubor utvořený ze všech vlastních čísel matice A, přičemž každé vlastní číslo se v něm vyskytuje právě tolikrát, jaká je jeho násobnost. [1]

Nechť číslo $\lambda \in C$ je vlastním číslem čtvercové matice A řádu n. Vektor $x \in C^n, x \neq o$ se nazývá vlastní vektor matice A (příslušný vlastnímu číslu λ), jestliže platí rovnost

$$Ax^T = \lambda x^T.$$

Z výše uvedené rovnosti plyne, že $(A - \lambda I)x = o$. Jelikož podle definice se $x \neq o$, tak uvedená rovnice musí mít nenulová řešení. Z toho vyplývá, že matice $A - \lambda I$ musí být singulární. Jelikož je matice $A - \lambda I$ singulární, pak má výše uvedená rovnice dle Frobeniovy věty nekonečně mnoho řešení, a tudíž ke každému vlastnímu číslu existuje nekonečně mnoho vlastních vektorů. [9]

Níže je uvedeno několik zajímavých vlastností vlastních čísel a vektorů. Pro důkazy těchto tvrzení viz [1].

a. Číslo 0 je vlastním číslem matice A pouze pokud je matice A singulární.
b. Pokud je a vlastní číslo matice A a x příslušný vlastní vektor, potom pro každé číslo $k \in N$ platí, že a^k je vlastní číslo matice A^k a x je příslušný vlastní vektor.
c. Matice AB a BA mají stejná vlastní čísla.
d. Nechť A je regulární matice nad T. Pokud je číslo $a \in T$ vlastním číslem matice A, potom číslo a^{-1} je vlastním číslem matice A^{-1}. Vektory příslušné těmto číslům jsou pro obě matice totožné.
e. Všechna vlastní čísla reálné symetrické matice jsou reálná.
f. Nechť A je matice nad tělesem T. Jsou-li $X_1, \ldots, X_k$ množiny lineárně nezávislých vektorů příslušné vlastním číslům matice A, potom je množina vektorů $X = X_1 \cup \cdots \cup X_k$ lineárně nezávislá.

K výpočtu vlastních čísel se využívá výše uvedených vztahů a definic, můžeme tedy rovnou přistoupit k praktickým ukázkám výpočtu.

2.5.1 Příklady

Příklad 1: Určete vlastní čísla a vektory matice A

$$A = \begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}$$

Řešení:

Nejprve vytvoříme charakteristickou matici matice A.

$$A - \lambda I = \begin{pmatrix} 5-\lambda & -2 \\ -2 & 5-\lambda \end{pmatrix}$$

Z této matice určíme charakteristický polynom vypočtením determinantu podle definice.

$$det(A - \lambda I) = (5-\lambda)(5-\lambda) - 4 = \lambda^2 - 10\lambda + 21$$

Nyní vypočítáme kořeny charakteristického polynomu:

$$\lambda^2 - 10\lambda + 21 = 0$$

$$(\lambda - 3)(\lambda - 7) = 0$$

Z této rovnice vyplývá, že $\lambda_1 = 3$ a $\lambda_2 = 7$. Nyní vypočítáme jednotlivé vlastní vektory. Nejprve určíme vlastní vektory příslušné vlastnímu číslu 3. To uděláme tak, že vlastní číslo 3 dosadíme do rovnice $(A - \lambda I)x = o$:

$$\left(\begin{array}{cc|c} 5-3 & -2 & 0 \\ -2 & 5-3 & 0 \end{array}\right) \sim \left(\begin{array}{cc|c} 2 & -2 & 0 \\ -2 & 2 & 0 \end{array}\right) \sim \left(\begin{array}{cc|c} 2 & -2 & 0 \\ 0 & 0 & 0 \end{array}\right) \sim (1 \quad -1|0)$$

Množina vlastních vektorů příslušných vlastnímu číslu 3 je: $\{t \cdot (1,1)\}$, kde $t \in R$

Nyní dopočítáme vlastní vektory příslušné číslu 7:

$$\left(\begin{array}{cc|c} 5-7 & -2 & 0 \\ -2 & 5-7 & 0 \end{array}\right) \sim \left(\begin{array}{cc|c} -2 & -2 & 0 \\ -2 & -2 & 0 \end{array}\right) \sim \left(\begin{array}{cc|c} 2 & 2 & 0 \\ 0 & 0 & 0 \end{array}\right) \sim (1 \quad 1|0)$$

Množina vlastních vektorů příslušných vlastnímu číslu 7 je: $\{t(-1,1)\}$, kde $t \in R$.

Příklad 2: Určete vlastní čísla a vektory matice A

$$A = \begin{pmatrix} -3 & 3 & -1 \\ -2 & 4 & -2 \\ -7 & 5 & -1 \end{pmatrix}$$

Řešení:

Nejprve vytvoříme charakteristickou matici matice A.

$$A - \lambda I = \begin{pmatrix} -3-\lambda & 3 & -1 \\ -2 & 4-\lambda & -2 \\ -7 & 5 & -1-\lambda \end{pmatrix}$$

Nyní vytvoříme charakteristický polynom vypočtením determinantu charakteristické rovnice. K výpočtu determinantu využijeme například Sarrusovo pravidlo.

$$det(A-\lambda I) = (-3-\lambda)(4-\lambda)(-1-\lambda) + (-2)\cdot 5(-1) + (-7)\cdot 3(-2) - \\ -(-1)(4-\lambda)(-7) - (-2)\cdot 5(-3-\lambda) - (-1-\lambda)\cdot 3(-2)$$

$$det(A-\lambda I) = -\lambda^3 + 13\lambda + 12 + 10 + 42 - 28 + 7\lambda - 30 - 10\lambda - 6 - 6\lambda$$

$$det(A-\lambda I) = -\lambda^3 + 4\lambda = \lambda^3 - 4\lambda$$

Nyní vypočítáme vlastní čísla neboli kořeny charakteristického polynomu:

$$\lambda^3 - 4\lambda = 0$$

$$\lambda(\lambda^2 - 4) = 0$$

$$\lambda(\lambda+2)(\lambda-2) = 0$$

Vlastní čísla matice A jsou tedy 0, 2 a -2. Nyní zbývá dopočítat vlastní vektory matice.

Pro $\lambda = 0$:

$$\left(\begin{array}{ccc|c} -3 & 3 & -1 & 0 \\ -2 & 4 & -2 & 0 \\ -7 & 5 & -1 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -3 & 3 & -1 & 0 \\ 0 & 6 & -4 & 0 \\ -7 & 5 & -1 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -3 & 3 & -1 & 0 \\ 0 & 6 & -4 & 0 \\ 0 & -6 & 4 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -3 & 3 & -1 & 0 \\ 0 & 6 & -4 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|c} -3 & 3 & -1 & 0 \\ 0 & 3 & -2 & 0 \end{array}\right)$$

Množina vlastních vektorů pro $\lambda = 0$: $\{t\left(\frac{1}{3}, \frac{2}{3}, 1\right)\}$, kde $t \in R$.

Pro $\lambda = 2$:

$$\left(\begin{array}{ccc|c} -3-2 & 3 & -1 & 0 \\ -2 & 4-2 & -2 & 0 \\ -7 & 5 & -1-2 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -5 & 3 & -1 & 0 \\ -2 & 2 & -2 & 0 \\ -7 & 5 & -3 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -5 & 3 & -1 & 0 \\ 0 & 4 & -8 & 0 \\ -7 & 5 & -3 & 0 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|c} -5 & 3 & -1 & 0 \\ 0 & 4 & -8 & 0 \\ 0 & 4 & -8 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -5 & 3 & -1 & 0 \\ 0 & 1 & -2 & 0 \end{array}\right)$$

Množina vlastních vektorů pro $\lambda = 2$: $\{t(1, 2, 1)\}$, kde $t \in R$.

Pro $\lambda = -2$:

$$\left(\begin{array}{ccc|c} -3+2 & 3 & -1 & 0 \\ -2 & 4+2 & -2 & 0 \\ -7 & 5 & -1+2 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -1 & 3 & -1 & 0 \\ -2 & 6 & -2 & 0 \\ -7 & 5 & 1 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -1 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -7 & 5 & 1 & 0 \end{array}\right) \sim$$

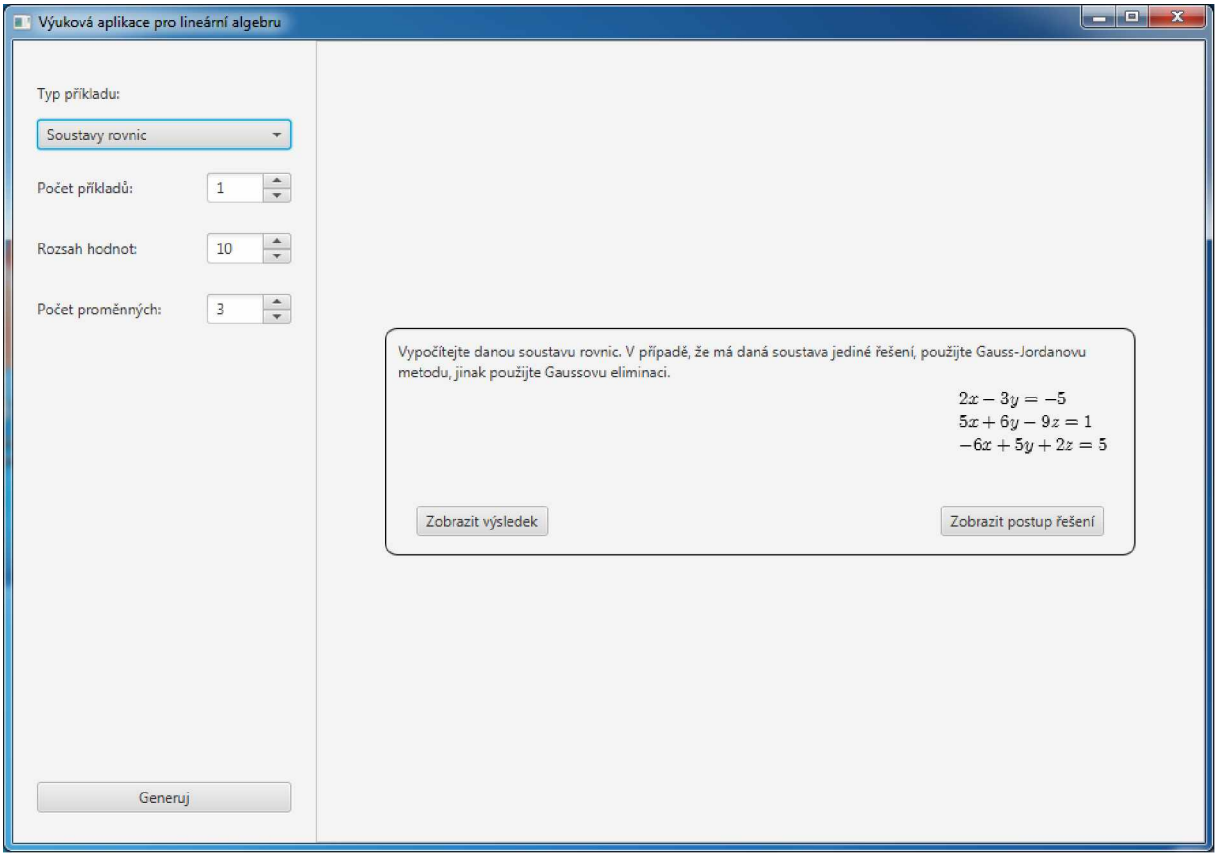
$$\sim \left(\begin{array}{ccc|c} -1 & 3 & -1 & 0 \\ 0 & -16 & 8 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} -1 & 3 & -1 & 0 \\ 0 & -2 & 1 & 0 \end{array}\right)$$

Množina vlastních vektorů pro $\lambda = -2$: $\{t\left(\frac{1}{2}, \frac{1}{2}, 1\right)\}$, kde $t \in R$.

3 UŽIVATELSKÁ DOKUMENTACE APLIKACE

3.1 Instalace a spuštění

Aplikaci není nutné instalovat. Pro spuštění aplikace stačí spustit LinearAlgebra.exe. Alternativně lze spustit aplikaci z příkazového řádku pomocí příkazu *java -jar LinearAlgebra.jar*. Pro spuštění aplikace je nutné mít nainstalované *JRE* (*Java Runtime Environment*) alespoň ve verzi 1.8.



Obrázek 2 – Aplikace po spuštění. Zdroj vlastní.

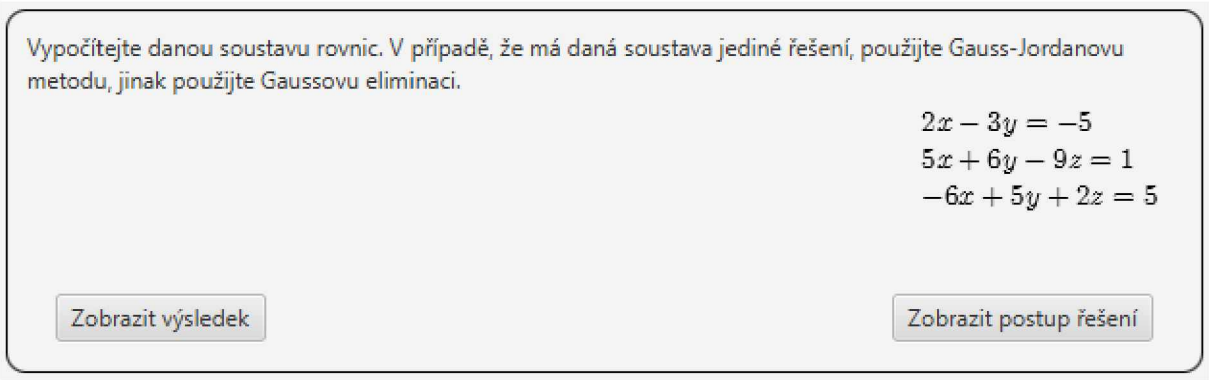
3.2 Použití aplikace

Aplikace obsahuje pouze jediné okno. To je rozděleno do dvou částí. V levé části se nachází komponenty pro ovládání generátoru příkladů. Nahoře se nachází combo box, kterým můžeme vybrat, jaký typ příkladu chceme vygenerovat. Obsahuje tyto hodnoty:

- soustavy rovnic,
- determinant,
- inverzní matice,
- Sarrusovo pravidlo,
- vlastní čísla a vektory,

- lineární kombinace vektorů,
- lineární závislost vektorů.

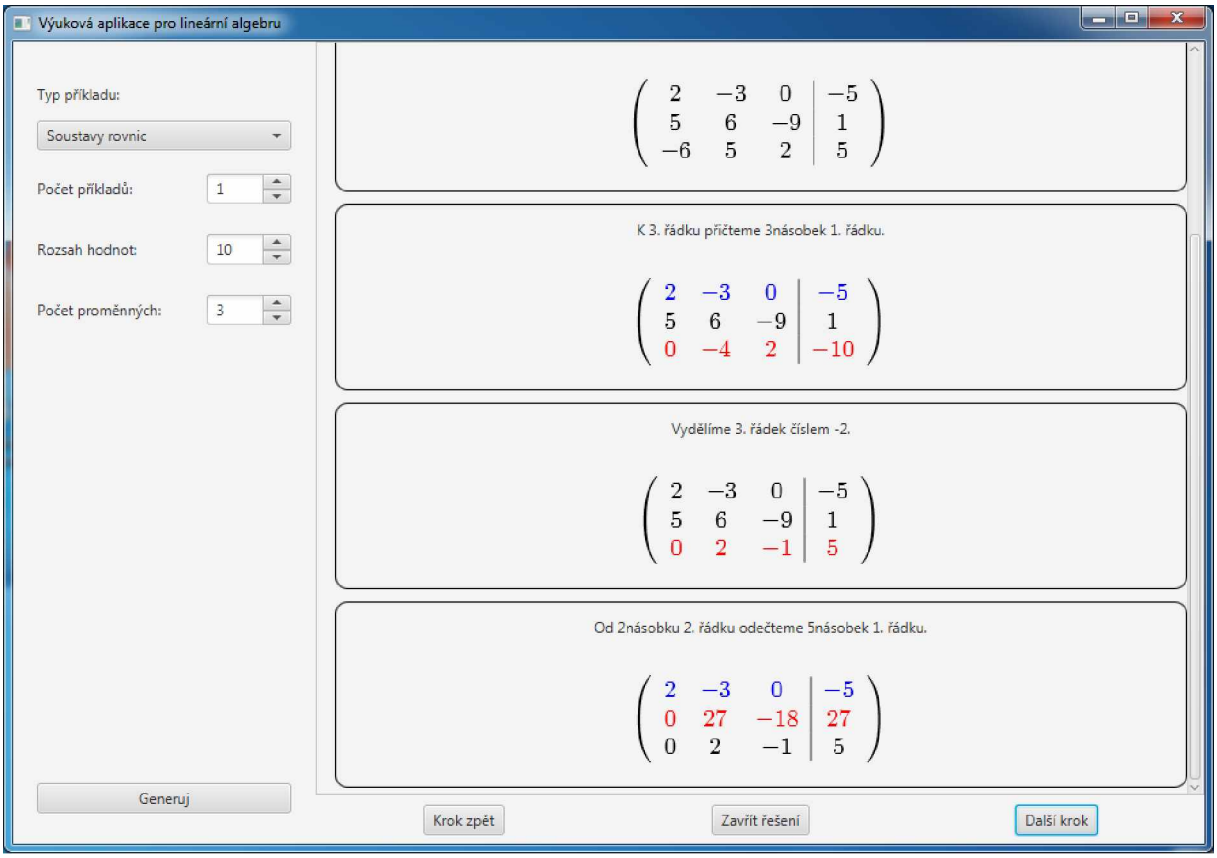
Pod tímto combo boxem se nachází spinner, kterým určujeme počet vygenerovaných příkladů. Může obsahovat celá čísla v rozsahu od 1 do 10, přičemž 1 je výchozí hodnota. Dále se zde nachází spinner, kterým ovládáme maximální absolutní hodnoty v zadáních generovaných příkladů. To znamená, že pokud je tento spinner nastaven např. na hodnotu 10, potom zadání příkladů obsahují hodnoty od -10 do 10. Rozsah hodnot tohoto spinneru je od 5 do 15, výchozí hodnota je 10. Pro všechny typy příkladů kromě Sarrusova pravidla objevíme ještě jeden spinner. Ten nám podle typu příkladu určuje velikost matice, počet proměnných či počet vektorů. Rozsah hodnot spinneru je od 2 do 5 (u typu vlastní čísla a vektory do 4), výchozí hodnota je 3. Ve spodní části této sekce se ještě nachází tlačítko *generuj*, kterým vygenerujeme příklady podle nastavení. Veškeré komponenty mají popisky, takže ovládání je velice intuitivní.



Obrázek 3 – Příklad se zobrazeným výsledkem. Zdroj vlastní.

V pravé části aplikace se zobrazují vygenerované příklady nebo jednotlivé kroky řešení daného příkladu. Každý příklad obsahuje v horní části text zadání, v pravé části konkrétní soustavu rovnic nebo matici či vektory zadání. V levé dolní části se nachází tlačítko na zobrazení nebo skrytí výsledku příkladu. Řešení příkladu se potom vyskytuje nad tímto tlačítkem. V pravé dolní části se nachází tlačítko pro zobrazení postupu řešení. Kliknutím na toto tlačítko skryjeme všechny vygenerované příklady a zobrazíme toto zadání jako první krok řešení příkladu. Zároveň se ve spodní části objeví lišta obsahující dvě až tři tlačítka. V levé části se nachází tlačítko *krok zpět*. Tímto tlačítkem skryjeme poslední krok. V případě, že je zobrazen pouze jeden krok (zadání), tak se toto tlačítko nezobrazuje. V prostřední části je vždy tlačítko *zavřít řešení*. Tímto tlačítkem skryjeme řešení příkladu i ovládací lištu a zobrazíme dříve vygenerované příklady. V pravé části se nachází tlačítko *další krok*, které nám odhalí další krok postupu řešení daného příkladu. Tento krok se zobrazí pod předchozí. V případě, že je kroků

více, než se jich vejde na obrazovku, objeví se na pravé straně posuvník umožňující procházet celé řešení. V případě, že jsou zobrazeny všechny kroky řešení, tlačítko *další krok* se skryje.



Obrázek 4 – Aplikace zobrazující postup řešení příkladu. Zdroj vlastní.

4 PROGRAMÁTORSKÁ DOKUMENTACE APLIKACE

4.1 Použité technologie

Tato aplikace je vytvořena v programovacím jazyce *Java*, pro tvorbu grafiky byla použita knihovna *JavaFX*. Pro usnadnění práce s maticemi byla využita knihovna *EJML* (*Efficient Java Matrix Library*). Pro přehledné zobrazení matic a vektorů je použita knihovna *JLaTeXMath* v kombinaci s komponentou *LatexView*. Spustitelný exe soubor byl vytvořen pomocí nástroje *Launch4j*.

4.1.1 Java

Java je objektově orientovaný jazyk vyvinutý společností *Sun Microsystems* v roce 1995. Od roku 2010 patří práva společnosti *Oracle Corporation*. Jedná se o jeden z nejoblíbenějších a nejrozšířenějších programovacích jazyků na světě. Jedná se o multiplatformní jazyk, který ke svému běhu potřebuje pouze *JVM* (*Java Virtual Machine*). Práce využívá verzi *Java SE 8* (*Standard Edition*). Tuto vezri Javy implementují dva hlavní produkty: *JRE 8* (*Java SE Runtime Environment*) sloužící k bezproblémovému běhu veškerých aplikací vytvořených v této verzi jazyka a *JDK 8* (*Java SE Development Kit*), který obsahuje veškeré komponenty *JRE 8*, ke kterým přidává nástroje pro vývoj aplikací. Pro více informací viz [12].

4.1.2 JavaFX

JavaFX je knihovna grafických a multimediálních balíčků umožňujících tvorbu grafického uživatelského rozhraní pro platformu *Java*. Tato knihovna nahrazuje již zastaralé grafické knihovny jako jsou *AWT* (*Abstract Window Toolkit*) nebo *Swing*. Tato knihovna je součástí *Javy SE* ve verzích 7 až 11. Pro více informací navštivte oficiální dokumentaci [13].

4.1.3 EJML

EJML je knihovna pro lineární algebru a snadnou manipulaci se všemi druhy matic. Tato knihovna byla vyvinuta v roce 2014 Peterem Abelesem. V práci je použita verze 0.39, která byla vydána 7. 4. 2020. Pro více informací navštivte oficiální stránky projektu **Chyba! N enalezen zdroj odkazů.**.

4.1.4 JLaTeXMath

JLaTeXMath je knihovna, jejímž účelem je zobrazování matematických formulí zapsaných v *LaTeXu*. Tato knihovna vznikla již v roce 2009. V práci je použita aktuálně poslední verze 1.0.7 z 18. 4. 2018. Pro více informací navštivte stránku projektu [15].

4.1.5 LatexView

LatexView je upravená komponenta pro *JavuFX*, která slouží ke snadnému renderování textu a matematických formulí zapsaných pomocí *LaTeXu*. K tomu využívá knihovnu *JLaTeXMath*. Komponentu vyvinul Egor Makarenko v roce 2019. V práci je použita ve verzi 0.6.1 z 24. 6. 2019. Pro více informací navštivte stránku projektu [16].

4.1.6 Launch4j

Launch4j je multiplatformní bezplatný nástroj pro zabalení spustitelných jar souborů do formátu exe, který lze snadno spouštět na platformně *Windows*. Tento nástroj byl napsán v jazyce *Java* a je velice malý (pouze 35 kB), ale poskytuje velikou řadu nastavení vytvářeného exe souboru. Tento nástroj je vyvíjen již od roku 2003, poslední verze 3.12 pochází z roku 2018. V práci byla použita právě tato verze. Pro více informací navštivte webové stránky projektu [17].

4.2 Architektura aplikace

Aplikace striktně odděluje výpočetní část a grafickou část. Z tohoto důvodu obsahuje dva hlavní balíčky: *gui* a *exercise*.

4.2.1 Balíček gui

Tento balíček obsahuje veškeré komponenty potřebné ke správnému zobrazování příkladů. Tento balíček obsahuje třídu *Main*, pomocí které se aplikace spouští, a balíčky *mainWindow*, *exercises*, *solution* a *formatStep*.

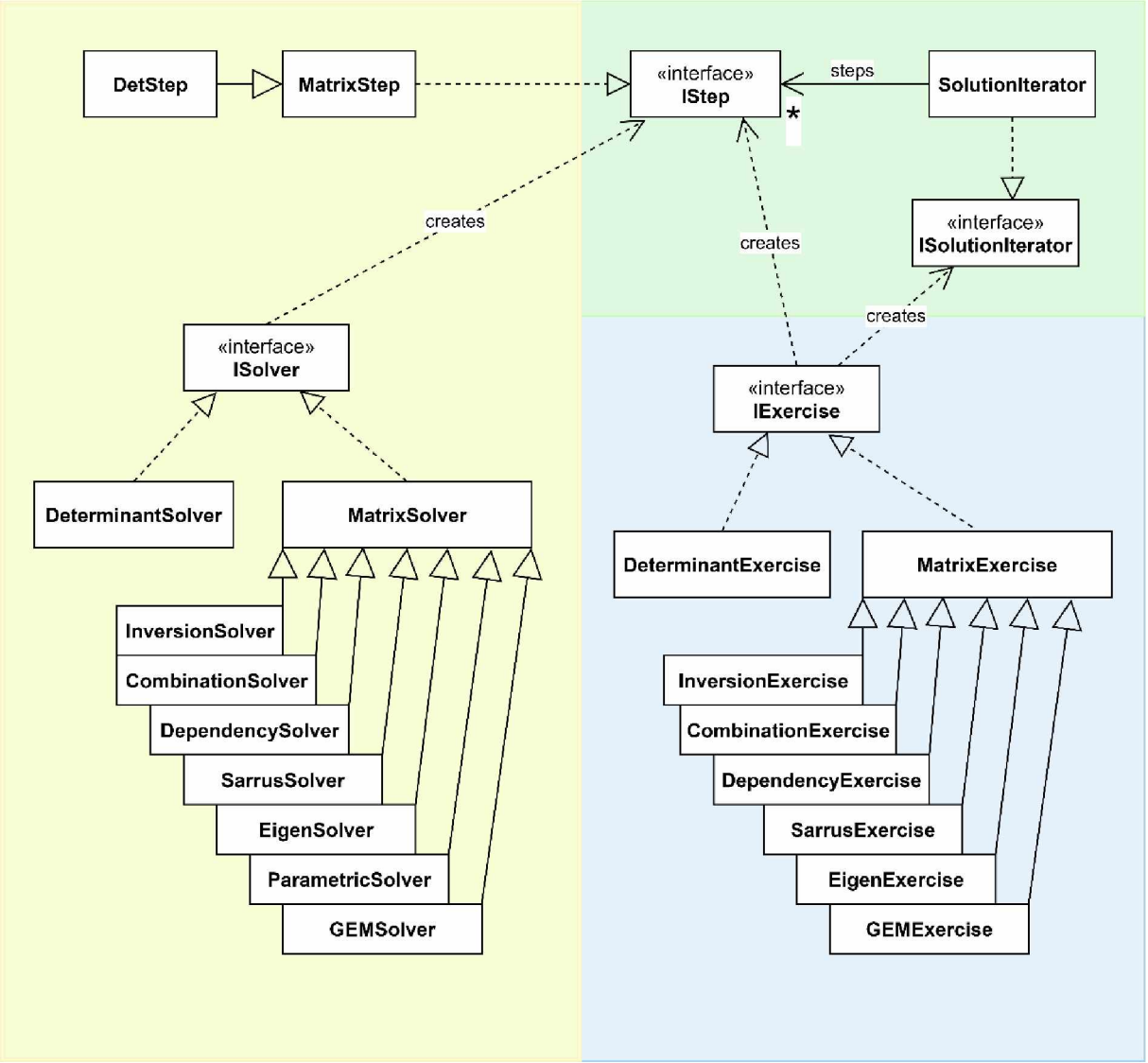
Balíček *mainWindow* obsahuje soubor *Mainwindow.fxml*, který obsahuje definici hlavního okna, a jeho controller, který se nazývá *MainWindowController*.

Balíčky *exercises* a *solution* obsahují podobně jako balíček *mainWindow* dva soubory na zobrazení a ovládání jednotlivých příkladů, resp. jednotlivých kroků daného příkladu.

Balíček *formatStep* obsahuje rozhraní *IFormatter*, které zapouzdřuje třídy *formatter* pro jednotlivé typy příkladů. Tyto třídy formátují jednotlivé kroky příkladů do *LaTeX* formulí, díky čemuž jsou jednotlivé příklady zobrazeny v přehledném formátu. Tyto třídy jsou vytvořeny podle návrhového vzoru *singleton*. V balíčku se dále nachází pomocná třída *FormatterUtil*, která obsahuje statické metody pro usnadnění tvorby *LaTeX* formulí. Poslední třídou tohoto balíčku je *FormatterFactory*. Tato třída obsahuje jedinou statickou metodu *getFormatter*, která přijímá typ příkladu a vrací odpovídající formatter.

4.2.2 Balíček exercise

Tento balíček je daleko obsáhlejší než balíček *gui*. Obsahuje veškeré třídy potřebné pro generování a řešení jednotlivých příkladů. Tento balíček celkem obsahuje dalších 7 balíčků, 3 rozhraní, dvě abstraktní třídy, čtyři normální třídy a jeden výčet. Vztahy mezi těmito komponentami obrázek 5.



Obrázek 5 – UML diagram tříd balíčku gem. Zdroj vlastní

Jednotlivé balíčky, které se jmenují *determinant*, *eigen*, *gem*, *inversion*, *vector*, *sarrus* a *utils*, obsahují třídy, jejichž názvy končí slovy *exercise*, *factory* nebo *solver*. Tyto třídy udržují informace o příkladech, vytvářejí příklady a řeší je. Jak to tyto třídy dělají, bude vysvětleno v dalších podkapitolách. Výjimkami jsou balíčky *determinant*, který kromě těchto tříd obsahuje ještě třídu *DeterminantStep*, dále balíček *gem*, který obsahuje balíčky *inversion* a *vector*, a poslední výjimkou je balíček *utils*.

Balíček *utils* obsahuje tři statické třídy, které se nazývají *Generator*, *MathFunctions* a *StringCreator*. Třída *Generator* obsahuje funkce pro generování různých druhů matic a vektorů potřebných pro tvorbu příkladů. Třída *MathFuctions* obsahuje dva druhy metod, První kategorie metod vytváří hluboké kopie matic a vektorů, případně je převádí do jiného formátu. Druhou kategorií jsou metody, které provádí nějakou matematickou operaci, např. sečtení řádků matice. Oba druhy metod jsou využívány při tvorbě řešení jednotlivých příkladů. Třída *StringCreator* obsahuje metody pro vytvoření popisu některých kroků výpočtu, většinou z důvodu složitého tvoření popisku, které by značně znepřehlednilo kód jednotlivých řešitelů příkladů.

Balíček *exercise* dále obsahuje třídy *Fraction* resp. *ParametricFraction*. Tyto třídy zastupují zlomek, resp. zlomek s parametrem. Třída *ParametricFraction* je potomkem třídy *Fraction*. V těchto třídách jsou definovány metody pro standardní operace se zlomky, jako je např. sčítání, násobení či absolutní hodnota zlomku. Tyto třídy jsou využívány z důvodu přehledného zobrazování. V případě ručního výpočtu libovolného příkladu na papír také používáme zlomky namísto desetinných čísel.

Další důležitou komponentou tohoto balíčku je výčet s názvem *Type*. Ten definuje typy příkladů, kterým se tato aplikace věnuje.

Ostatní třídy tohoto balíčku jsou využívány pro udržení a předávání informací o příkladech či jejich krocích. Výjimkou je pouze třída *MatrixSolver* věnující se výpočtům příkladů. Všechny tyto třídy si popíšeme v dalších podkapitolách.

4.3 Uchovávání a předávání příkladů

4.3.1 Uchovávání jednotlivých kroků

Pro uchovávání jednotlivých kroků každého příkladu se využívají třídy implementující rozhraní *IStep*. Toto rozhraní slouží hlavně k zapouzdření jednotlivých kroků a předání informací o nich do tříd v balíčku *gui*. Obsahuje pouze tři metody.

První metoda se nazývá *getData*. Tato metoda nemá žádné parametry a předává nám konkrétní matematický výraz. Pro skrytí implementace a sjednocení manipulace předává tyto informace ve formě Stringu. Další metoda nese název *getExplanation* a má stejnou signaturu jako předchozí metoda. Tato metoda slouží k předání popisu daného kroku. Poslední metodou je metoda *getSize*, která vrací integer. Slouží k předání informace o počtu sloupců daného kroku. Tato informace usnadňuje nastavení velikosti jednotlivých kroků a příkladů při zobrazení.

Třídy implementující toto rozhraní jsou dvě: *MatrixStep* a *DeterminantStep*. Třída *MatrixStep* uchovává informace o jednotlivých matematických výrazech ve formě dvourozměrného pole zlomků (třída *Fraction*). Kromě konstruktorů a výše uvedených metod obsahuje ještě metodu *getMatrix*, která předává informace o daném výrazu v nezměněné podobě (na rozdíl od metody *getData*). Tato metoda se využívá při vytváření řešení příkladů. Třída *DeterminantStep* dědí ze třídy *MatrixStep*. Kromě výše uvedených atributů a metod obsahuje ještě informaci o multiplikátu ve formě zlomku a metodu pro jeho vrácení.

4.3.2 Procházení jednotlivých kroků řešení

O procházení jednotlivých kroků řešení se stará třída implementující rozhraní *ISolutionIterator*. Jedná se o klasický iterátor, který umožňuje průchod vpřed i zpět. Obsahuje čtyři metody typické pro iterátory.

Tyto metody se nazývají *hasNextStep* resp. *hasPreviousStep*, které informují o existenci dalšího resp. předchozího prvku. Dále potom metody *nextStep* resp. *previousStep,* které vrací další resp. předchozí prvek (třída implementující rozhraní *IStep*) a posouvají ukazatel aktuálního prvku. V případě neexistence prvku tyto metody vyhazují výjimku *NoSuchElementException*.

Třída implementující toto rozhraní je pouze jedna, nazývá se *SolutionIterator*. Slouží k iterování jednotlivých kroků, které obdrží ve formě listu.

4.3.3 Uchovávání kompletních příkladů

Pro uchovávání a předávání informací o celých příkladech včetně kompletního řešení se starají třídy implementující rozhraní *IExercise*. Toto rozhraní předepisuje hned několik metod. Jako první uveďme metodu *getTaskText*, která vrací popis zadání. Dalšími metodami jsou *getTask* a *getSolution*, které vrací zadání, resp. výsledek příkladu jako instanci typu *IStep*. Dále Obsahuje metodu *getFullSolution*, která vrací všechny kroky řešení příkladu ve formě instance typu *ISolutionIterator*. Další metodou je *getType*, která vrací typ daného příkladu. Poslední metou je *getSize*, která slouží k předání maxima rozměrů zadání a řešení z důvodu usnadnění formátování aplikace v balíčku *gui*.

Toto rozhraní částečně implementuje abstraktní třída *MatrixExercise*, která uchovává informace o zadání jako instanci typu *MatrixStep* a informace o řešiteli jako instanci typu *MatrixSolver*. Z této třídy potom dědí jednotlivé třídy reprezentující konkrétní typ příkladu. Těmito třídami jsou *GEMExercise*, *EigenExercise*, *InversionExercise*, *CombinationExercise*, *DependencyExercise* a *SarrusExercise*. Tyto třídy jsou si navzájem velice podobné, liší se pouze ve vytváření konkrétního řešitele a návratové hodnotě metody *getType*.

Další třída implementující rozhraní *IExercise* je třída *DeterminantExercise*. Tato třída je opět velice podobná předchozím, ale informace o zadání uchovává jako instanci typu *DeterminantStep*, a proto nemůže dědit ze třídy *MatrixExercise*.

4.4 Řešení jednotlivých příkladů

Pro řešení příkladů jsou využívány třídy implementující rozhraní *ISolver*. Toto rozhraní předepisuje pouze dvě metody. První z nich je metoda *getSolution*, která vrací výsledek příkladu jako instanci typu *IStep*. Druhá metoda se nazývá *getFullSolution* a vrací všechny kroky řešení jako List *IStepů*.

Toto rozhraní implementuje abstraktní třída *MatrixSolver*, která slouží jako pomocná třída pro všechny typy příkladů využívající *MatrixStep* pro uchování informací o jednotlivých krocích. Tato třída uchovává informace o všech krocích daného řešení ve formě listu, dále uchovává matematický výraz posledního vypočteného kroku řešení ve formě dvourozměrného pole zlomků a konečné řešení příkladu jako instanci třídy *MatrixStep*. Tato třída obsahuje kromě metod definovaných rozhraním dvě chráněné metody pro snadné přidávání jednotlivých kroků (metoda *addStep*) a volání jiného řešitele v průběhu řešení (metoda *callAnotherSolver*). Poslední metodou je veřejná metoda *getMaxAbsNumberInSolution*, která, jak název napovídá, vrací informaci o velikosti největšího prvku vyskytujícího se v průběhu řešení. Tato metoda se využívá při generování vhodných zadání příkladů.

4.4.1 Řešení soustav rovnic

Pro řešení soustav rovnic se využívá třída *GEMSolver*, která je potomkem třídy *MatrixSolver*. Tato třída je velice důležitá, jelikož se využívá při řešení i dalších typů příkladů. Třída obsahuje kromě zděděných metod další tři veřejné metody. Jimi jsou metody *isSolvable* a *isRegular*, které informují o řešitelnosti soustavy, resp. o tom, zda je matice soustavy regulární, a *getSolvedVariables*, která vrací vyřešené proměnné ve formě hashmapy. Tyto metody mají využití hlavně při zavolání tohoto řešitele jinými.

Hlavní metodou této třídy je ovšem privátní metoda *makeSolution*, která se stará o vytvoření kompletního řešení a je zavolána již v konstruktoru. Tato metoda nejprve přepíše soustavu rovnic do rozšířené matice soustavy, v případě, že již dostaneme matici soustavy, tak je tento krok vynechán. Dále se projdou všechny řádky této matice. Všechny nulové řádky jsou vyškrtnuty, a pokud má řádek největší společný dělitel větší než jedna, tak je daný řádek tímto číslem vydělen.

Dále jsou prováděny jednotlivé řádkové elementární úpravy pro vytvoření schodovité matice. Způsob řešení zobrazuje následující kód:

```
for (int i = 0; i < lastMatrix.length; i++) {
    removeAllIdenticalRows();
    switchZeroFromPivot(i);
    if (!solvable) return;
    int column = MathFunctions.findFirstNonZeroFraction(lastMatrix[i]);
    for (int j = i+1; j < lastMatrix.length; j++) {
        int[] rows = new int[2];
        if (!solvable){
            return;
        }
        if (lastMatrix.length > 1 &&
              MathFunctions.findLcmRowsByColumn(lastMatrix, i,
              column, true, rows)) {
            if (createSteps(rows[0], rows[1], column)){
                j--;
            }
        }
    }
}
```

V následujícím textu jsou popsány důležité části algoritmu. Proměnná *column* nám říká, ve kterém sloupci se nachází pivot. Vnořený for cyklus prochází všechny řádky pod pivotem. Metoda *findLcmRowsByColumn* vyhledává nejvhodnější řádky pro vynulování prvku pod pivotem, Tyto řádky jsou vybírány tak, aby bylo nutné co nejméně násobit. Metoda nevrací tyto řádky, ale informaci, zda pod pivotem ještě existuje nenulový prvek. Indexy těchto řádků ukládá do pole v parametru (v našem případě pole *rows*). Metoda *createSteps* potom provede potřebné kroky k vynulování daného prvku. Dále zkontroluje, zda není řádek nulový nebo ho není vhodné vydělit. Metoda vrací boolean, aby v případě vyškrtnutí řádku nebyl další řádek přeskočen.

Nyní máme schodovitou matici soustavy a potřebujeme dokončit řešení. V případě, že má matice pouze jedno řešení, pokračuje algoritmus v řádkových elementárních úpravách, což znamená, že provádí Gauss-Jordanovu eliminaci. Algoritmus je velice podobný předešlému algoritmu, proto zde není uveden. Ještě před prováděním tohoto algoritmu je zkontrolováno, zda se nejedná o homogenní soustavu rovnic. V takovém případě se algoritmus neprovádí a je rovnou určen výsledek (všechny proměnné se rovnají nule).

Pokud má soustava rovnic nekonečně mnoho řešení, je zavolána třída *parametricSolver*. Tato třída je také potomkem třídy *MatrixSolver* a obsahuje metodu *getSolvedVariables*, která má stejný účel jako v případě *GEMSolveru*. O výpočet řešení se stará metoda *makeParametricSolution*, která je velice jednoduchá.

```
private void makeParametricSolution(){
    rewriteToEquations();
    solveVars();
    for (int i = 0; i < eliminatedLinesCount; i++) {
        addParameter(i);
        solveVars();
    }
}
```

Metoda *rewriteToEquations*, převádí rozšířenou matici soustavy zpět na rovnice. Metoda *solveVars* projde všechny rovnice a odhalí již vyřešené proměnné. Tyto proměnné poté dosadí do ostatních rovnic a převede čísla a parametry na pravé strany. Tento postup se opakuje, dokud lze vyřešit libovolnou proměnnou. Metoda *addParameter* dosazuje za poslední nevyřešenou proměnnou. Hodnota *eliminatedLinesCount* udává rozdíl mezi počtem rovnic a počtem proměnných. Tím je zajištěno dosazení správného počtu parametrů a korektní vyřešení rovnice. Na závěr je důležité podotknout, že uvedený algoritmus funguje pouze pro matice ve schodovitém tvaru.

4.4.2 Výpočet inverzní matice

Třída starající se o výpočet inverzní matice se nazývá *InversionSolver*. Tato třída je potomkem třídy *MatrixSolver* a neobsahuje žádné další veřejné metody. O vytvoření řešení se stará privátní metoda *makeSolution*.

```
private void makeSolution(){
     createExtendedMatrix();
     GEMSolver solver = new GEMSolver(solution, lastMatrix.length);
     callAnotherSolver(solver);


if (solver.isRegular()){
          Fraction[][] newMatrix =
              MathFunctions.clone2DArray(lastMatrix,lastMatrix.length,
              lastMatrix.length, 0, lastMatrix.length);
          addStep(new MatrixStep(newMatrix, "Na pravé straně nám vznikla
               inverzní matice k původní matici. Pro přehlednost odstraníme
               jednotkovou matici na levé straně"));
     }else {
          Fraction[] zero = new Fraction[]{new Fraction(0)};
          addStep(new MatrixStep(zero,1,1, "k zadané matici neexistuje
          inverzní matice"));
     }
}
```

Metoda *createExtendedMatrix* přidá k matici na pravou stranu odpovídající jednotkovou matici. Poté je zavolán *GEMSolver*, který se stará o celý výpočet. Nakonec je již pouze vypsán výsledek. Jelikož se o celý výpočet stará *GEMSolver*, tak se celý balíček *inversion* nachází v balíčku *gem*.

4.4.3 Výpočet lineární kombinace vektorů

O určení, zda je nějaký vektor lineární kombinací ostatních, se stará třída *CombinationSolver*. Tato třída opět dědí ze třídy *MatrixSolver* a neobsahuje žádné další veřejné metody. O výpočet se stará privátní metoda *makeSolution*.

```
private void makeSolution(){
    removeVectorNames();
    GEMSolver solver = new GEMSolver(solution, 1);
    callAnotherSolver(solver);
    finishSolution(solver);
}
```

Jelikož se opět o většinu výpočtu stará *GEMSolver*, tak je balíček obsahující tuto třídu umístěn v balíčku *gem*.

4.4.4 Výpočet lineární závislosti vektorů

Třída řešící lineární závislost vektorů se nazývá *DependencySolver*. Jedná se o dalšího potomka třídy *MatrixSolver*. Jelikož jsou témata lineární kombinace a lineární závislost vektorů velmi podobná, je velice podobné i řešení těchto příkladů. O výpočet se stejně jako v případě *CombinationSolveru* stará metoda *makeSolution*.

```
private void makeSolution(){
    removeVectorNames();
    GEMSolver solver = new GEMSolver(solution, 1);
    callAnotherSolver(solver);
    List<Fraction> variables = rewriteToEquations(solver);
    finishSolution(variables);
}
```

Rozdíl v řešení je pouze v odlišném vyjádření výsledků. Jelikož si jsou obě témata tak podobná, nacházejí se ve stejném balíčku nesoucí název *vector*.

4.4.5 Výpočet vlastních čísel a vektorů

Pro výpočet vlastních čísel a vektorů slouží třída *EigenSolver*. Jedná se o další třídu dědící z třídy *MatrixSolver*. Třída opět neobsahuje žádné veřejné metody. Zvláštností této třídy je, že vlastní čísla jsou obdržena v konstruktoru, nejsou tedy počítána touto třídou. Ta pouze zobrazuje postup, jak je lze získat, a počítá odpovídající vlastní vektory. O vytvoření řešení se opět stará privátní metoda *makeSolution*.

```
private void makeSolution() {
    addLambdas();
    if (createPolynom()) {
        rewriteToPolynoms();
    } else {
        polynomFromSolution();
    }
    createCharacteristicEquation();
    solveEquation();
```

```
    for (Double eigenValue : nonDuplicateEigenNumbers) {
        findVector(eigenValue);
    }
    finishSolution();
}
```

Metoda *addLambdas* vytvoří z původní matice charakteristickou matici. Metoda *createPolynom* vytvoří mezikrok výpočtu determinantu charakteristické matice, ale pouze v případě, že je řád matice menší než 4. Metody *rewriteToPolynoms* a *polynomFromSolution* nevytváří žádný krok výpočtu, pouze vytváří polynom, ze kterého potom lze snadno vytvořit charakteristickou rovnici. O to se stará metoda *createCharacteristicEquation*. Metoda *solveEquation* slouží k výpisu kořenů charakteristické rovnice. Jak již bylo zmíněno, tato metoda kořeny nepočítá, ale pouze vypisuje. Metoda *findVector* počítá vlastní vektory příslušné danému vlastnímu číslu. K výpočtu je využíván *GEMSolver*. Poslední metoda *finishSolution* už pouze vypisuje kompletní výsledky.

4.4.6 Výpočet determinantu pomocí Sarrusova pravidla

Pro výpočet determinantu pomocí Sarrusova pravidla existuje třída *SarrusSolver*. Je to poslední třída dědící z *MatrixSolveru*. Vypadá velmi podobně jako předchozí třídy řešící příklady. O výpočet se stará privátní metoda *makeSolution*.

```
private void makeSolution(){
    createExtendedMatrix();
    Fraction solution = new Fraction(0);
    Fraction[] diagonal;
    for (int i = 0; i < NUMBER_OF_COLUMNS; i++) {
        diagonal = getMainDiagonal(i);
        solution = computeIntermediateSolution(solution, diagonal, i);
    }
    for (int i = 0; i < NUMBER_OF_COLUMNS; i++) {
        diagonal = getSecondaryDiagonal(i);
        solution = computeIntermediateSolution(solution, diagonal, i+3);
    }
    finalSolution();
}
```

První metoda s názvem *createExtendedMatrix* přepisuje první dva řádky matice pod původní matici. Zlomek *solution* udržuje informace o mezivýsledcích a pole zlomků s názvem *diagonal* nám poskytuje konkrétní diagonálu. O předání správné diagonály se starají metody *getMainDiagonal*, resp. *getSecondaryDiagonal*. Metoda *computeIntermediateSolution* se stará o vynásobení konkrétní diagonály a její přičtení či odečtení od předchozích mezivýsledků. Jelikož Sarrusovo pravidlo slouží pro výpočet matic třetího řádu, je konstanta *NUMBER_OF_MATRIX* rovna třem.

4.4.7 Výpočet determinantu pomocí Gaussovy eliminace

K řešení toto typu příkladu se využívá *DeterminantSolver*. Je to jediný řešitel příkladů, který nedědí ze třídy *MatrixSolver*, ale pouze implementuje rozhraní *ISolver*. Tato třída totiž pracuje s kroky typu *DeterminantStep*, a to z toho důvodu, že při řádkových úpravách se mění hodnota determinantu, a proto je potřeba tyto změny někam ukládat (jsou ukládány jako multiplikát, viz podkapitola 4.3.1). O řešení příkladu se stará privátní metoda *makeFullSolution*. Řešení je velice podobné řešení soustav rovnic, ale kvůli drobným rozdílům bohužel nelze použít *GEMSolver*.

```
private void makeFullSolution() {
    for (int i = 0; i < lastMatrix.length; i++) {
        divideRowSteps(i);
        if (checkZeroedRow(i)){
            zeroSolution();
            return;
        }
    }
    for (int i = 0; i < lastMatrix.length-1; i++) {
        sortRows(i);
        int column = MathFunctions.findFirstNonZeroFraction(lastMatrix[i]);
        for (int j = i+1; j < lastMatrix.length; j++) {
            int[] rows = new int[2];
            if (MathFunctions.findLcmRowsByColumn(lastMatrix, i, column,
                    true, rows)) {
                nullNumberInRow(rows[0], rows[1], column);
            }
            if (checkZeroedRow(rows[1])){
                zeroSolution();
                return;
            }
        }
    }
    getDiagonal();
    getFinalSolution();
}
```

První for cyklus vydělí řádky, pokud je to vhodné, a zjistí, zda determinant neobsahuje nulový řádek. V tom případě je determinant je roven nule a řešení je ukončeno. Další průběh algoritmu je velice podobný jako algoritmus v podkapitole 4.4.1. Rozdíl je pouze v metodě *sortRows*, která prohazuje řádky tak, aby byl pivot neboli číslo na diagonále co nejmenší, a tím pádem řešení determinantu co nejjednodušší. Poslední změna je ukončení řešení v případě nalezení nulového řádku. Všechny metody jsou samozřejmě upraveny pro práci s multiplikátem tak, aby byl výpočet determinantu korektní.

4.5 Generování vhodných příkladů

O generování příkladů se starají třídy, jejichž název končí slovem *Factory*. Tyto třídy zpravidla obsahují jedinou veřejnou metodu, která se jmenuje *generateExercise*. Tyto metody jsou

statické a vracejí instanci typu *IExercise*. Důvod, proč tyto třídy nemají společné rozhraní či předka, je ten, že mohou obsahovat odlišné parametry.

4.5.1 Generování soustav rovnic

Aplikace generuje příklady, které mají jedno, žádné i nekonečně mnoho řešení. Tyto příklady jsou rozděleny v poměru 50 %, 25 % a 25 %, kde nejvyšší podíl mají příklady s jediným řešením. O generování těchto příkladů se stará třída *GEMFactory*.

Generování příkladů, které mají žádné nebo nekonečně mnoho řešení, je triviální. Pouze vygenerujeme libovolnou matici správných rozměrů a poté vytvoříme z jednoho řádku lineární kombinaci ostatních. Takové matice generuje statická metoda *generateSingularMatrix* třídy *Generator*, V případě, že chceme vytvořit příklad, který nemá řešení, tak ještě změníme hodnoty na pravých stranách rovnic.

Generování příkladů s jediným řešením je složitější. Nejprve vygenerujeme řešení dané soustavy rovnic. Poté vygenerujeme libovolný vektor jako koeficienty proměnných dané rovnice a pravou stranu vypočítáme jejich roznásobením s řešením dané soustavy. Opakováním tohoto postupu pro každou rovnici vznikne soustava rovnic, jejíž kompletní řešení je celočíselné. Jelikož chceme omezit velikost všech vygenerovaných hodnot zadání, tak generujeme tyto hodnoty tak dlouho, dokud pravá strana není menší než definované maximum.

```
double[] solutionVector = Generator.generateDoubleVector(numberOfRows, -
max, max);
double[][] taskDoubleMatrix = new double[numberOfRows][numberOfRows];
boolean again;
double[] taskDoubleVector = new double[numberOfRows];
do {
    for (int i = 0; i < numberOfRows; i++) {
        again = true;
        while (again){
            again = false;
            taskDoubleMatrix[i] =
                Generator.generateDoubleVector(numberOfRows, -max, max);
            taskDoubleVector[i] = 0;
            for (int j = 0; j < taskDoubleMatrix[i].length; j++) {
                double value = taskDoubleMatrix[i][j];
                taskDoubleVector[i] += value * solutionVector[j];
            }
            if (Math.abs(taskDoubleVector[i])> max){
                again = true;
            }
        }
    }
}while (new SimpleMatrix(taskDoubleMatrix).determinant() == 0);
```

Takto vypadá výše popsaný algoritmus, kde *solutionVector* reprezentuje řešení dané soustavy, *taskDoubleMatrix* koeficienty proměnných dané matice a *taskDoubleVector* jednotlivé hodnoty pravých stran.

Před schválením jakéhokoliv příkladu je ještě zkoumána maximální hodnota, která se vyskytne v průběhu jeho řešení. K tomu slouží metoda definovaná v *MatrixSolveru* s názvem *getMaxAbsNumberInSolution*. Tato hodnota nesmí přesáhnout výraz $\sqrt{max^n}$, kde *max* je maximální velikost generovaných hodnot v zadání a n je počet rovnic. Např. pro soustavu rovnic obsahující tři rovnice s maximem 10 je to přibližně 31,6.

4.5.2 Generování matic pro výpočet inverzní matice

Generování těchto příkladů má na starost třída *InversionFactory*. Pro tento účel jsou generovány pouze matice, jejichž determinant je roven jedné. Pokud je totiž absolutní hodnota determinantu matice obsahující pouze celá čísla rovna jedné, inverzní matice k této matici obsahuje také pouze celá čísla. Příklady, kdy nelze vytvořit inverzní matici, nejsou používány.

Pro vytvoření matice s determinantem rovným jedné je použit následující algoritmus. Nejprve je vygenerována libovolná trojúhelníková matice, která má na hlavní diagonále samé jedničky. Poté jsou prováděny řádkové elementární úpravy, které nemění hodnotu determinantu. Po provedení dostatečného počtu těchto operací (v aplikaci nastaveno 100 operací) se začnou provádět tyto operace cíleně tak, aby se maximální i minimální hodnoty dostaly do požadovaného rozsahu.

Před schválením příkladu je opět zkoumána maximální hodnota dosažená v průběhu řešení. Podmínka se stejná jako v případě soustav rovnic, tedy $\sqrt{max^n}$, kde n je řád matice a max je maximální velikost generovaných hodnot.

4.5.3 Generování příkladů pro určování lineárních kombinací vektorů

Příklady tohoto typu jsou generovány třídou *CombinationFactory*. Pro generování příkladů, kdy vektor je lineární kombinací ostatních, se využívá metoda *generateGoodExercise* ze třídy *GEMFactory* (viz podkapitola 4.5.1). V opačném případě je využita metoda *generateSingularMatrix* třídy *Generator*. Tyto příklady se generují v poměru 1:1. Také zde platí podmínka maximální velikosti hodnoty využité při řešení příkladu, která je určena výrazem $\sqrt{max^n}$, kde n je počet složek jednotlivých vektorů a max je maximální velikost generovaných hodnot.

4.5.4 Generování příkladů pro určování lineární závislosti vektorů

Tvorbu těchto příkladů obstarává třída *DependencyFactory*. V 50 % příkladů jsou vektory lineárně závislé. Z toho vyplývá, že v 50 % příkladů jsou vektory lineárně nezávislé. Oba tyto typy příkladů jsou generovány pomocí třídy *Generator*. V případě, že jsou vektory lineárně závislé, je použita metoda *generateSingularMatrix*. V opačném případě je to metoda *generateRegularMatrix*. Opět je zde omezena maximální hodnota dosažená v průběhu řešení výrazem $\sqrt{max^n}$, kde n je počet vektorů a max je maximální velikost generovaných hodnot.

4.5.5 Generování příkladů pro výpočet determinantu pomocí Sarrusovy metody.

O generování těchto příkladů se stará třída *SarrusFormatter*. Tato třída generuje regulární matice třetího řádu. K tomu využívá metodu *generateRegularMatrix* ze třídy *Generator*. U těchto příkladů není omezena maximální hodnota, která se může vyskytnout v průběhu výpočtu determinantu, ale je omezena velikost výsledného determinantu výrazem $\frac{3^{max}}{3 \cdot max}$, kde max je maximální velikost generovaných hodnot.

4.5.6 Generování příkladů pro výpočet determinantu pomocí Gaussovy eliminace

Příklady na toto téma jsou generovány třídou *DeterminantFactory*. Pro generování příkladů je využita metoda *generateRegularMatrix* ze třídy *Generator*. U příkladů je omezena absolutní hodnota determinantu matice výrazem $\frac{n^{max}}{n \cdot max}$, kde n je řád matice a max je maximální velikost generovaných hodnot.

4.5.7 Generování příkladů pro určování vlastních čísel a vektorů matice

Pro generování vhodných příkladů, což znamená celočíselné matice s celočíselnými vlastními čísly, se využívá třída *EigenFactory*. Nalezení takovýchto matic ovšem není jednoduché. Pravděpodobnost, že náhodná celočíselná matice bude mít alespoň jedno celočíselné vlastní číslo, je velmi malá. Postup pro generování vhodných příkladů však existuje.

Nechť P je libovolná regulární matice. To znamená, že $det\ (P) \neq 0$. D je diagonální matice stejného řádu jako matice P, jejíž nenulové prvky se rovnají libovolným násobkům $det\ (P)$. Lze dokázat, že matice $A = PDP^{-1}$ je celočíselná matice, jejíž vlastní čísla jsou taktéž celočíselná a rovnají se nenulovým prvkům matice D.

Tuto techniku generování příkladů lze ještě zobecnit. Nechť D_n je libovolný nenulový prvek diagonální matice $D\ a\ n$ je libovolné celé číslo. Pokud pro všechna D_n platí, že

$D_n \ mod \ det\ (P) = n$, potom matice $A = PDP^{-1}$ je celočíselná matice, jejíž vlastní čísla jsou taktéž celočíselná a rovnají se nenulovým prvkům matice D. [18]

Pro generování příkladů v aplikaci je použit algoritmus, který využívá tuto zobecněnou techniku.

```
double[] eigenValues = new double[matrixSize];
double eigenValuesMultiple;
SimpleMatrix result;
do {
    SimpleMatrix randomMatrix;
    do {
        randomMatrix = Generator.generateRegularMatrix(matrixSize, -5, 5);
    }while (Math.abs(randomMatrix.determinant()) > 5);
    double determinant = randomMatrix.determinant();

    eigenValuesMultiple = 1;
    int multiple = random.nextInt((int) Math.abs(determinant));
    for (int i = 0; i < matrixSize; i++) {
        do {
            eigenValues[i] = random.nextInt(2*max)-max;
        }while (eigenValues[i]%determinant != multiple);
                    eigenValuesMultiple *= eigenValues[i];
    }
    SimpleMatrix diagonal = SimpleMatrix.diag(eigenValues);
    SimpleMatrix inverse = randomMatrix.invert();
    result = randomMatrix.mult(diagonal);
    result = result.mult(inverse);
}while (result.elementMaxAbs() > max || !duplicate(eigenValues, matrixSize-
            2) || eigenValuesMultiple > Math.max(20, Math.pow(matrixSize,
            matrixSize)));
```

Algoritmus nejprve vygeneruje pomocí metody *generateRegularMatrix* regulární matici, jejíž determinant je nenulové číslo v rozsahu od -5 do 5. Poté je vygenerováno náhodné celé číslo v rozsahu od 0 do velikosti absolutní hodnoty determinantu vygenerované matice. Poté jsou vygenerována vlastní čísla výsledné matice. Tato čísla splňují výše uvedené podmínky. Z těchto čísel je vytvořena diagonální matice. Dále je vypočtena inverzní matice původně vygenerované matice. Výsledná matice vznikne vhodným vynásobením dříve vytvořených matic.

Dále je potřeba ověřit, že vygenerovaný příklad splňuje podmínky vhodného výukového příkladu. Tyto podmínky jsou tři. První podmínka je, že čísla v takto vygenerované matici nepřesahují rozsah hodnot požadovaný uživatelem. Dále musí mít vytvořená matice alespoň dvě různá vlastní čísla. Poslední podmínka udává, že absolutní člen charakteristického polynomu matice nebude větší než hodnota n^n, kde n je řád matice. V případě, že je vygenerována matice druhého řádu, je tato hodnota nahrazena číslem 20.

ZÁVĚR

Cílem této bakalářské práce bylo vytvoření aplikace pro podporu výuky lineární algebry. Cíl se podařilo splnit, výsledná aplikace obsahuje sedm témat, ze kterých lze generovat vhodné příklady k procvičování a umožňuje simulovat jejich výpočet lidským řešitelem i s popisem jednotlivých kroků řešení.

Aplikaci je možné v budoucnu vylepšit přidáním dalších témat nebo přidáním možnosti exportu vygenerovaných příkladů pro tisk. Vhodným rozšířením by také mohlo být přidání teorie a obecného popisu řešení příkladů ke každému tématu. Uživatelé této aplikace by tak měli veškeré potřebné informace ke studiu na jednom místě.

POUŽITÁ LITERATURA

[1] BEČVÁŘ, Jindřich. *Lineární algebra*. Vyd. 3. Praha: Matfyzpress, 2005. ISBN 80-86732-57-6.

[2] HRACH, Karel. *Matematika pro ekonomy*. Praha: Vysoká škola ekonomie a managementu, 2007. ISBN 978-80-86730-09-7.

[3] HAVRLANT, Lukáš, Matice, *Matematika.cz* [online]. 2014 [cit. 21. 7. 2020]. Dostupné z: https://matematika.cz/matice

[4] HAVRLANT, Lukáš, Tělesa, *Matematika.cz* [online]. 2014 [cit. 22. 7. 2020]. Dostupné z: https://matematika.cz/telesa

[5] HORÁK, Pavel, *Lineární algebra a geometrie* 1. Učební text Přírodovědecké fakulty Masarykovy univerzity [online], Brno 2017 [cit. 22. 7. 2020]. Dostupné z: https://www.math.muni.cz/~janyska/LA_CELE_2017.pdf

[6] TŮMA, Jiří, *Lineární algebra 1: Gaussova eliminace*, Učební text Matematicko-fyzikální fakulty Univerzity Karlovy. [online] Praha 22. 10. 2002 [cit. 24. 7. 2020]. Dostupné z: http://www.karlin.mff.cuni.cz/~tuma/2002/NLinalg2.pdf

[7] HAŠEK, Roman, *Lineární algebra a geometrie*, Učební text Pedagogické fakulty Jihočeské univerzity v Českých Budějovicích. [online] České Budějovice 21. 6. 2020 [cit. 28. 7. 2020]. Dostupné z: http://home.pf.jcu.cz/~hasek/LA2/Linearni_algebra_a_geometrie_studijni_text_2020.pdf

[8] BARTO Libor a TŮMA Jiří, *Lineární Algebra*, Učební text Matematicko-fyzikální fakulty Univerzity Karlovy. [online] Praha 13. 1. 2015 [cit. 28. 7. 2020]. Dostupné z: http://msekce.karlin.mff.cuni.cz/~sir/la/LinAlg/skripta.pdf

[9] OLŠÁK, Petr, *Lineární Algebra*, Učební text Elektrotechnické fakulty ČVUT. [online] Praha 31. 7. 2007 [cit. 30. 7. 2020]. Dostupné z: http://petr.olsak.net/ftp/olsak/linal/linal.pdf

[10] KOVÁŘ, Petr, *Algebra,* Učební text VŠB – Technické univerzity Ostrava. [online] Krmelín 19. 3. 2020 [cit. 30. 7. 2020]. Dostupné z: http://homel.vsb.cz/~kov16/files/skriptum_algebra_rozsirene.pdf

[11] *Wikipedie: Otevřená encyklopedie: Sarrusovo pravidlo* [online]. c2019 [cit. 6. 08. 2020]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Sarrusovo_pravidlo&oldid=17560622

[12] *Oracle Corporation, Java Platform Standard Edition 8 Documentation* [online]. 2014 [cit. 17. 8. 2020]. Dostupné z: https://docs.oracle.com/javase/8/docs/

[13] *Oracle Corporation, JavaFX: Getting Started with JavaFX* [online]. 2014 [cit. 17. 8. 2020]. Dostupné z: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

[14] ABELES, Petr, *EJML* [online]. 2014, 6. 4. 2020 [cit. 17. 8. 2020]. Dostupné z: http://ejml.org/wiki/index.php?title=Main_Page

[15] *JLaTeXMath,* [online]. 2009, 10. 4. 2019 [cit. 17. 8. 2020]. Dostupné z: https://github.com/opencollab/jlatexmath

[16] MAKARENKO, Egor, *LatexView* [online]. 28. 1. 2019, 24. 6. 2019 [cit. 17. 8. 2020]. Dostupné z: https://github.com/egormkn/latexview

[17] KOVAL, Grzegorz, aj. *Launch4j, sourceforge.net* [online]. © 2020, 6. 8. 2019 [cit. 18. 8. 2020]. Dostupné z: https://sourceforge.net/projects/launch4j/

[18] TOWSE, Christopher a CAMPBELL, Eric, Constructing integer matrices with integer eigenvalues, *Applied Probability Trust*, [online]. Claremont, USA, 26. 3. 2016 [cit. 19. 8. 2020]. Dostupné z: https://ericthewry.github.io/pdfs/imies.pdf