Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Program na podporu výuky lineární algebry

2023

Bc. Filip Joska

Vedoucí práce:
doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

Bibliografické údaje

Autor:	Bc. Filip Joska
Název práce:	Program na podporu výuky lineární algebry
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Aplikovaná informatika, Specializace: Vývoj software
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	60
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Bc. Filip Joska
Title:	Learning support program for linear algebra
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Applied Computer Science, Specialization: Software Development
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	60
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Práce se zabývá návrhem a implementací programu na podporu výuky lineární algebry ve formě webové aplikace. Aplikace umí simulovat chod výpočtu základních maticových operací, určit vlastnosti matic (hodnost, determinant) a vypočítat inverzní matici (pokud existuje). Umí počítat soustavy lineárních rovnic a demonstrovat výpočty související s vektorovými prostory, jako je lineární (ne)závislost vektorů, nalezení báze a transformace souřadnic od báze k bázi vektorového prostoru. Součástí práce je výukový text o výše zmíněných pojmech, který je zároveň součástí aplikace. Aplikace také nabízí sekci procvičování, ve které jsou uživateli náhodně generovány příklady k řešení.

Synopsis

The thesis is about the design and implementation of a learning support program for linear algebra in the form of a web application. The application can simulate the computation of basic matrix operations, determine matrix properties (rank, determinant) and compute the inverse matrix (if it exists). It can solve systems of linear equations and demonstrate calculations related to vector spaces, such as linear (in)dependence of vectors, finding the basis and transforming coordinates from the basis to other basis of the vector space. An educational text on the mentioned concepts is also part of the thesis and the application. The application also offers a practice section in which examples are randomly generated for the user to solve.

Klíčová slova: lineární algebra; výuka; výukový program; algebraické výrazy

Keywords: linear algebra; teaching; learning program; algebraic expressions

Děkuji doc. RNDr. Miroslavu Kolaříkovi, Ph.D. za ochotu a věcné připomínky k textu práce i programu. Dále děkuji Mgr. Janu Třískovi, Ph.D. za rady při implementaci výrazů. A v neposlední řadě děkuji rodině a blízkým za podporu.

Odevzdáním tohoto textu jeho autor místopřísežně prohlašuje, že celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Úvod

Při výuce lineární algebry je mnohdy pro pochopení probírané látky vhodné si ji vyzkoušet na několika příkladech. K tomuto účelu byla navržena a implementována webová aplikace na podporu výuky lineární algebry.

Aplikace umožňuje zadávat příklady a následně zobrazovat jednotlivé kroky řešení, procvičovat jednotlivé výpočty (program umí uživateli náhodně generovat příklady a kontrolovat zadané řešení) a také obsahuje výukový text k implementovaným operacím.

Většina podobných programů či aplikací, umožňujících simulovat maticové operace a výpočty související s vektorovými prostory, je zpoplatněná, nebo je minimálně zpoplatněné zobrazení postupu výpočtů. Výjimkou je maticová kalkulačka dostupná na stránce matrixcalc.org, která je zcela zdarma včetně postupu výpočtů a nabízí větší spektrum operací, než aplikace vyvinutá v rámci této diplomové práce.

Hlavní výhodou tohoto programu oproti alternativám je, že je zdarma a open-source. Lze ho snadno rozšiřovat o další funkce nebo i na další platformy. Také nabízí na jednom místě výukový text, procvičování i řešení uživatelem zadaných příkladů.

1 Lineární algebra

V této kapitole jsou probrána vybraná témata lineární algebry. Text této kapitoly je obsažen ve výukovém programu.

V následujícím textu předpokládám znalost pojmů: okruh, komutativní okruh, invertibilní prvek okruhu, grupa, abelovská grupa, permutace a znaménko permutace.

1.1 Matice

Definice 1 (Matice)

Nechť X je neprázdná množina a $m, n \in \mathbb{N}$. Potom obdélníkové schéma

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

kde $a_{ij} \in X$ pro každé $i = 1, \ldots, n$ a každé $j = 1, \ldots, m$, se nazývá matice typu $n \times m$ nad množinou X, značeno též (a_{ij}). [1]

Užíváme termíny řádek matice a sloupec matice. Matice typu $n \times m$ má tedy n řádků a m sloupců.

Je-li $r = \min\{m, n\}$, pak prvky $a_{11}, a_{22}, \ldots, a_{rr}$ tvoří hlavní diagonálu a prvky $a_{1,n}, a_{2,n-1}, \ldots, a_{r,n-(r-1)}$ tvoří vedlejší diagonálu matice A. [2]

Definice 2 (Typy matic)

Definujme následující typy matic [1, 2]:

- Čtvercová matice stupně n je matice A typu $n \times n$, říkáme, že je řádu n.
- Diagonální matice je čtvercová matice, kde jsou všechny prvky, které neleží na hlavní diagonále, rovny 0.
- Skalární matice je diagonální matice, kde jsou si všechny prvky na hlavní diagonále rovny.
- Jednotková matice stupně n (značena E_n) je skalární matice, kde jsou všechny prvky na hlavní diagonále rovny 1.
- Horní trojúhelníková matice je matice, kde jsou všechny prvky pod hlavní diagonálou rovny 0.
- Dolní trojúhelníková matice je matice, kde jsou všechny prvky nad hlavní diagonálou rovny 0.

Definice 3 (Transponovaná matice)
Nechť $A = (a_{ij})$ je matice typu $n \times m$, pak maticí transponovanou k matici A nazýváme matici $A^T = (a_{ji})$ typu $m \times n$, která vznikne z matice A překlopením matice A podle hlavní diagonály, resp. záměnou řádků a sloupců. [1, 2]

1.1.1 Operace s maticemi

Důležitou roli hrají matice nad tělesy či okruhy. Pro takové matice definujeme sčítání a násobení. [1]

Dále budeme slovem okruh a symbolem $\mathcal{R}$ značit komutativní okruh $\mathcal{R} = (R, +, \cdot)$. Pro jednoduchost uvažujme matice nad okruhem $\mathcal{R}$.

Definice 4 (Sčítání matic)
Nechť $A = (a_{ij})$ a $B = (b_{ij})$ jsou matice typu $n \times m$ nad okruhem $\mathcal{R}$. Součtem matic A a B rozumíme matici

$$A + B = (a_{ij} + b_{ij})$$

typu $n \times m$, pro každé $i = 1, \ldots, n$, $j = 1, \ldots, m$. Říkáme, že sčítáme po složkách. [1]

Součet $A + B$ matic A a B je definován právě když mají matice A a B stejný typ. Součet $A + B$ má následně stejný typ jako matice A, B. [1]

Definice 5 (Násobení matic)
Nechť $A = (a_{is})$ je matice typu $n \times m$ a $B = (b_{sj})$ matice typu $m \times k$ nad okruhem $\mathcal{R}$. Součinem matice A a B rozumíme matici

$$A \cdot B = \left(\sum_{s=1}^{m} a_{is} b_{sj} \right)$$

typu $n \times k$, která má na pozici ij součet součinů odpovídajících prvků i-tého řádku matice A a j-tého sloupce matice B. [1]

Součin $A \cdot B$ je definován právě když se počet sloupců matice A rovná počtu řádků matice B. Součin $A \cdot B$ má pak stejný počet řádků jako matice A a stejný počet sloupců jako matice B. [1]

Věta 6 (Vlastnosti sčítání a násobení matic)
Platí [1]:

1. *Sčítání matic je asociativní a komutativní.*
2. *Násobení matic je asociativní.*
3. *Násobení matic není komutativní.*
4. *Násobení matic je distributivní vzhledem ke sčítání.*

Nechť M je množina všech matic nad okruhem $\mathcal{R}$. Sčítání ani násobení matic není binární operací na množině M, protože není definován součet ani součin libovolných dvou matic množiny M. Sčítání a násobení na množině M jsou tzv. parciální operace. [1]

Poznámka 7

Chceme-li sčítání a násobení matic uvažovat jako binární operace, musíme zvolit vhodnou podmnožinu M (například množinu $R^{n\times n}$ všech čtvercových matic řádu n). [1]

Definice 8 (Násobení matic skaláry)

Nechť $A = (a_{ij})$ je matice typu $n \times m$ nad okruhem $\mathcal{R}$ a nechť $c \in R$ je libovolný prvek. Pro násobení tzv. skalárem c platí

$$c \cdot A = (c \cdot a_{ij}),$$

pro každé $i = 1, \ldots, n$, $j = 1, \ldots, m$. [1]

1.1.2 Determinant matice

Uvažujme matice nad komutativním okruhem $\mathcal{R}$.

Definice 9 (Determinant matice)

Nechť $A = (a_{ij})$ je čtvercová matice řádu n nad okruhem $\mathcal{R}$. Determinant $\det A$ matice A definujeme

$$\det A = \sum_{P\in\mathbb{S}_n} \operatorname{sgn} P \cdot a_{1P(1)} \cdot a_{2P(2)} \cdot \ldots \cdot a_{nP(n)},$$

kde $\mathbb{S}_n$ je symetrická grupa stupně n, $\operatorname{sgn} P$ znaménko permutace P. [1]

Zapisujeme také pomocí schématu

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

Determinant matice je součet $n!$ součinů, ve kterých je z každého řádku a každého sloupce matice A právě jeden prvek, přičemž je každý ze součinů vynásoben znaménkem permutace ($\operatorname{sgn} P$) určeným řádkovými a sloupcovými indexy prvků součinu. Sčítá se přes všechny permutace

$$P = \begin{pmatrix} 1 & 2 & \ldots & n \\ P(1) & P(2) & \ldots & P(n) \end{pmatrix}$$

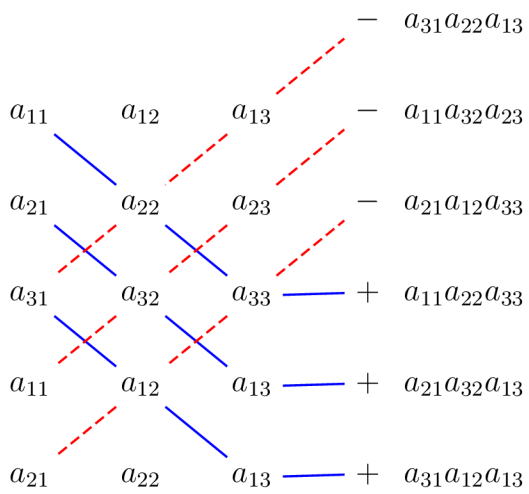
množiny $\{1, 2, \ldots, n\}$. [1, 2]

Pro matice prvního a druhého řádu platí

$$\left|a_{11}\right| = a_{11},$$

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}.$$

Determinant matice třetího řádu lze vypočítat podle tzv. Sarrusova pravidla. Pod determinant opíšeme jeho první dva řádky a vynásobíme diagonálně prvky podle obrázku 1. Výsledné součiny sečteme s odpovídajícími znaménky. [1]



Obrázek 1: Ukázka Sarrusova pravidla

Ve výsledku tedy platí

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

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

Sarrusovo pravidlo ale pro determinanty čtvercových matic stupňů vyšších než tři nemá tak přehlednou analogii. [2]

Věta 10

Pro každou čtvercovou matici $A = (a_{ij})$ nad okruhem $\mathcal{R}$ platí

$$detA^T = detA. \text{ [2]}$$

Věta 10 je významná pro práci s determinanty. Uvedeme-li tvrzení o determinantech matic taková, že se v jejich formulaci používají řádky matic, pak platí analogické tvrzení, ve kterých se používají sloupce matic. [2]

Věta 11

Má-li čtvercová matice A v některém řádku (sloupci) samé nuly, je determinant matice A roven nule.

Věta 12

Pokud má matice A všechny prvky pod (nebo nad) hlavní diagonálou nulové, je determinant matice A roven součinu prvků na hlavní diagonále.

Věta 13

Vznikne-li matice B ze čtvercové matice A záměnou i-tého a j-tého řádku, kde $i \neq j$, potom $detB = -detA$. [2]

Věta 14

Vznikne-li matice B z matice A provedením některé permutace P na řádky matice A, pak

$$detB = sgnP \cdot detA. \text{ [2]}$$

Věta 15

Jestliže se v matici A rovnají i-tý a j-tý řádek ($i \neq j$), pak $detA = 0$. [2]

Věta 16

Přičteme-li k nějakému sloupci (řádku) matice A c-násobek nějakého jiného sloupce (řádku), kde $c \in R$, je determinant vzniklé matice roven determinantu matice A. [1]

Přičteme-li k nějakému sloupci (řádku) matice A lineární kombinaci ostatních sloupců (řádků) matice A s koeficienty $c_i \in R$, je determinant vzniklé matice roven determinantu matice A. [1]

Definice 17 (Regulární matice)

Matice A nad tělesem $\mathcal{T} = (T, +, \cdot)$ se nazývá singulární právě tehdy, je-li její determinant roven nule a regulární právě tehdy, když je její determinant nenulový. [1]

Definice 18 (Dílčí matice a subdeterminant)

Nechť $A = (a_{ij})$ je matice typu $n \times m$. Pak každou matici, která vznikne z matice A vynecháním některých řádků a některých sloupců, nazýváme dílčí maticí matice A. [2]

Je-li dílčí matice matice A čtvercová, potom její determinant nazýváme subdeterminantem matice A. [2]

Definice 19 (Minor matice)

Nechť $A = (a_{ij})$ je matice typu $n \times n$, pak subdeterminant dílčí matice A_{ij} stupně $n - 1$ vzniklé vynecháním i-tého řádku a j-tého sloupce A nazýváme minor matice A příslušný k prvku a_{ij} a značíme jej $\mathcal{M}_{ij}$. [2]

Algebraickým doplňkem prvku a_{ij} nazýváme prvek $\mathcal{A}_{ij} = (-1)^{i+j}\mathcal{M}_{ij}$. [2]

Věta 20 (Laplaceova věta o rozvoji determinantu)
Nechť $A = (a_{ij})$ je matice řádu $n > 1$ nad okruhem $\mathcal{R}$. Pro každé $j = 1, \ldots, n$ je

$$detA = \sum_{i=1}^{n} a_{ij}\mathcal{A}_{ij},$$

kde $\mathcal{A}_{ij}$ je algebraický doplněk prvku a_{ij}. [1]

Věta 21 (Věta o násobení determinantů)
Nechť A, B jsou čtvercové matice téhož řádu nad okruhem $\mathcal{R}$. Pak platí

$$det(A \cdot B) = detA \cdot detB. \text{ [1]}$$

Pomocí uvedených vlastností determinantů lze vypočítat determinanty stupně n.

Příklad 22 (Determinant řádu 5)

$$\begin{vmatrix} 3 & -2 & 6 & -3 & 1 \\ 2 & 1 & 1 & 7 & -1 \\ 3 & -4 & 6 & -3 & 1 \\ 0 & 5 & 2 & 1 & 3 \\ 4 & -2 & 3 & 7 & -1 \end{vmatrix} = \begin{vmatrix} 0 & 2 & 0 & 0 & 0 \\ 2 & 1 & 1 & 7 & -1 \\ 3 & -4 & 6 & -3 & 1 \\ 0 & 5 & 2 & 1 & 3 \\ 4 & -2 & 3 & 7 & -1 \end{vmatrix} = (-1)^{1+2} \cdot 2 \cdot \begin{vmatrix} 2 & 1 & 7 & -1 \\ 3 & 6 & -3 & 1 \\ 0 & 2 & 1 & 3 \\ 4 & 3 & 7 & -1 \end{vmatrix}$$

$$= -2 \cdot \begin{vmatrix} 2 & 1 & 7 & -1 \\ 3 & 6 & -3 & 1 \\ 0 & 2 & 1 & 3 \\ 2 & 2 & 0 & 0 \end{vmatrix} = -2 \cdot \begin{vmatrix} 2 & -1 & 7 & -1 \\ 3 & 3 & -3 & 1 \\ 0 & 2 & 1 & 3 \\ 2 & 0 & 0 & 0 \end{vmatrix} = -2 \cdot (-1)^{4+1} \cdot 2 \cdot \begin{vmatrix} -1 & 7 & -1 \\ 3 & -3 & 1 \\ 2 & 1 & 3 \end{vmatrix}$$

$$= 4 \cdot ((-1) \cdot (-3) \cdot 3 + 3 \cdot 1 \cdot (-1) + 2 \cdot 7 \cdot 1 - (-1) \cdot (-3) \cdot 2 - 1 \cdot 1 \cdot (-1) - 3 \cdot 7 \cdot 3)$$

$$= 4 \cdot (-48) = -192$$

Postup řešení:

1. od 1. řádku odečteme 3. řádek (podle věty 16),
2. provedeme Laplaceův rozvoj podle prvků 1. řádku (podle věty 20),
3. od 4. řádku odečteme 1. řádek,
4. od 2. sloupce odečteme 1. sloupec,
5. provedeme Laplaceův rozvoj podle 4. řádku,
6. použijeme Sarrusovo pravidlo pro determinant stupně 3.

1.1.3 Inverzní matice

Definice 23 (Inverzní matice)
Nechť $\mathcal{R}$ je okruh s jednotkovým prvkem a A je čtvercová matice řádu n nad $\mathcal{R}$. Inverzní maticí k matici A rozumíme matici A^{-1}, pro kterou platí

$$A \cdot A^{-1} = A^{-1} \cdot A = E_n. \text{ [1]}$$

Matice A, ke které inverzní matice existuje, se nazývá invertibilní a také regulární.

Důsledek 24
Matice A je nad tělesem invertibilní právě když je determinant matice A nenulový. [1, 2]

Věta 25
Nechť $\mathcal{R}$ je okruh s jednotkovým prvkem a A, B invertibilní matice téhož řádu nad $\mathcal{R}$. Pak je matice $A \cdot B$ také invertibilní a platí

$$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}. \text{ [1]}$$

Věta 26
Jestliže je A čtvercová invertibilní matice nad okruhem $\mathcal{R}$, pak je matice A^T rovněž invertibilní a platí

$$(A^T)^{-1} = (A^{-1})^T. \text{ [1]}$$

Definice 27 (Reciproká matice)
Nechť $A = (a_{ij})$ je čtvercová matice řádu $n > 1$ nad okruhem $\mathcal{R}$. Reciprokou maticí k matici A rozumíme matici A_{rec} řádu n, ve které na pozici ij stojí algebraický doplněk prvku a_{ji} matice A, tj. prvek $\mathcal{A}_{ji} = (-1)^{i+j} \det A_{ji}$. [1]

Věta 28
Nechť $\mathcal{R}$ je okruh s jednotkovým prvkem a A čtvercová matice nad okruhem $\mathcal{R}$. Matice A je invertibilní právě když je její determinant $detA$ invertibilním prvkem okruhu $\mathcal{R}$. Nastane-li tento případ, platí

$$A^{-1} = (detA)^{-1} \cdot A_{rec}. \text{ [1]}$$

Mějme čtvercovou matici A řádu $n > 1$ nad tělesem. Inverzní matici k matici A lze vypočítat pomocí věty 28. Platí tedy

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{21} & \cdots & \mathcal{A}_{n1} \\ \mathcal{A}_{12} & \mathcal{A}_{22} & \cdots & \mathcal{A}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{A}_{1n} & \mathcal{A}_{2n} & \cdots & \mathcal{A}_{nn} \end{pmatrix}. \text{ [3]}$$

Příklad 29 (Výpočet inverzní matice pomocí reciproké matice)
Určete inverzní matici k matici

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

Podle Sarrusova pravidla: $\det A = 6$. Matice je tedy invertibilní (podle důsledku 24). Dále vypočítáme potřebné algebraické doplňky.

$$\mathcal{A}_{11} = (-1)^{1+1} \cdot \begin{vmatrix} 4 & 1 \\ 2 & 5 \end{vmatrix} = 18, \; \mathcal{A}_{21} = (-1)^{2+1} \cdot \begin{vmatrix} 2 & 0 \\ 2 & 5 \end{vmatrix} = -10,$$

$$\mathcal{A}_{31} = (-1)^{3+1} \cdot \begin{vmatrix} 2 & 0 \\ 4 & 1 \end{vmatrix} = 2, \; \mathcal{A}_{12} = (-1)^{1+2} \cdot \begin{vmatrix} 5 & 1 \\ 1 & 5 \end{vmatrix} = -24,$$

$$\mathcal{A}_{22} = (-1)^{2+2} \cdot \begin{vmatrix} 3 & 0 \\ 1 & 5 \end{vmatrix} = 15, \; \mathcal{A}_{32} = (-1)^{3+2} \cdot \begin{vmatrix} 3 & 0 \\ 5 & 1 \end{vmatrix} = -3,$$

$$\mathcal{A}_{13} = (-1)^{1+3} \cdot \begin{vmatrix} 5 & 4 \\ 1 & 2 \end{vmatrix} = 6, \; \mathcal{A}_{23} = (-1)^{2+3} \cdot \begin{vmatrix} 3 & 2 \\ 1 & 2 \end{vmatrix} = -4,$$

$$\mathcal{A}_{33} = (-1)^{3+3} \cdot \begin{vmatrix} 3 & 2 \\ 5 & 4 \end{vmatrix} = 2,$$

Nakonec podle věty 28 určíme inverzní matici:

$$A^{-1} = \frac{1}{6} \begin{pmatrix} 18 & -10 & 2 \\ -24 & 15 & -3 \\ 6 & -4 & 2 \end{pmatrix} = \begin{pmatrix} 3 & \frac{-5}{3} & \frac{1}{3} \\ -4 & \frac{5}{2} & \frac{-1}{2} \\ 1 & \frac{-2}{3} & \frac{1}{3} \end{pmatrix}$$

Definice 30 (Elementární řádkové transformace)
Nechť A je matice nad tělesem $\mathcal{T} = (T, +, \cdot)$. Elementárními řádkovými transformacemi matice A nazýváme operace [3]:

1. výměnu libovolných dvou řádků v matici A,
2. vynásobení některého řádku v matici A skalárem $c \in T, c \neq 0$,
3. přičtení libovolného násobku některého řádku z A k libovolnému řádku matice A.

Nyní uvažujme množinu $M_n(T)$ čtvercových matic n-tého stupně nad tělesem $\mathcal{T} = (T, +, \cdot)$.

Definice 31
Nechť je dána libovolná elementární řádková transformace čtvercové matice n-tého stupně. Pak maticí této elementární transformace rozumíme matici, která vznikne z jednotkové matice n-tého stupně použitím této transformace. [2]

Věta 32

Je-li $A \in M_n(T)$, pak jsou si řádkově ekvivalentní matice, které dostaneme z matice A

- *provedením dané elementární řádkové transformace na A,*
- *vynásobením matice A maticí této transformace zleva. [2]*

Věta 33

Každá regulární matice je součinem elementárních transformačních matic. [1]

Věta 34

Je-li $A \in M_n(T)$ regulární, pak je možno pomocí elementárních řádkových transformací přejít od matice A k matici E. Zároveň pomocí stejných transformací přejdeme od matice E k matici A^{-1}. [2]

Při výpočtu inverzní matice sestavíme matici $(A|E)$ typu $n \times 2n$, kterou převedeme řádkovými elementárními transformacemi na matici $(E|B)$. Matice B je součinem elementárních transformačních matic, které odpovídají použitým řádkovým úpravám. Víme tedy, že $B = A^{-1}$. [1]

Příklad 35 (Výpočet inverzní matice pomocí elementárních transformací)

Určete inverzní matici k matici

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 3 & 2 & 5 \\ 2 & 2 & 2 \end{pmatrix}.$$

Podle Sarrusova pravidla: $\det A = 4$, matice je invertibilní. Následně sestavíme matici $(A|E)$:

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

Od druhého řádku odečteme trojnásobek prvního a od třetího řádku odečteme dvojnásobek prvního. Dostaneme matici:

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

Od druhého řádku odečteme dvojnásobek třetího:

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

Následně k prvnímu řádku přičteme třetí a zaměníme druhý a třetí řádek. Dostaneme matici:

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

Druhý řádek vynásobíme $\frac{-1}{2}$ a třetí řádek vynásobíme $\frac{1}{2}$:

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

Nakonec od prvního řádku odečteme třetí:

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

$$A^{-1} = \begin{pmatrix} -\frac{3}{2} & -\frac{1}{2} & 2 \\ 1 & 0 & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -1 \end{pmatrix}$$

1.2 Vektorové prostory

Definice 36 (Vektorový prostor)
Nechť $(V, +)$ je abelovská grupa a $\mathcal{T} = (T, +, \cdot)$ je komutativní těleso. Nechť je dáno zobrazení $T \times V \to V$ značeno symbolem „$\cdot$“. Nechť dále platí $\forall u, v \in V$, $\forall a, b \in T$:

1. $a \cdot (u + v) = a \cdot u + a \cdot v$,
2. $(a + b) \cdot u = a \cdot u + b \cdot u$,
3. $(a \cdot b) \cdot u = a \cdot (b \cdot u)$,
4. $1 \cdot u = u$.

Pak budeme říkat, že $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor nad tělesem $\mathcal{T}$. Prvky množiny V budeme nazývat vektory, prvky množiny T skaláry. [1, 3]

Definice 37 (Aritmetický vektorový prostor)
Nechť je $\mathcal{T} = (T, +, \cdot)$ těleso a $n \in \mathbb{N}$. Na kartézském součinu $T^n = V$ definujeme operaci $+$ (tzv. po složkách).

Jestliže $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in V$, pak

$$(a_1, \ldots, a_n) + (b_1, \ldots, b_n) = (a_1 + b_1, \ldots, a_n + b_n).$$

Vidíme, že $(V, +)$ je abelovská grupa, $o = (0, \ldots, 0)$ její jednotkový prvek a $(-a_1, \ldots, -a_n)$ její inverzní prvek k prvku $(a_1, \ldots, a_n)$.

Definujme levou vnější operaci pro $c \in T$ a $(a_1, \ldots, a_n) \in V$:

$$c \cdot (a_1, \ldots, a_n) = (c \cdot a_1, \ldots, c \cdot a_n).$$

Vektorový prostor $\mathcal{V} = (V, +, T, \cdot)$ nazýváme aritmetický, značíme $\mathcal{T}^n$. [3]

Definice 38 (Lineární kombinace vektorů)
Nechť je $\mathcal{V} = (V, +, T, \cdot)$ vektorový prostor nad tělesem $\mathcal{T} = (T, +, \cdot)$ a $v, u_1, \ldots, u_k \in V$. Pak řekneme, že vektor v je lineární kombinací vektorů $u_1, \ldots, u_k$ existují-li skaláry $c_1, \ldots, c_k \in T$ takové, že

$$v = \sum_{i=1}^{k} c_i \cdot u_i. \text{ [2, 3]}$$

Definice 39 (Nulový vektor)
Nechť $u_1, \ldots, u_k$ jsou libovolné vektory z vektorového prostoru $\mathcal{V}$ nad tělesem $\mathcal{T}$. Pak jejich triviální nulovou kombinaci

$$o = 0 \cdot u_1 + \ldots + 0 \cdot u_k$$

nazýváme nulový vektor. Každou jejich jinou nulovou lineární kombinaci budeme nazývat netriviální nulová kombinace. [2, 3]

Poznámka 40
Pro libovolné vektory $u_1, \ldots, u_k$ z vektorového prostoru $\mathcal{V}$ nad tělesem $\mathcal{T}$ existuje alespoň jedna jejich (nulová) lineární kombinace. Vždy lze uvažovat jejich nulový vektor. [2, 3]

Definice 41 (Lineární závislost vektorů)
Vektory $u_1, \ldots, u_k$ z vektorového prostoru $\mathcal{V}$ se nazývají lineárně závislé, existuje-li alespoň jedna jejich netriviální nulová kombinace.
V opačném případě se vektory $u_1, \ldots, u_k$ nazývají lineárně nezávislé. [2, 3]

Věta 42
Jsou-li ve vektorovém prostoru $\mathcal{V}$ mezi vektory $u_1, \ldots, u_m \in V$ některé lineárně závislé, pak jsou $u_1, \ldots, u_m$ lineárně závislé. [2, 3]

Důsledek 43
Je-li mezi vektory $u_1, \ldots, u_m$ nulový vektor o, pak jsou $u_1, \ldots, u_m$ lineárně závislé. [3]

Věta 44
Jsou-li vektory $u_1, \ldots, u_m \in V$ lineárně nezávislé a je-li $\{u_{j_i}, \ldots, u_{j_k}\} \subseteq \{u_1, \ldots, u_m\}$, pak jsou lineárně nezávislé také vektory $u_{j_i}, \ldots, u_{j_k}$. [2, 3]

Věta 45

Jsou-li $u_1, \ldots, u_m \in V$ vektory z vektorového prostoru $\mathcal{V} = (V, +, T, \cdot)$, pak jsou tyto vektory lineárně závislé právě když je alespoň jeden z nich lineární kombinací ostatních vektorů. [2, 3]

Příklad 46

Určete, zda jsou v aritmetickém vektorovém prostoru $\mathbb{R}^2$ vektory $u = (2, 3)$ a $v = (1, 7)$ lineárně závislé nebo nezávislé.

Hledáme $c_1, c_2 \in \mathbb{R}$ takové, že $c_1 u + c_2 v = o$, tedy

$$c_1(2, 3) + c_2(1, 7) = (0, 0).$$

Dostáváme soustavu lineárních rovnic

$$\begin{aligned} 2c_1 + c_2 &= 0 \\ 3c_1 + 7c_2 &= 0, \end{aligned}$$

která má jediné řešení: $c_1 = c_2 = 0$, vektory jsou tedy lineárně nezávislé.

Definice 47 (Vektorový podprostor)

Nechť $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor nad tělesem $\mathcal{T} = (T, +, \cdot)$, nechť $W \subseteq V$. Pak $\mathcal{W} = (W, +, T, \cdot)$ nazveme vektorovým podprostorem vektorového prostoru $\mathcal{V}$ právě když platí:

1. $W \neq \emptyset$,
2. $\forall u, v \in W \qquad u + v \in W$,
3. $\forall u \in W, \forall a \in T \qquad a \cdot u \in W$. [1, 3]

Každý vektorový prostor $\mathcal{V}$ je zřejmě podprostorem sám v sobě. Tento podprostor nazýváme nevlastní, všechny ostatní podprostory prostoru $\mathcal{V}$ se nazývají vlastní. [1]

Jednoprvková množina obsahující nulový vektor $o \in V$ je též podprostorem prostoru $\mathcal{V}$, nazývá se nulový nebo triviální podprostor, značí se $\mathcal{O}$ a je nejmenší možný vektorový podprostor. Ostatní podprostory prostoru $\mathcal{V}$ se nazývají nenulové nebo netriviální. [1, 3]

Věta 48

Nechť $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor nad tělesem $\mathcal{T} = (T, +, \cdot)$ a nechť $\emptyset \neq W \subseteq V$. Pak $\mathcal{W} = (W, +, T, \cdot)$ je vektorovým podprostorem $\mathcal{V}$ právě když množina W s každými prvky $u_1, \ldots, u_k$ obsahuje i jejich lineární kombinaci. [3]

Lemma 49

Průnik neprázdné množiny podprostorů vektorového prostoru $\mathcal{V}$ nad tělesem $\mathcal{T}$ je podprostorem prostoru $\mathcal{V}$. [1]

Definice 50 (Lineární obal množiny ve vektorovém prostoru)
Nechť $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor a $M \subseteq V$. Průnik všech podprostorů prostoru $\mathcal{V}$, které množinu M obsahují, nazveme lineární obal množiny M a označíme $L(M)$ (nebo $[M]$). [1, 3]

Věta 51
Nechť $\mathcal{V}$ je vektorový prostor nad tělesem $\mathcal{T}$ a $M \subseteq V$. Lineární obal $L(M)$ množiny M ve $\mathcal{V}$ je roven množině všech lineárních kombinací vektorů z M s koeficienty z tělesa $\mathcal{T}$. [1, 3]

Věta 52
Nechť $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor a $\emptyset \neq M \subseteq V$. Pak $(L(M), +, T, \cdot)$ je právě vektorový podprostor $\mathcal{M} = (L(M), +, T, \cdot)$ generovaný neprázdnou množinou M. [3]

Definice 53 (Množina generátorů vektorového prostoru)
Mějme vektorový prostor $\mathcal{V} = (V, +, T, \cdot)$ a množinu $\emptyset \neq M \subseteq V$. Je-li $L(M) = V$, pak říkáme, že M je množina generátorů prostoru $\mathcal{V}$, respektive že množina M generuje prostor $\mathcal{V}$. [1, 3]
Řekneme, že prostor $\mathcal{V}$ je konečné dimenze (nebo také konečně generovaný), existuje-li konečná množina, která ho generuje. V opačném případě se nazývá nekonečně generovaný. [1, 3]

1.2.1 Hodnost matice

Nechť $A \in M_{m,n}(T)$. Řádky budeme uvažovat jako řádkové vektory z aritmetického vektorového prostoru $\mathcal{T}^n = (T^n, +, T, \cdot)$.

Definice 54 (Řádkový podprostor matice)
Řádkovým podprostorem matice A rozumíme podprostor prostoru $\mathcal{T}^n$ generovaný všemi řádkovými vektory matice A. [3]

Definice 55 (Řádkově ekvivalentní matice)
Matice $A, B \in M_{m,n}(T)$ jsou řádkově ekvivalentní, pokud lze získat B z A pomocí konečného počtu elementárních řádkových operací, zapisujeme $A \sim B$. [3]

Definice 56 (Hodnost matice)
Hodnost matice $A \in M_{m,n}(T)$ je dimenze řádkového podprostoru matice A v aritmetickém vektorovém prostoru $\mathcal{T}^n$, značíme $h(A)$. [2, 3]

Poznámka 57
Platí [2, 3]:

1. hodnost matice A je rovna počtu jejích lineárně nezávislých řádků, tj. $h(A) \leq m$,

2. řádkově ekvivalentní matice mají stejnou hodnost,

3. $h(A)$ se rovná počtu nenulových řádků libovolné matice trojúhelníkového tvaru, která je řádkově ekvivalentní s A.

Věta 58

Hodnost matice A je rovna maximálnímu stupni nenulového subdeterminantu matice A. [2, 3]

Důsledek 59

Hodnost matice A je rovna maximálnímu počtu lineárně nezávislých sloupců matice A. [2, 3]

Lemma 60

Čtvercová matice je regulární, je-li hodnost matice rovna jejímu řádu. [1]

Definice 61 (Odstupňovaná matice)

Matice $A = (a_{ij})$ typu $n \times m$ nad tělesem $\mathcal{T}$ se nazývá odstupňovaná, platí-li pro její prvky:

1. je-li $n > 1$, pak $a_{21} = 0$,

2. pokud pro některé $i \in \{1, \ldots, n-1\}$, $k \in \{1, \ldots, m-1\}$ je

$$a_{i,1} = a_{i,2} = \ldots = a_{i,k} = 0,$$

pak též

$$a_{i+1,1} = a_{i+1,2} = \ldots = a_{i+1,k+1} = 0.$$

V odstupňované matici každý nenulový řádek začíná větším počtem nul, než předcházející. [1]

Definice 62 (Redukovaná matice)

Matice A se nazývá redukovaná, je-li první nenulový prvek každého nenulového řádku v A roven 1, a jestliže v každém sloupci obsahujícím takový prvek některého řádku jsou všechny zbývající prvky rovny 0. [3]

Důsledek 63

Nenulové řádky redukované matice jsou lineárně nezávislé. [3]

Příklad 64

Určete hodnost matice

$$A = \begin{pmatrix} 1 & 2 & 6 \\ 1 & 4 & 2 \\ 3 & -2 & 7 \\ 2 & -6 & 5 \end{pmatrix}.$$

Matici převedeme na redukovaný trojúhelníkový tvar:

$$\begin{pmatrix} 1 & 2 & 6 \\ 1 & 4 & 2 \\ 3 & -2 & 7 \\ 2 & -6 & 5 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 6 \\ 1 & 4 & 2 \\ 3 & -2 & 7 \\ 3 & -2 & 7 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 6 \\ 0 & 2 & -4 \\ 3 & -2 & 7 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 6 \\ 0 & 2 & -4 \\ 0 & -8 & -11 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 6 \\ 0 & 2 & -4 \\ 0 & -8 & -11 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 6 \\ 0 & 2 & -4 \\ 0 & 0 & -27 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Podle důsledků 59 a 63 víme, že počet nenulových řádků redukované matice odpovídá její hodnosti, tudíž

$$h(A) = 3.$$

Hodnost matice lze určit i podle věty 58 pomocí subdeterminantů:

$$\begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} = 2, \quad \begin{vmatrix} 1 & 2 & 6 \\ 1 & 4 & 2 \\ 3 & -2 & 7 \end{vmatrix} = -54.$$

Maximální stupeň nenulového subdeterminantu matice A je 3, tudíž $h(A) = 3$.

1.2.2 Báze

Definice 65 (Báze vektorového prostoru)
Bází vektorového prostoru $\mathcal{V}$ konečné dimenze rozumíme libovolnou lineárně nezávislou množinu $\{u_1, \ldots, u_n\}$ jeho generátorů. [2, 3]

Věta 66
Nechť $M = \{u_1, \ldots, u_n\}$ je báze vektorového prostoru $\mathcal{V} = (V, +, T, \cdot)$. Pak lze každý vektor $v \in V$ vyjádřit právě jedním způsobem jako lineární kombinaci vektorů $u_1, \ldots, u_n$. [2, 3]

Věta 67
Je-li $M = \{u_1, \ldots, u_n\}$ množina generátorů vektorového prostoru $\mathcal{V}$, pak existuje množina $M' \subseteq M$, která je bází $\mathcal{V}$. [2, 3]

Definice 68 (Dimenze vektorového prostoru)
Je-li $\mathcal{V} \neq \mathcal{O}$ vektorový prostor konečné dimenze, pak počet prvků jeho libovolné báze nazýváme dimenze vektorového prostoru $\mathcal{V}$, značíme $\dim\mathcal{V}$. Je-li $\mathcal{V} = \mathcal{O}$, pak $\dim\mathcal{O} = 0$. [2, 3]

Věta 69 (Steinitzova věta o výměně bází)
Nechť $\{u_1, \ldots, u_n\}$ je množina generátorů vektorového prostoru $\mathcal{V} = (V, +, T, \cdot) \neq \mathcal{O}$ a $v_1, \ldots, v_k \in V$ jsou lineárně nezávislé vektory. Pak

$k \leq n$ a při vhodném očíslování vektorů z množiny $\{u_1, \ldots, u_k\}$ je množina $\{v_1, \ldots, v_k, u_{k+1}, \ldots, u_n\}$ množinou generátorů prostoru $\mathcal{V}$. [2, 3]

Důsledek 70

Je-li $\mathcal{V}$ nenulový vektorový prostor konečné dimenze, pak každé dvě jeho báze mají stejný počet prvků. [2, 3]

Důsledek 71

Nechť množina $\{u_1, \ldots, u_n\}$ generuje prostor $\mathcal{V} = ([\{u_1, \ldots, u_n\}], +, T, \cdot)$ a $v_1, \ldots, v_k \in V$. Je-li $k > n$, pak jsou vektory $v_1, \ldots, v_k$ lineárně závislé. [2, 3]

Důsledek 72

Nechť $\mathcal{V} = ([\{u_1, \ldots, u_n\}], +, T, \cdot)$, pak $dim\mathcal{V} \leq n$. [2, 3]

Důsledek 73

Nechť $\mathcal{V}$ je vektorový prostor konečné dimenze $dim\mathcal{V} = n$. Pak každá množina $u_1, \ldots, u_k \in V$ lineárně nezávislých vektorů je obsažena v některé bázi prostoru $\mathcal{V}$. [2, 3]

Věta 74

Nechť $\mathcal{V} = (V, +, T, \cdot)$, $dim\mathcal{V} = n$ a $u_1, \ldots, u_n \in V$. Pak jsou následující podmínky ekvivalentní [2, 3]:

1. *$u_1, \ldots, u_n$ jsou lineárně nezávislé,*
2. *množina $\{u_1, \ldots, u_n\}$ generuje prostor $\mathcal{V}$,*
3. *množina $\{u_1, \ldots, u_n\}$ je báze prostoru $\mathcal{V}$.*

Příklad 75

Je dán vektorový prostor $\mathcal{W} = (L(M), +, T, \cdot)$ generovaný množinou

$$M = \{(3, 2, 1, 7), (-6, 4, 0, 1), (12, 8, 4, 28), (3, 10, 3, 22)\}.$$

Nalezněte bázi vektorového prostoru $\mathcal{W}$.

Vektory generující $\mathcal{W}$ zapíšeme do matice, kterou následně převedeme na redukovaný trojúhelníkový tvar:

$$\begin{pmatrix} 3 & 2 & 1 & 7 \\ -6 & 4 & 0 & 1 \\ 12 & 8 & 4 & 28 \\ 3 & 10 & 3 & 22 \end{pmatrix} \sim \begin{pmatrix} 3 & 2 & 1 & 7 \\ 0 & 8 & 2 & 15 \\ 12 & 8 & 4 & 28 \\ 0 & 8 & 2 & 15 \end{pmatrix} \sim \begin{pmatrix} 3 & 2 & 1 & 7 \\ 0 & 8 & 2 & 15 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Podle důsledku 63 víme, že jsou nenulové řádky matice lineárně nezávislé a tvoří tedy bázi vektorového prostoru $\mathcal{W}$. Báze $B = \{(3, 2, 1, 7), (0, 8, 2, 15)\}$.

1.3 Soustavy lineárních rovnic

Definice 76 (Soustava lineárních rovnic)

Soustavou n lineárních rovnic o m neznámých nad tělesem $\mathcal{T} = (T, +, \cdot)$ budeme rozumět rovnice tvaru

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

kde koeficienty $a_{11}, \ldots, a_{nm} \in T$ a $y_1, \ldots, y_n \in T$.

Každá uspořádaná m-tice $x = (x_1, \ldots, x_m) \in T^m$, pro kterou platí všech n výše uvedených vztahů, se nazývá řešení soustavy. Pokud $y_1 = y_2 = \cdots = y_n = 0$, soustava se nazývá homogenní, jinak se nazývá nehomogenní. [1, 3]

Definice 77

Nechť je dána soustava lineárních rovnic. Pak matici

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

nazýváme matice soustavy,

$$y^T = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

nazýváme sloupec pravých stran a matici

$$(A|y^T) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1m} & y_1 \\ a_{21} & a_{22} & \cdots & a_{2m} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} & y_n \end{array}\right)$$

rozšířená matice soustavy. [1, 3]

Poznámka 78

Soustavy lineárních rovnic se často zapisují ve tvaru

$$\sum_{j=1}^{m} a_{ij}x_j = y_i, \quad i = 1, \ldots, n,$$

nebo v tzv. maticovém tvaru

$$A \cdot x = y. \text{ [1, 3]}$$

Definice 79

Soustava lineárních rovnic $Ax = y$ se nazývá řešitelná, existuje-li alespoň jedno řešení. V opačném případě se nazývá neřešitelná. Dvě soustavy $Ax = y$ a $Bx' = y'$ se nazývají ekvivalentní, mají-li stejné množiny řešení. [1, 3]

Věta 80

Nehomogenní soustava lineárních rovnic $Ax = y$ je řešitelná právě když je vektor y lineární kombinací sloupců matice A. [3]

Věta 81 (Frobeniova)

Nehomogenní soustava lineárních rovnic $Ax = y$ je řešitelná právě když je $h(A) = h((A|y^T))$. [1, 3]

1.3.1 Gaussova eliminační metoda

Věta 82

Nechť $Ax = y$, $Bx = z$ jsou nehomogenní soustavy lineárních rovnic o m neznámých nad tělesem $\mathcal{T}$. Jsou-li $(A|y^T)$ a $(B|z^T)$ řádkově ekvivalentní, pak jsou tyto soustavy ekvivalentní. [3]

Věta 83 (Doplněná věta Frobeniova)

Nechť $Ax = y$ je soustava n rovnic o m neznámých nad tělesem $\mathcal{T}$. Tato soustava má řešení, právě když $h(A) = h((A|y^T))$. Je-li $h(A) = m$, soustava má jediné řešení. Je-li $h(A) = k < m$, soustava má nekonečně mnoho řešení závislých na $m - k$ parametrech. [1, 3]

Větu 82 využívá Gaussova eliminační metoda, která umožňuje obecně řešit libovolnou nehomogenní soustavu lineárních rovnic. Při výpočtu se postupně rozšířená matice soustavy pomocí elementárních řádkových transformací upravuje na redukovaný trojúhelníkový tvar. [1, 3]

Nechť $Ax = y$ je soustava n lineárních rovnic o m neznámých a rozšířená matice soustavy je

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

Je-li $a_{11} = 0$, lze podle věty 82 zaměnit řádky tak, aby byl prvek a_{11} nenulový. Dále předpokládejme $a_{11} \neq 0$. Pro každé $k = 2, \ldots, n$ přičteme ke každému k-tému řádku $(-\frac{a_{k1}}{a_{11}})$-násobek 1. řádku rozšířené matice. Takto vynulujeme všechny prvky prvního sloupce matice s výjimkou prvku a_{11}. Následně lze ze vzniklé ma-

tice vynechat všechny nulové řádky. Výsledkem je matice ve tvaru:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1m} & y_1 \\ 0 & a'_{22} & \cdots & a'_{2m} & y'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{r2} & \cdots & a'_{rm} & y'_r \end{array}\right),$$

kde $r \leq n$. [3]

Je-li $a'_{22} = 0$, zaměníme řádky, aby byl prvek nenulový, jsou-li všechny $a'_{22} = \cdots = a'_{r2} = 0$, přejdeme ke 3. sloupci. Za předpokladu $a'_{22} \neq 0$ pro každé $j = 3, \ldots, r$ přičteme k j-tému řádku $(-\frac{a'_{j2}}{a'_{22}})$-násobek 2. řádku, vynecháme nulové řádky. [3]

Nyní postup opakujeme, dokud nedostaneme odstupňovanou matici ve tvaru:

$$\left(\begin{array}{ccccccc|c} a_{11} & a_{12} & a_{13} & \cdots & \cdots & \cdots & a_{1m} & y_1 \\ 0 & a'_{22} & a'_{23} & \cdots & \cdots & \cdots & a'_{2m} & y'_2 \\ 0 & 0 & a''_{33} & \cdots & \cdots & \cdots & a''_{3m} & y''_3 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & c_{kk} & \cdots & c_{km} & z_k \end{array}\right),$$

kde $k \leq m$, $k = h(A)$. [3]

Poté z rovnice $c_{kk}x_k + \cdots + c_{km}x_m = z_k$ odpovídající poslednímu řádku vyjádříme x_k pomocí $x_{k+1}, \ldots, x_m$. Z předposlední rovnice vypočítáme x_{k-1}, atd. Je-li $k = m$, soustava má jedno řešení. Pokud je $k < m$, má daná soustava nekonečně mnoho řešení závislých na parametrech $x_{k+1}, \ldots, x_m$. [3]

PŘÍKLAD 84

Nalezněte řešení x soustavy lineárních rovnic

$$\begin{aligned} 4x_1 - 7x_2 + 8x_3 &= 37, \\ 2x_1 + 3x_2 - x_3 &= -11, \\ x_1 + x_2 - 4x_3 &= -11 \end{aligned}$$

pomocí Gaussovy eliminační metody.

Rozšířená matice soustavy je

$$(A|y^T) = \left(\begin{array}{ccc|c} 4 & -7 & 8 & 37 \\ 2 & 3 & -1 & -11 \\ 1 & 1 & -4 & -11 \end{array}\right).$$

S využitím elementárních řádkových transformací převedeme matici $(A|y^T)$ na odstupňovaný tvar

$$\left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 2 & 3 & -1 & -11 \\ 4 & -7 & 8 & 37 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 0 & 1 & 7 & 81 \\ 0 & -11 & 24 & 11 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 0 & 1 & 7 & 11 \\ 0 & 0 & 101 & 202 \end{array}\right).$$

Z posledního řádku víme, že $101x_3 = 202$, tedy $x_3 = 2$. Pomocí x_3 a předposledního řádku odstupňované matice vypočítáme: $x_2 = 11 - 7 \cdot 2 = -3$. Nakonec z prvního řádku vypočítáme $x_1 = -11 + 3 + 4 \cdot 2 = 0$.

Věta 85

Nechť $A_{n,m}$ je matice nad tělesem $\mathcal{T}$ a y, z jsou n-tice prvků tělesa $\mathcal{T}$. Je-li B regulární matice řádu n, $C = BA$ a platí $z^T = B \cdot y^T$, pak soustava $Ax = y$ má stejnou množinu řešení, jako soustava $Cx = z$. [1]

Důsledek 86

Nechť A je regulární matice řádu n nad tělesem $\mathcal{T}$ a y je n-tice prvků tělesa $\mathcal{T}$. Pak má soustava $Ax = y$ právě jedno řešení $x \in T^n$, pro které platí $x^T = A^{-1} \cdot y^T$. [1]

Mějme regulární matici A řádu n nad tělesem $\mathcal{T}$. Pak podle důsledku 86 má soustava rovnic $Ax = y$ právě jedno řešení x, kde $x^T = A^{-1} \cdot y^T$. Řešení tedy získáme vynásobením sloupce pravých stran inverzní maticí matice soustavy zleva. [1]

Matice A však nemusí být regulární. Při řešení nejprve od rozšířené matice soustavy $(A|y^T)$ přejdeme elementárními řádkovými transformacemi k odstupňované matici $(B|z^T)$ a zjistíme, zda je regulární.

Je-li matice singulární a řešitelná, pokračujeme v řešení s využitím Gaussovy eliminační metody. Pokud je matice regulární, lze elementárními řádkovými transformacemi přejít od matice $(B|z^T)$ k matici $(E|A^{-1} \cdot y^T)$ a nalézt tak řešení. [1]

Příklad 87

Nalezněte řešení x soustavy lineárních rovnic

$$\begin{aligned} 4x_1 - 7x_2 + 8x_3 &= 37, \\ 2x_1 + 3x_2 - x_3 &= -11, \\ x_1 + x_2 - 4x_3 &= -11 \end{aligned}$$

pomocí inverzní matice.

Rozšířená matice soustavy je

$$(A|y^T) = \left(\begin{array}{ccc|c} 4 & -7 & 8 & 37 \\ 2 & 3 & -1 & -11 \\ 1 & 1 & -4 & -11 \end{array}\right).$$

S využitím elementárních řádkových transformací převedeme matici $(A|y^T)$ na odstupňovaný tvar

$$(B|z^T) = \left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 0 & 1 & 7 & 11 \\ 0 & 0 & 101 & 202 \end{array}\right).$$

Dalšími úpravami transformujeme matici $(B|z^T)$ na matici $(E|x^T)$:

$$\left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 0 & 1 & 7 & 11 \\ 0 & 0 & 101 & 202 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & -4 & -11 \\ 0 & 1 & 7 & 11 \\ 0 & 0 & 1 & 2 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & 0 & -3 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 2 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 2 \end{array}\right).$$

Řešení nalezneme v posledním sloupci:

$$x^T = \begin{pmatrix} 0 \\ -3 \\ 2 \end{pmatrix},$$

platí tedy $x = \begin{pmatrix} 0 & -3 & 2 \end{pmatrix}$.

Známe-li inverzní matici, lze řešení získat podle důsledku 86 následovně:

$$x^T = A^{-1} \cdot y^T = \begin{pmatrix} \frac{11}{101} & \frac{20}{101} & \frac{17}{101} \\ \frac{-7}{101} & \frac{24}{101} & \frac{-20}{101} \\ \frac{1}{101} & \frac{11}{101} & \frac{-26}{101} \end{pmatrix} \cdot \begin{pmatrix} 37 \\ -11 \\ -11 \end{pmatrix} = \begin{pmatrix} 0 \\ -3 \\ 2 \end{pmatrix}.$$

1.3.2 Cramerovo pravidlo

Věta 88 (Cramerovo pravidlo)
Nechť $Ax = y$ je soustava n lineárních rovnic o n neznámých ($n \geq 1$) nad tělesem $\mathcal{T}$ taková, že $detA \neq 0$. Pak pro $j = 1, \ldots, n$ platí

$$x_j = \frac{detA_j}{detA},$$

kde A_j je matice, která vznikne z A tak, že j-tý sloupec nahradíme sloupcem pravých stran y^T. [2, 3]

Příklad 89
Nalezněte neznámou x_2 soustavy lineárních rovnic

$$\begin{aligned} x_1 - 7x_2 + 8x_3 &= 37, \\ x_1 + 3x_2 - x_3 &= -11, \\ {}_1 + x_2 - 4x_3 &= -11. \end{aligned}$$

Matice soustavy je

$$A = \begin{pmatrix} 4 & -7 & 8 \\ 2 & 3 & -1 \\ 1 & 1 & -4 \end{pmatrix},$$

sloupec pravých stran

$$y^T = \begin{pmatrix} 37 \\ -11 \\ -11 \end{pmatrix}.$$

Podle věty 88 platí

$$x_2 = \frac{\det A_2}{\det A} = \frac{\begin{vmatrix} 4 & 37 & 8 \\ 2 & -11 & -1 \\ 1 & -11 & -4 \end{vmatrix}}{\begin{vmatrix} 4 & -7 & 8 \\ 2 & 3 & -1 \\ 1 & 1 & -4 \end{vmatrix}} = \frac{303}{-101} = -3.$$

1.4 Homomorfismy

Definice 90 (Homomorfismus)
Nechť $\mathcal{V}_1 = (V_1, +, T, \cdot)$ a $\mathcal{V}_2 = (V_1, +, T, \cdot)$ jsou vektorové prostory nad tělesem $\mathcal{T}$. Zobrazení $f : V_1 \to V_2$ se nazývá homomorfismus (též lineární zobrazení) vektorového prostoru $\mathcal{V}_1$ do $\mathcal{V}_2$, jestliže platí [1, 3]:

1. $\forall u, v \in V_1 \qquad f(u + v) = f(u) + f(v)$,
2. $\forall u \in V_1, \forall c \in T \qquad f(c \cdot u) = c \cdot f(u)$.

Jestliže f je homomorfismus prostoru $\mathcal{V}_1$ do $\mathcal{V}_2$, potom množinu

$$\text{Ker } f = \{u \in V_1; f(u) = o\}$$

nazveme jádrem homomorfismu f, kde o je nulový vektor z V_2, a množinu

$$\text{Im } f = \{w \in V_2; \exists u \in V_1, f(u) = w\}$$

nazveme obraz homomorfismu f. [1, 3]

Definice 91 (Izomorfismus)
Bijektivní homomorfismus z $\mathcal{V}_1$ do $\mathcal{V}_2$ se nazývá izomorfismus $\mathcal{V}_1$ na $\mathcal{V}_2$. Řekneme, že vektorový prostor $\mathcal{V}_1$ je izomorfní s $\mathcal{V}_2$, jestliže existuje izomorfismus $\mathcal{V}_1$ na $\mathcal{V}_2$. [2, 3]

Věta 92
Je-li f homomorfismus $\mathcal{V}_1$ do $\mathcal{V}_2$, pak

1. *f je injektivní, právě když Ker $f = \{o\}$, nazývá se monomorfismus,*
2. *f je surjektivní, právě když Im $f = \mathcal{V}_2$, nazývá se epimorfismus,*
3. *f je izomorfismus, právě když Ker $f = \{o\}$ a Im $f = \mathcal{V}_2$,*

kde o je nulový vektor z $\mathcal{V}_1$. [2, 3]

Věta 93
Nechť $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ jsou vektorové prostory nad tělesem $\mathcal{T}$, pak [2, 3]:

1. *jsou-li* $f : V_1 \to V_2, g : V_2 \to V_3$ *homomorfismy, pak složené zobrazení* $f \circ g$ *je homomorfismus z* $\mathcal{V}_1$ *do* $\mathcal{V}_3$,

2. *je-li* f *izomorfismus* $\mathcal{V}_1$ *na* $\mathcal{V}_2$, *pak* f^{-1} *je izomorfismus* $\mathcal{V}_2$ *na* $\mathcal{V}_1$,

3. *identické zobrazení* $id : \mathcal{V}_1 \to \mathcal{V}_1$ *je izomorfismus* $\mathcal{V}_1$ *na* $\mathcal{V}_1$.

Věta 94

Nechť $\mathcal{V}_1, \mathcal{V}_2$ *jsou vektorové prostory nad tělesem* $\mathcal{T}$. *Je-li* $f : \mathcal{V}_1 \to \mathcal{V}_2$ *izomorfismus a je-li* $\{u_1, \ldots, u_n\}$ *báze* $\mathcal{V}_1$, *pak množina* $\{f(u_1), \ldots, f(u_n)\}$ *je báze* $\mathcal{V}_2$. *[2, 3]*

1.4.1 Transformace souřadnic

V následujícím textu budeme uvažovat vektorové prostory konečné dimenze a jejich báze jako tzv. pevné báze. Pevná báze je báze, kde záleží na pořadí vektorů, které ji definují. Proto budeme dále báze zapisovat, jako uspořádané n-tice.

Definice 95 (Souřadnice vektoru vzhledem k bázi)

Nechť $\mathcal{V} = (V, +, T, \cdot)$ je vektorový prostor nad tělesem $\mathcal{T}$, $\dim\mathcal{V} = n$ a $M = \langle u_1, \ldots, u_n \rangle$ je báze prostoru $\mathcal{V}$. Nechť $u \in V, c_1, \ldots, c_n \in T$, $u = \sum_{i=1}^{n} c_i u_i$. Potom koeficienty $c_1, \ldots, c_n$ nazýváme souřadnice vektoru u vzhledem k bázi M, zapisujeme $\{u\}_M = (c_1, \ldots, c_n)$. [2, 3]

Definice 96 (Matice přechodu)

Nechť $M_1 = \langle u_1, \ldots, u_n \rangle, M_2 = \langle v_1, \ldots, v_n \rangle$ jsou báze vektorového prostoru $\mathcal{V}$ nad tělesem $\mathcal{T}$ a nechť $\{v_i\}_{M_1} = (a_{i1}, \ldots, a_{in})$, $i = 1, \ldots, n$. Potom matici $A = (a_{ij})$ nazýváme matice přechodu od báze M_1 k bázi M_2. [2, 3]

Věta 97

Nechť M_1, M_2 *jsou dvě báze vektorového prostoru* $\mathcal{V} = (V, +, T, \cdot)$ *nad tělesem* $\mathcal{T}$, A *je matice přechodu od báze* M_1 *k* M_2. *Pak* $\forall u \in V$ *platí* $\{u\}_{M_1} = \{u\}_{M_2} \cdot A$. *[2, 3]*

Věta 98

Nechť M_1, M_2, M_3 *jsou báze vektorového prostoru* $\mathcal{V}$ *nad tělesem* $\mathcal{T}$. *Je-li* $A = (a_{ij})$ *matice přechodu od báze* M_1 *k* M_2, $B = (b_{ij})$ *matice přechodu od báze* M_2 *k* M_3, *pak* BA *je matice přechodu od báze* M_1 *k* M_3. *[2, 3]*

Věta 99

Nechť M_1, M_2 *jsou báze vektorového prostoru* $\mathcal{V}$ *nad tělesem* $\mathcal{T}$, A *je matice přechodu od* M_1 *k* M_2. *Pak matice* A *je regulární a* A^{-1} *je maticí přechodu od báze* M_2 *k* M_1. *[2, 3]*

Věta 100

Nechť $M_1 = \langle u_1, \ldots, u_n \rangle$ je báze vektorového prostoru $\mathcal{V}$ nad tělesem $\mathcal{T}$, A je regulární čtvercová matice stupně n. Pak množina $M_2 = \langle v_1, \ldots, v_n \rangle$, kde

$$v_i = \sum_{j=1}^{n} a_{ij} u_j, i = 1, \ldots, n,$$

je bází prostoru $\mathcal{V}$, A je maticí přechodu od M_1 k M_2. [2, 3]

Příklad 101

Určete matici přechodu A od báze $M_1 = \langle (-5, 9, 2), (6, -10, 5), (-1, 2, 9) \rangle$ k bázi $M_2 = \langle (0, 0, -5), (1, 0, 2), (-4, 2, 7) \rangle$ v prostoru $\mathbb{R}^3$.

Matici přechodu získáme při výpočtu souřadnic vektorů $(0, 0, -5)$, $(1, 0, 2)$, $(-4, 2, 7)$ vzhledem k bázi M_1. Dostáváme tři rovnice:

$$\begin{aligned}
(0, 0, -5) &= a_{11}(-5, 9, 2) + a_{12}(6, -10, 5) + a_{13}(-1, 2, 9), \\
(1, 0, 2) &= a_{21}(-5, 9, 2) + a_{22}(6, -10, 5) + a_{23}(-1, 2, 9), \\
(-4, 2, 7) &= a_{31}(-5, 9, 2) + a_{32}(6, -10, 5) + a_{33}(-1, 2, 9),
\end{aligned}$$

ze kterých dostáváme tři soustavy lineárních rovnic:

$$\begin{aligned}
-5a_{11} + 6a_{12} - a_{13} &= 0, \\
9a_{11} - 10a_{12} + 2a_{13} &= 0, \\
2a_{11} + 5a_{12} + 9a_{13} &= -5,
\end{aligned}$$

$$\begin{aligned}
-5a_{21} + 6a_{22} - a_{23} &= 1, \\
9a_{21} - 10a_{22} + 2a_{23} &= 0, \\
2a_{21} + 5a_{22} + 9a_{23} &= 2,
\end{aligned}$$

$$\begin{aligned}
-5a_{31} + 6a_{32} - a_{33} &= -4, \\
9a_{31} - 10a_{32} + 2a_{33} &= 2, \\
2a_{31} + 5a_{32} + 9a_{33} &= 7.
\end{aligned}$$

Protože jsou pravé strany všech tří soustav stejné, lze řešit všechny tři soustavy současně. Sestavíme tedy rozšířenou matici soustavy se třemi pravými stranami.

$$\left(\begin{array}{ccc|c|c|c} -5 & 6 & -1 & 0 & 1 & -4 \\ 9 & -10 & 2 & 0 & 0 & 2 \\ 2 & 5 & 9 & -5 & 2 & 7 \end{array}\right) \sim \left(\begin{array}{ccc|c|c|c} -5 & 6 & -1 & 0 & 1 & -4 \\ 0 & 4 & 1 & 0 & 9 & -26 \\ 0 & 37 & 43 & -25 & 12 & 27 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|c|c|c} -5 & 6 & -1 & 0 & 1 & -4 \\ 0 & 4 & 1 & 0 & 9 & -26 \\ 0 & 0 & 1 & -\frac{20}{27} & -\frac{19}{9} & \frac{214}{27} \end{array}\right) \sim \left(\begin{array}{ccc|c|c|c} 1 & 0 & 0 & \frac{10}{27} & \frac{32}{9} & -\frac{296}{27} \\ 0 & 1 & 0 & \frac{5}{27} & \frac{25}{9} & -\frac{229}{27} \\ 0 & 0 & 1 & -\frac{20}{27} & -\frac{19}{9} & \frac{214}{27} \end{array}\right)$$

Hledaná matice přechodu od báze M_1 k bázi M_2 je

$$A = \begin{pmatrix} \frac{10}{27} & \frac{5}{27} & -\frac{20}{27} \\ \frac{32}{9} & \frac{25}{9} & -\frac{19}{9} \\ -\frac{296}{27} & -\frac{229}{27} & \frac{214}{27} \end{pmatrix}.$$

2 Cílová platforma a zvolené technologie

Hlavní cílovou platformou byl zvolen web, aby bylo možné aplikaci využívat na co největším počtu zařízení. Díky zvolené technologii lze aplikaci snadno rozšířit také na telefony se systémem Android a iOS, viz dále.

Aplikace obsahuje také výukový text. Aby bylo možné text jednoduše aktualizovat nebo rozšiřovat, je zapotřebí ho načítat z databáze a nedistribuovat ho jako součást aplikace. Proto bylo třeba navrhnout a zrealizovat backend s API pro práci s daty.

2.1 Backend

Pro backend byl zvolen framework Django v jazyce Python, protože jsem s ním v uplynulém roce pracoval nejčastěji, a také protože nabízí intuitivní práci s databází, code-first přístup a jednoduchý způsob správy dat v podobě integrovaného administrátorského systému.

Pro data byl zvolen databázový engine PostgreSQL.

2.1.1 Django

Django je free open-source aplikační framework v jazyce Python. Drží se architektury podobné MVC (Model-View-Controller), i když přesnější akronym je spíše MTV (Model-Template-View). [4]

Nabízí objektově relační mapování. Modely se specifikují jako třídy v Pythonu, které dědí od třídy Model a jejich proměnné jsou definované jako instance subtříd Field, které reprezentují jednotlivá pole v databázi. [4]

View podle interpretace Djanga popisuje, jaká data jsou prezentována uživateli, ale ke specifikování, jak jsou data prezentována (jak vypadají), slouží Template. V tom se architektura Djanga liší od tradiční MVC architektury. [4]

2.2 Frontend

Protože aplikace cílí především na web a telefony, byl použit pro frontend framework Flutter v jazyce Dart.

Hlavní myšlenkou Flutteru je, abychom byli schopní aplikaci naprogramovat pouze jednou a kód následně zkompilovat pro jednotlivé platformy, na kterých aplikace běží nativně. Není tedy nutné udržovat různé verze aplikace pro různé platformy.

2.2.1 Dart

Dart je free open-source jazyk pro multiplatformní vývoj, který vznikl v Googlu. Umožňuje kompilaci do strojového kódu pro ARM, x64 a do javascriptu pro

webové aplikace. Nabízí AOT (Ahead of time) i JIT (Just in time, používaný především při vývoji) kompilátor. Má podobnou syntaxi jako například Typescript nebo Kotlin. [5]

Jazyk Dart je typově bezpečný a staticky typovaný jazyk. Typové anotace ale nejsou povinné, mohou být implicitně odvozené od hodnoty, a to při překladu nebo až za běhu pomocí typu *dynamic*. Kontrola typů tedy probíhá při překladu i za běhu. [5]

Nabízí také tzv. „sound null safety“, tedy že proměnná nemůže mít hodnotu *null*, pokud není explicitně definovaná jako „nullable“. Na rozdíl od jiných jazyků, pokud Dart stanoví proměnnou jako „non-nullable“, je vždy „non-nullable“ a pokud bychom se do takové proměnné pokusili přiřadit hodnotu *null*, okamžitě bychom dostali výjimku. [5]

Především kvůli svému typovému systému Dart pro běh aplikace vždy vyžaduje Dart runtime, nehledě na platformu. Runtime se stará o správu paměti, vynucení typového systému (jak bylo zmíněno, některé kontroly typů jsou dynamické) a správu tzv. „isolates“ (vlákna s izolovanou pamětí). Executable pro nativní platformy automaticky obsahuje Dart runtime. [5]

2.2.2 Flutter

Flutter je framework v jazyce Dart pro vývoj multiplatformních aplikací, které na jednotlivých platformách běží nativně. V současné době podporuje vývoj pro Android, iOS, Windows, macOS, Linux a web. [6]

Bere si inspiraci z webového frameworku React a ve Flutteru je téměř vše widget, i samotná aplikace.

Každý widget popisuje, jak má vypadat jeho uživatelské rozhraní vzhledem k jeho současné konfiguraci a stavu. Když se stav změní, widget přegeneruje svůj popis, framework ho porovná s předchozím a rozhodne, jaké minimální změny ve vykreslovacím stromu je nutné provést k přechodu do nového stavu. [6]

Veškerá uživatelská rozhraní se tvoří kompozicí jednotlivý widgetů. Flutter nabízí základní widgety, jako například *Row*, *Column*, *Text* nebo *Transform*. Vestavěné jsou však i knihovny *Material* (obsahující widgety podle Material Design od společnosti Google) a *Cupertino* (obsahující widgety implementující současný design iOS). [6]

Při vývoji vytváříme vlastní widgety, které se dělí na dva typy – *StatelessWidget* a *StatefulWidget*.

StatelessWidget, jak vidno z názvu, nemá změnitelný stav. Všechna jeho pole (proměnné) musí být deklarovaná jako *final* a nemohou tedy později změnit hodnotu. Při definovaní nového widgetu dědíme *StatelessWidget* a musíme implementovat metodu *build*, ve které definujeme, jak má vypadat uživatelské rozhraní. [6]

Druhý typ widgetů, *StatefulWidgets*, nám umožňuje vytvářet komplexnější aplikace a reagovat na uživatelský vstup. *StatefulWidget* má definováno, jak vytvořit *State* objekt, ve kterém se uchovává stav.

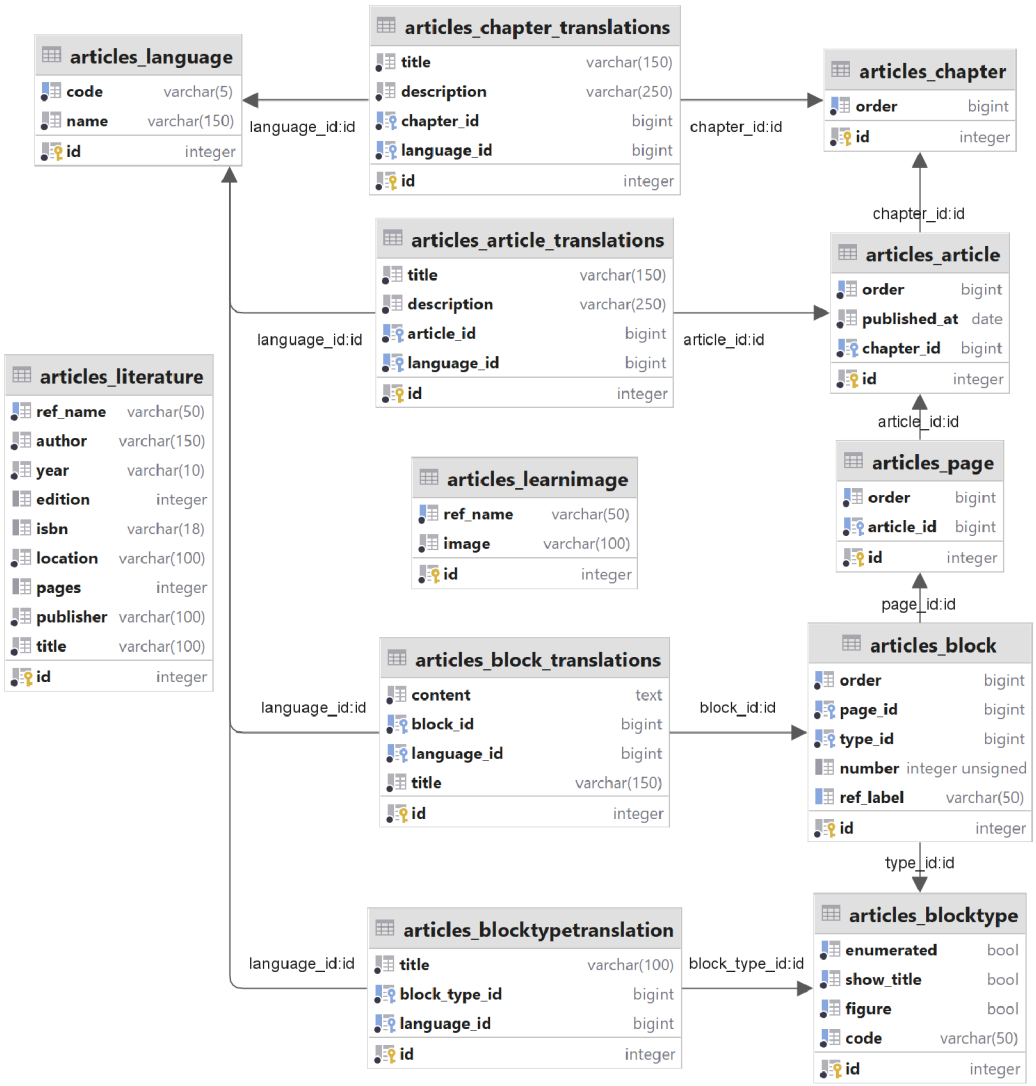
Když definujeme vlastní *StatefulWidget*, definujeme dvě třídy – jednu, která dědí *StatefulWidget* a stejně jako *StatelessWidget* může obsahovat pouze pole deklarovaná jako *final* (používají se pro specifikování konfigurace předané rodičem) a reprezentuje samotný widget, ale neimplementujeme v ní metodu build. A druhou třídu, která dědí *State*, implementujeme v ní metodu build a reprezentuje stav widgetu.

Třída *State* už může obsahovat proměnné, které nejsou konstantní. Když chceme stav změnit, pomocí metody *setState* signalizujeme frameworku, že došlo ke změně a je třeba widget překreslit. [6]

3 Řešení

V této kapitole je nejprve popsána struktura databáze a implementace backendu poskytující rozhraní pro čtení dat z databáze. Následně je popsán frontend – nejprve uživatelské rozhraní a následně samotná implementace.

3.1 Návrh databáze



Obrázek 2: Schéma databáze

Výukový text je rozdělen do kapitol, podkapitol (v rámci databáze tzv. článků) a stránek. Navíc je každá strana rozdělena do bloků (na definice, věty, ...), pro zjednodušení odkazování na bloky (definice, věty, ...) a snazší zobrazování textu v aplikaci.

Při návrhu databáze jsem počítal s možností překladu textů do různých jazyků. Každá z tabulek obsahujících texty (kapitoly, články, bloky, typy bloků) má k sobě ještě sekundární tabulku s jednotlivými překlady, viz obrázek 2.

V textu lze odkazovat na bloky pomocí tagu *ref* (stejným způsobem jako v TeXu). Aby bylo možné na blok odkazovat, je třeba u něho nastavit label (pole *ref_label*).

Je možné i citovat literaturu, stejně jako v TeXu pomocí tagu *cite*. Pro citovanou literaturu je v databázi tabulka *articles_literature* obsahující všechna data potřebná k citování a label (pole *ref_name*), viz obrázek 2.

Nakonec je třeba zmínit tabulku *articles_learnimage*, díky které je možné do výukového textu vkládat obrázky. Tabulka obsahuje cestu k souboru (pole *image*) a label (pole *ref_name*). Pro vytvoření bloku obsahujícího obrázek je třeba nastavit typ bloku na „image“ (záznam v tabulce *articles_blocktype* s polem *code* obsahujícím hodnotu „image“) a obsah bloku nastavit na label požadovaného obrázku.

3.2 Backend API

Django backend nabízí šest koncových bodů. Tři z nich jsou jazykově závislé a všechny podporují pouze metodu GET. Ke specifikování jazyka slouží parametr *locale_id* (integer).

Adresa všech koncových bodů začíná „*/api/learn*“ (např. */api/learn/literature*), pro jednoduchost budou adresy dále uváděny bez této společné části (např. */literature*).

3.2.1 Koncový bod – GET /:locale_id/chapter

Koncový bod vracející seznam všech kapitol. Pro každou kapitolu vrací její id, nadpis a popis ve specifikovaném jazyce, viz zdrojový kód 1.

```
1 @require_http_methods(["GET"])
2 def chapters_view(request: HttpRequest, locale_id: int):
3     chapters = Chapter.objects.filter(chaptertranslation__language=
          locale_id).values(
4         chapter_id=F("id"),
5         chapter_title=F("chaptertranslation__title"),
6         chapter_description=F("chaptertranslation__description"),
7     )
8     return JsonResponse({"chapters": list(chapters)})
```

Zdrojový kód 1: Implementace seznamu kapitol

Implementace je velmi přímočará a skládá se z jediného dotazu. Je pouze nutné filtrovat kapitoly podle jazyku a specifikovat, jaké hodnoty chceme získat.

3.2.2 Koncový bod – GET /:locale_id/chapter/:chapter_id

Koncový bod, který vrací detail kapitoly specifikované pomocí parametru *chapter_id* (integer). Konkrétně vrací její id, nadpis, popis a seznam článků, které kapitola obsahuje, s jejich id, nadpisem a popisem.

Implementace, viz zdrojový kód 2, se skládá ze dvou dotazů. Nejprve se vyhledá kapitola podle id. Pokud existuje, ve druhém dotazu získáme seznam článků dané kapitoly, jinak vracíme error.

```
@require_http_methods(["GET"])
def chapter_view(request: HttpRequest, locale_id: int, chapter_id:
    int):
    chapter = Chapter.objects.filter(
        id=chapter_id, chaptertranslation__language=locale_id
    ).values(
        chapter_id=F("id"),
        chapter_title=F("chaptertranslation__title"),
        chapter_description=F("chaptertranslation__description"),
    )

    if not chapter.exists():
        return JsonResponse({"error": "Chapter not found"}, status
            =404)

    articles = Article.objects.filter(
        chapter=chapter_id, articletranslation__language=locale_id
    ).values(
        article_id=F("id"),
        article_title=F("articletranslation__title"),
        article_description=F("articletranslation__description"),
    )

    return JsonResponse(
        {**chapter.get(), "articles": list(articles)}
    )
```

Zdrojový kód 2: Implementace detailu kapitoly

3.2.3 Koncový bod – GET /:locale_id/article/:article_id

Koncový bod vracející detail článku specifikovaného pomocí parametru *article_id* (integer). Vrací informace o kapitole, ve které se článek nachází (id, nadpis), informace o článku (id, nadpis) a seznam stránek článku.

Každou stránku vrací jako seznam bloků, kde každý blok má číslo bloku (pořadové číslo číslovaného bloku uložené v poli *number*, jiné než id), boolean hodnotu *type_visible* (určuje, zda má mít blok zobrazený název typu), název typu (např. „Definice“), nadpis a obsah bloku.

Detaily článků jsou získány pomocí dvou dotazů, viz zdrojový kód 3. První dotaz získá informace o článku a kapitole, ve které se článek nachází. Druhý získá všechny bloky nacházející se v článku. Následně je třeba bloky roztřídit podle stránek, aby se s nimi na frontendu snadno pracovalo.

```
@require_http_methods(["GET"])
def article_view(request: HttpRequest, locale_id: int, article_id: int):
    article = Article.objects.filter(
        id=article_id,
        chapter__chaptertranslation__language=locale_id,
        articletranslation__language=locale_id,
    ).values(
        "chapter_id",
        chapter_title=F("chapter__chaptertranslation__title"),
        article_id=F("id"),
        article_title=F("articletranslation__title"),
    )

    if not article.exists():
        return JsonResponse({"error": "Article not found"}, status=404)

    blocks = list(
        Block.objects.filter(
            page__article=article_id,
            blocktranslation__language=locale_id,
            type__blocktypetranslation__language=locale_id,
        )
        .values(
            page_index=F("page__order"),
            block_number=F("number"),
            block_type_visible=F("type__show_title"),
            block_type_title=F("type__blocktypetranslation__title"),
            block_title=F("blocktranslation__title"),
            block_content=F("blocktranslation__content"),
        )
        .order_by("page_index", "order")
    )

    pages, curr_page = [], -1
    for block in blocks:
        if block["page_index"] != curr_page:
            curr_page = block["page_index"]
            pages.append([])
        pages[-1].append(block)

    return JsonResponse({**article.get(), "pages": pages})
```

Zdrojový kód 3: Implementace detailu článku

3.2.4 Koncový bod – GET /literature

Koncový bod sloužící k získání detailu použité literatury. Má volitelný parametr *ref_name* (řetězec). Pokud je parametr specifikovaný, vrací detail konkrétního literárního zdroje, jinak vrací seznam všech zdrojů v databázi. Pro každý literární zdroj vrací všechna data potřebná k citování (tedy jméno autora, rok vydání, název, ...).

Při vyhodnocení dotazu na koncový bod nejprve dojde ke kontrole, zda je specifikován parametr *ref_name*, viz zdrojový kód 4. Pokud ano, použije se jako filtr dotazu na databázi, jinak se vrací seznam všech literárních zdrojů v databázi. V obou případech se vrací všechna pole modelu (tabulky) *Literature*.

```
@require_http_methods(["GET"])
def get_literature_view(request: HttpRequest):
    if "ref_name" in request.GET:
        lit = Literature.objects.filter(
            ref_name__iexact=request.GET["ref_name"]
        ).values()

        if not lit.exists():
            return JsonResponse({"error": "Literature not found"},
                status=404)

        return JsonResponse({**lit.get()})
    else:
        return JsonResponse({"literature": list(
            Literature.objects.all().values()
        )})
```

Zdrojový kód 4: Implementace detailu reference literatury

3.2.5 Koncový bod – GET /ref

Koncový bod k získání detailu reference na blok. Má volitelný parametr *ref_name* (řetězec). Pokud je specifikovaný, vrací detail konkrétní reference, jinak vrací seznam všech bloků, které mají definovaný label. Pro každou referenci vrací typ bloku, číslo bloku, id kapitoly, článku a strany, kde se odkazovaný blok nachází.

Implementace je velmi podobná implementaci literárních referencí, viz zdrojový kód 5. V případě reference na blok je však třeba definovat hodnoty, které chceme získat, jelikož se data nenachází pouze v jedné tabulce.

3.2.6 Koncový bod – GET /image/:ref_name

Koncový bod pro získání obrázku pomocí parametru *ref_name* (řetězec, který může obsahovat pouze ASCII písmena, čísla, pomlčku a podtržítko). Jedná se o jediný koncový bod, jehož standardní odpovědí není JSON (který vrací pouze

```python
@require_http_methods(["GET"])
def get_reference_view(request: HttpRequest):
    is_filtered = "ref_name" in request.GET
    if is_filtered:
        ref = Block.objects.filter(ref_label__iexact=request.GET["
            ref_name"])

        if not ref.exists():
            return JsonResponse({"error": "Reference not found"},
                status=404)
    else:
        ref = Block.objects.filter(ref_label__isnull=False)

    ref = ref.values(
        "ref_label",
        block_type=F("type"),
        block_number=F("number"),
        page_order=F("page__order"),
        article=F("page__article"),
        chapter=F("page__article__chapter"),
    )

    return JsonResponse(
        {**ref.get()} if is_filtered else {"reference": list(ref)}
    )
```

Zdrojový kód 5: Implementace detailu reference

```python
@require_http_methods(["GET"])
def get_learn_image_view(request: HttpRequest, ref_name: str):
    image_path_q = LearnImage.objects.filter(ref_name__iexact=
        ref_name).values("image")

    if image_path_q.exists(): # Check if the image is in the database
        image_path = os.path.join(MEDIA_ROOT, image_path_q.get()["
            image"])
        if os.path.exists(image_path): # Check if the image file
            exists
            return FileResponse(open(image_path, "rb"))

    return JsonResponse({"error": "Image not found"}, status=404)
```

Zdrojový kód 6: Implementace obrázku

v případě chyby), ale soubor. Slouží k zobrazování obrázků ve výukové sekci aplikace.

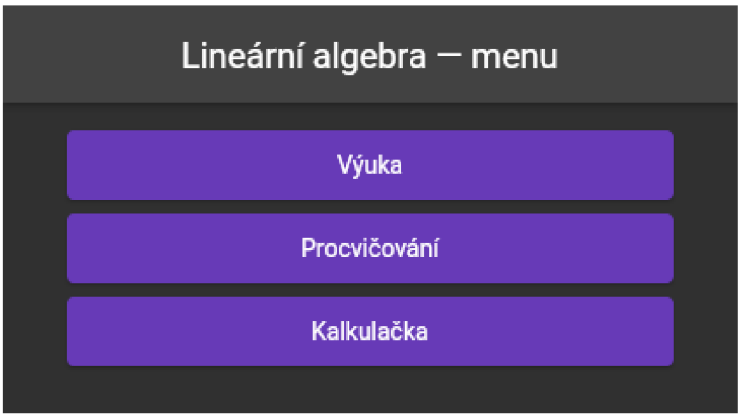
Při vyhodnocení dotazu nejprve dojde ke kontrole, zda se zadaný obrázek nachází v databázi, viz zdrojový kód 6. Pokud ano, přečte se z databáze cesta k souboru a existuje-li, dotaz vrátí soubor, jinak vrací chybu ve formátu JSON.

3.3 Frontend

V následující kapitole je popsán frontend – respektive webová aplikace s možností procházení výukového textu a simulování maticových operací a operací souvisejících s vektorovými prostory.

3.3.1 Uživatelské rozhraní

Po načtení aplikace se uživateli zobrazí hlavní menu s odkazy do tří sekcí aplikace – sekcí výuka, procvičování a kalkulačka, viz obrázek 3.



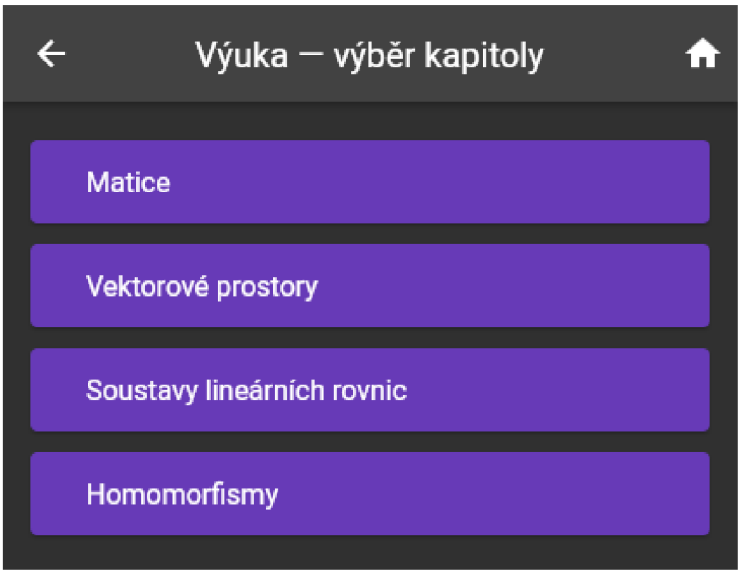
Obrázek 3: Hlavní menu aplikace

3.3.1.1 Sekce výuka

Sekce výuka obsahuje již zmíněný výukový text, který je rozdělen na kapitoly, podkapitoly (články) a stránky. Při zvolení sekce výuka se uživateli zobrazí výběr kapitoly, viz obrázek 4. Po zvolení kapitoly se uživateli zobrazí obdobné menu s výběrem podkapitoly.

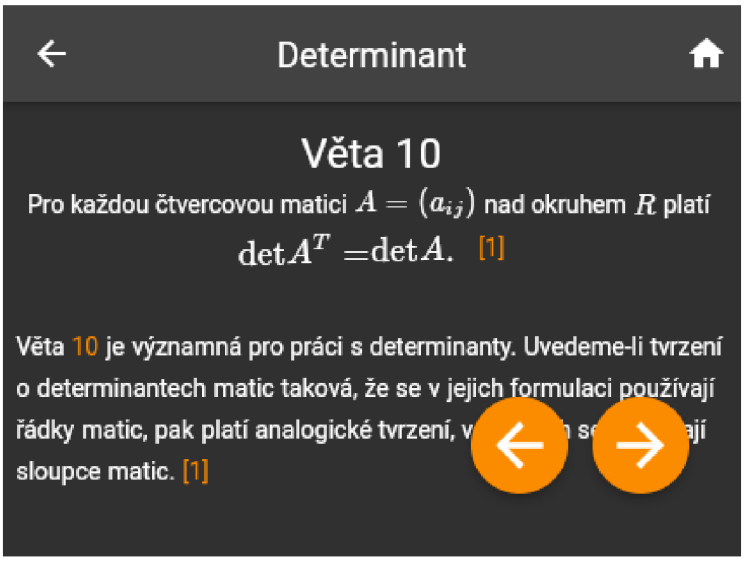
Oproti hlavnímu menu je zde navíc v levém horním rohu tlačítko zpět, pro návrat o jednu nabídku zpět a v pravém horním rohu je tlačítko pro návrat do hlavního menu, které je velmi užitečné pro rychlý přechod mezi sekcemi.

Po zvolení podkapitoly, resp. článku, se uživateli již zobrazí samotný výukový text. Text článků je rozdělen do stránek, mezi kterými lze přecházet pomocí tlačítek zpět a vpřed umístěných v pravém dolním rohu obrazovky. Jak již bylo zmíněno v kapitole 3.1, jednotlivé stránky jsou dále rozděleny do bloků, jako jsou



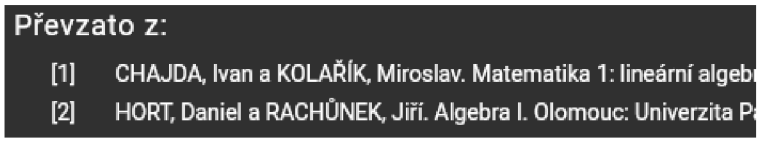
Obrázek 4: Výuka – výběr kapitoly

věty, definice apod. Jsou zde však i textové bloky bez nadpisu (a čísla), například odstavec pod větou 10 na obrázku 5.



Obrázek 5: Výuka – podkapitola (článek)

V textu se také nachází reference na bloky, které uživatele přepnou na stránku s odkazovanou větou, a citace literatury, ze které jsem čerpal při psaní výukového textu. Citace na literaturu jsou v textu zobrazeny pouze pořadovým číslem a na konci každé strany je zobrazen seznam literatury použitý na dané straně, viz obrázek 6.

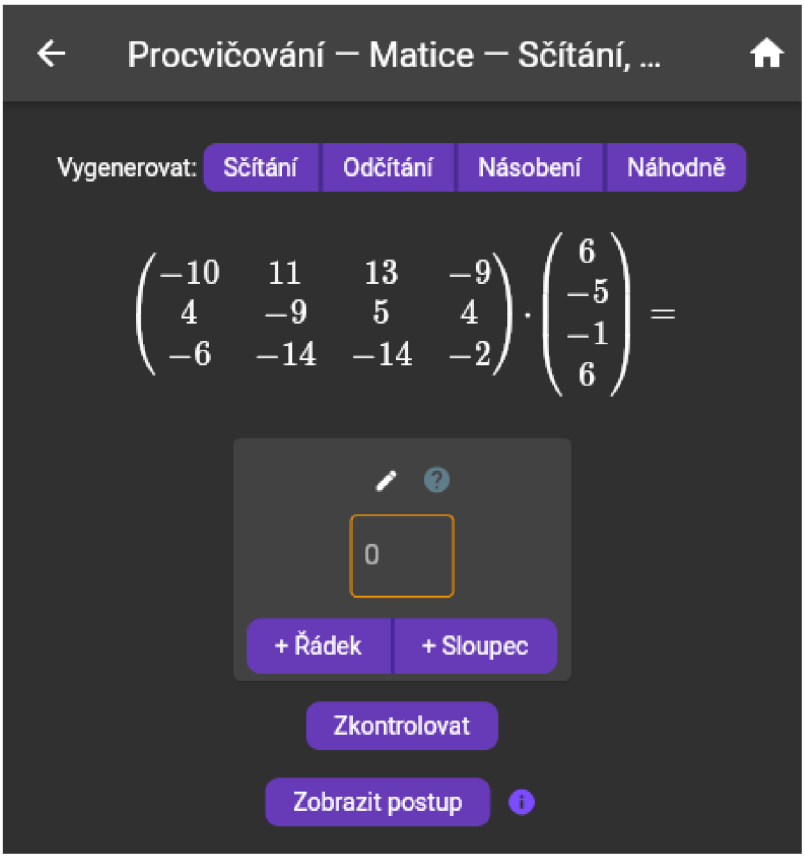


Obrázek 6: Výuka – seznam literatury

3.3.1.2 Sekce procvičování

Sekce procvičování je dále rozdělena do podsekcí podle procvičované látky. Je zde procvičování operací s maticemi (sčítání, odčítání a násobení), výpočtu determinantu a inverzní matice. Dále také procvičování lineární (ne)závislosti vektorů, nalezení báze, výpočet transformační matice, procvičování homomorfismů a řešení soustav lineárních rovnic.

Všechny podsekce procvičování mají stejnou strukturu, viz obrázek 7. První řádek je složen z tlačítek k vygenerování nového příkladu – různá tlačítka slouží ke generování různých typů příkladů, na obrázku 7 k vygenerování příkladů na různé maticové operace nebo například ke generování různých rozměrů u determinantů (2×2, 3×3, nebo větší) apod.



Obrázek 7: Procvičování – maticové operace

Po tlačítkách následuje zadání samotného příkladu a pod ním rozhraní pro za-

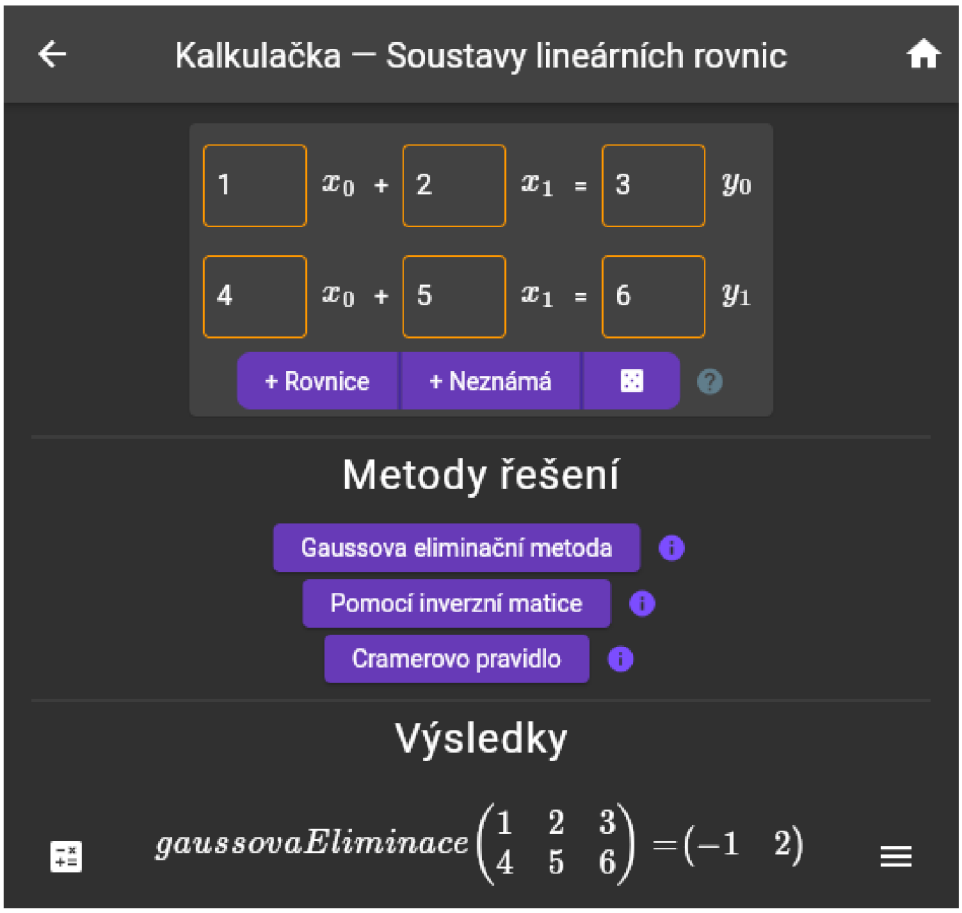
dání řešení. Rozhraní pro zadávání řešení se liší v závislosti na typu příkladu – jedná se buď o matici, vektor, množinu vektorů, nebo skalár.

Dále je zde tlačítko pro kontrolu řešení, nebo v některých případech k zadání řešení samotného (např. lineární nezávislost vektorů má pro řešení pouze tlačítka „Lin. nezávislé“ / „Lin. závislé“).

Nakonec sekci uzavírá tlačítko pro zobrazení postupu (umožňuje krokovat řešení od zadání k výsledku, viz obrázek 11, podrobněji popsané v kapitole 3.3.1.3) a tlačítko „Zjistit více“ (s ikonou „i“), které uživatele odkáže na výukový text na téma zadaného příkladu.

3.3.1.3 Sekce kalkulačka

Sekce kalkulačka je podobně jako sekce procvičování rozdělena do několika podsekcí. Všechny výpočty, které lze procvičovat, lze také počítat v sekci kalkulačka (s výjimkou homomorfismů).



Obrázek 8: Kalkulačka – soustavy lineárních rovnic

Navíc je zde násobení matice skalárem, určení hodnosti matice, transponované matice a transformace souřadnic od báze k bázi. Soustavy lineárních rovnic

lze kromě Gaussovy eliminační metody (používané pro zobrazení řešení v procvičování) řešit také pomocí Cramerova pravidla a inverzní matice.

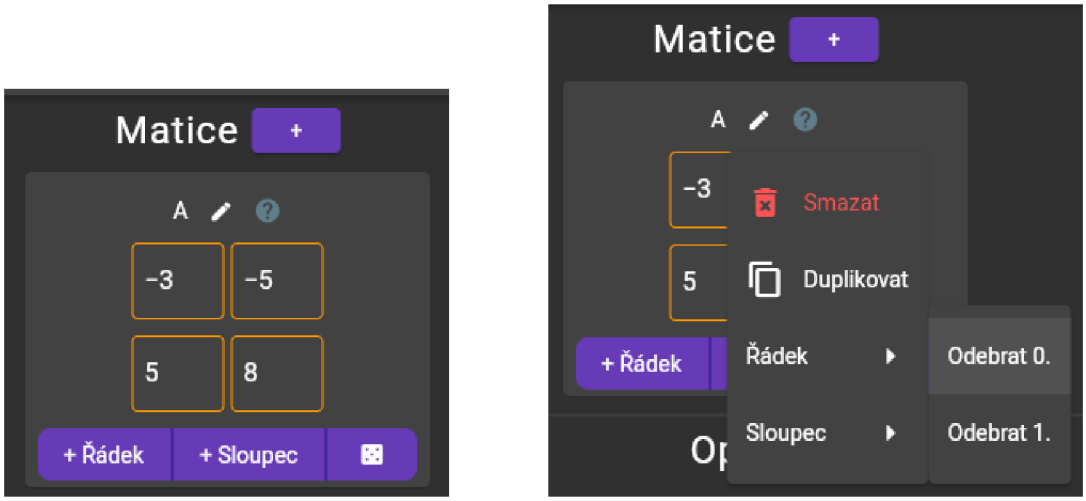
Všechny podsekce kalkulačky se drží jednotné struktury skládající se ze tří částí – vstupní data (matice, vektory, soustava lineárních rovnic), operace (např. determinant, nalezení báze, řešení soustavy pomocí Cramerova pravidla) a výsledky, viz obrázek 8.

V horní části obrázku 8 můžeme vidět rozhraní pro zadávání soustavy lineárních rovnic. Nabízí tlačítka pro přidání rovnice (řádku) a neznámé (sloupce). Třetí tlačítko s ikonou kostky slouží k vyplnění soustavy náhodnými celými čísly v rozsahu $\langle -10; +10 \rangle$.

Kliknutím na názvy proměnných lze odebírat řádky/sloupce (např. kliknutím na x_0 lze odebrat první sloupec a kliknutím na y_1 lze odebrat druhý řádek).

Ve spodní části obrázku 8 jsou výsledky předchozích operací, v tomto případě výsledek řešení soustavy pomocí Gaussovy eliminační metody.

U každého výsledku ve všech sekcích kalkulačky se vlevo nachází tlačítko s ikonou kalkulačky, které slouží k zobrazení kroků výpočtu a vpravo se nachází menu tlačítko, kterým lze daný výsledek smazat, nebo přidat do vektoru/matice pro další výpočet.



(a) Zadávání matic

(b) Menu pro úpravu matic

Obrázek 9: Kalkulačka – zadávání matic

Na obrázku 9a lze vidět rozhraní pro zadávání matic. Stejně jako rozhraní pro zadávání rovnic má ve spodní části tlačítka pro přidání řádku/sloupce a vyplnění matice náhodnými celými čísly v rozsahu $\langle -10; +10 \rangle$.

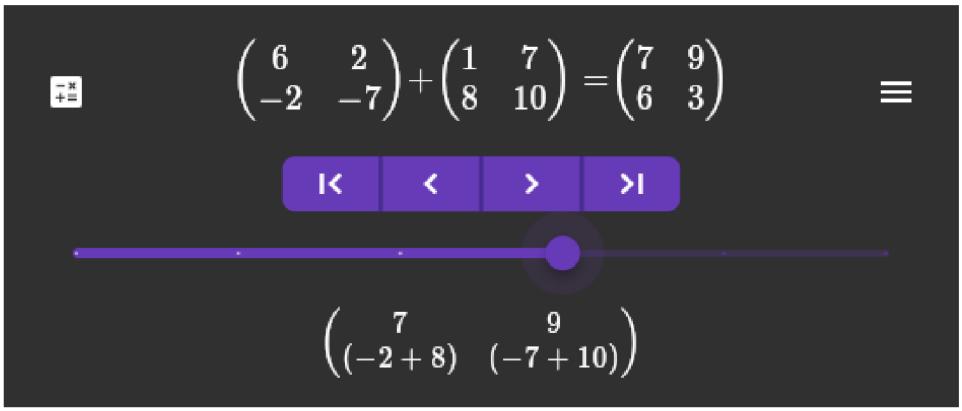
V horní části vedle názvu matice se nachází tlačítko pro úpravu matice (ikona tužky), které umožňuje matici odebrat, duplikovat a odebírat řádky/sloupce matice, viz obrázek 9b.

Rozhraní pro zadávání vektorů je velice podobné tomu pro zadávání matic, viz obrázek 10. Rozdílem je, že vektory neumožňují odebírání konkrétních položek, ale vždy se odebírá ta poslední (položka nejvíce vpravo), také je tlačítko

umístěno ve spodní části společně s tlačítkem pro přidávání. Navíc se místo menu pro úpravu vedle názvu vektoru nachází tlačítko pro jeho smazání.

Obrázek 10: Kalkulačka – zadávání vektorů

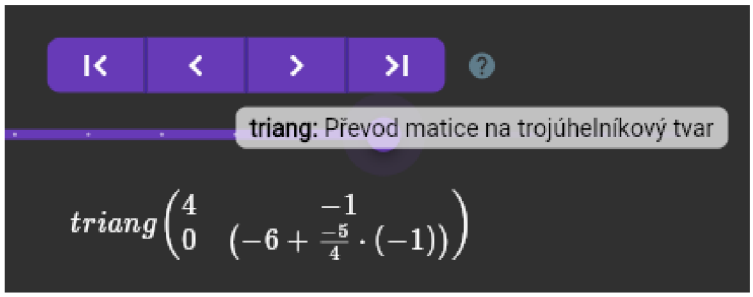
Krokování umožňuje dva způsoby procházení kroků, viz obrázek 11. A to tlačítka pro posunutí o krok vpřed nebo zpět a tlačítka pro skok na první krok (zadání) a poslední krok (řešení).



Obrázek 11: Krokování výpočtů

U některých operací (např. u výpočtu inverzní matice, nebo řešení soustavy lineárních rovnic) je však kroků mnoho a tato tlačítka jsou nedostačující. Proto byl přidán ještě posuvník, který plní dvě funkce – vizualizace, jak blízko řešení se daný krok nachází, a možnost rychlého průchodu kroky.

Komplexnější operace, jako například převod matice na trojúhelníkový nebo redukovaný tvar, jsou v krocích a v řešení zapsané pomocí klíčových slov (např. *triang* = převod matice na trojúhelníkový tvar), aby byl tento zápis kratší. V případě takových operací se vedle tlačítek pro přechod mezi kroky výpočtu zobrazí ikona otázníku s vysvětlivkami, viz obrázek 12 (vysvětlivky se zobrazí po najetí myší na ikonu, případně stisknutím ikony na telefonu).



Obrázek 12: Vysvětlivky operací

3.3.2 Struktura aplikace

Při návrhu struktury aplikace jsem cílil na snadnou rozšiřitelnost o další části (další podsekce procvičování a kalkulačky). K tomu slouží datové modely *SectionChapterModel*, *SectionPageModel* a tři generické widgety *AlgMenuView*, *AlgChapterView* a *AlgPageView*.

Aplikace má definované tři sekce – výuku, procvičování a kalkulačku. Výuková sekce má svou specifickou implementaci, která stahuje, zpracovává a následně zobrazuje data z databáze. Zbylé dvě sekce využívají snadno rozšiřitelnou strukturu.

Procvičování a kalkulačka mají definovaný seznam kapitol (*SectionChapterModels*), které obsahují. Kapitoly mají definovaný nadpis, seznam stránek (*SectionPageModels*) a nepovinně také podtitulek. Stránky mají také svůj nadpis a nepovinný podtitulek, ale především mají definovaný widget, který určuje obsah stránky.

Výsledný seznam kapitol může vypadat například jako zdrojový kód 7. V tomto příkladu jsou `BinaryMatrixExc()` a `LinIndependenceExc()` widgety, které tvoří tělo daných stránek.

Zobrazení takto definovaných stránek probíhá pomocí výše zmíněných generických widgetů. Přejde-li uživatel do jedné z takto definovaných sekcí, widget *AlgMenuView* zobrazí hlavní panel aplikace s nadpisem sekce a v těle aplikace zobrazí menu pro výběr kapitoly.

Při zvolení jedné z kapitol vykreslí její obsah widget *AlgChapterView*, který se stará o zobrazení hlavního panelu aplikace a obsahu těla aplikace. Obsah závisí na počtu stránek kapitoly. Má-li kapitola pouze jednu stránku, zobrazí se její obsah, v opačném případě se zobrazí menu pro výběr stránek kapitoly a až po zvolení stránky se zobrazí její obsah pomocí widgetu *AlgPageView*.

Pokud bychom tedy chtěli například přidat další stránku do jedné z podsekcí procvičování, stačí vytvořit widget implementující obsah nové stránky a následně ho přidat do definice seznamu *exerciseChapters*. Nové části procvičování a kalkulačky lze takto přidávat s minimálním zásahem do existujícího kódu aplikace.

```
const List<SectionChapterModel> exerciseChapters = [
  SectionChapterModel(
    title: 'Matice',
    subtitle: 'Součet, rozdíl, součin, determinant, inverzní matice',
    pages: [
      SectionPageModel(
        title: 'Sčítání, odčítání, násobení',
        page: BinaryMatrixExc(),
      ),
    ],
  ),
  SectionChapterModel(
    title: 'Vektorové prostory',
    subtitle:
        'Lineární (ne)závislost vektorů, nalezení báze, transformace
            souřadnic od báze k bázi, ...',
    pages: [
      SectionPageModel(
        title: 'Lineární (ne)závislost vektorů',
        page: LinIndependenceExc(),
      ),
    ],
  ),
```

Zdrojový kód 7: Příklad definice sekcí procvičování

3.3.3 Zobrazení výukového textu v aplikaci

Výukový text obsahuje kromě běžného textu také spoustu matematiky, kterou bylo třeba v aplikaci korektně zobrazovat. Pro reprezentaci a zobrazování matematiky byl TeX jasnou volbou.

Pro Flutter existují knihovny, které TeX zobrazují (resp. jen matematiku). Zvolil jsem knihovnu *flutter_math_fork*, protože na rozdíl od ostatních dostupných knihoven vykresluje TeX čistě pomocí Dartu a Flutteru.

Ostatní knihovny používají pro překlad a zobrazení TeXu javascriptové knihovny. Výsledek nebývá dobrý. Způsob, jakým knihovnu a její výstup integrují do Flutteru, je velmi neflexibilní a také se s těmito knihovnami nepracuje přívětivě.

Oproti tomu použití knihovny *flutter_math_fork* je velmi přímočaré. K zobrazení matematiky slouží widget *Math*, který nabízí konstruktor `Math.tex(String expression)`, kde parametr *expression* je výraz v TeXu.

Při zpracování výukového textu jsem nejprve jednotlivé jeho bloky rozděloval na prostý text, (inline) matematiku a blok (display) matematiky. Později však bylo nutné implementovat i seznamy, citace literatury a reference (obrázků a ostatních bloků textu, např. vět).

3.3.3.1 Datový model výukového textu

V databázi je text uložen v podobě řetězce, který je na straně frontendu zpracován a pro zobrazení uchován ve specifické datové struktuře. Ta se skládá ze dvou úrovní (hlavních tříd, které mají několik potomků) – *LBlockContent* a *LBlockSegment*, které budou dále společně s jejich potomky pro jednoduchost uváděny bez společného prefixu „LBlock“.

Třída *Content* reprezentuje větší úseky textu, jedná se o třídy *ParagraphContent*, *ListContent* a *LiteratureContent*. Třída *ParagraphContent* reprezentuje odstavec textu a skládá se ze seznamu objektů typu *Segment*. Dále třída *ListContent* obsahuje seznam objektů *ParagraphContent*, kde každý z objektů (odstavců) reprezentuje jednu odrážku seznamu. Zbývající třída *LiteratureContent* se skládá ze seznamu literatury a slouží k zobrazení seznamu použité literatury na konci strany.

Segment a jeho potomci reprezentují dílčí části odstavců, ačkoliv v krajním případě se může odstavec skládat z jediného segmentu, jestliže odstavec neobsahuje žádnou formu matematiky ani žádnou referenci. Třída *Segment* se skládá z typu a obsahu (řetězec), je použitá pro běžný text, matematiku a odkazy v textu na literaturu a ostatní bloky. Typ slouží k určení, jaký je obsah segmentu a jakým způsobem by měl být zobrazen.

Jediným potomkem třídy *Segment* je *TabularCellSegment*, jehož název může být poněkud zavádějící. Neslouží totiž k sestavování tabulek, ale k zajištění stejného odsazení v různých řádcích seznamu (použito např. v číslovaném seznamu v definici 47).

3.3.3.2 Překlad bloků výukového textu

Pro zjednodušení zpracování bloků výukového textu jsem definoval rozšíření `String.splitWithDelim(RegExp pattern)`, které funguje podobně jako `String.split(RegExp pattern)`, ale výsledný seznam navíc obsahuje i oddělovače, podle kterých byl řetězec rozdělený.

Při zpracování bloku se nejprve blok rozdělí pomocí nově definovaného rozšíření `String.splitWithDelim` a regulárního výrazu `RegExp(r'\(begin|end){(enumerate|itemize)}')`. Vzniklé pole řetězců procházíme v cyklu a pokud narazíme na oddělovač, změníme podle jeho hodnoty aktuální typ obsahu, viz zdrojové kódy 8 a 9. Pokud se nejedná o oddělovač, je řetězec zpracován v závislosti na aktuálně nastaveném typu obsahu.

Výchozem typem je odstavec. Při zpracování odstavce se řetězec rozdělí na matematiku, text a citace/reference pomocí rozšíření `String.splitWithDelim` podle regulárního výrazu `RegExp(r'\${1,2}|\\(cite|ref){([a-zA-Z\d:\-_,]+)}')`. Opět se kontroluje přítomnost oddělovačů a upravuje se podle nich aktuální typ segmentu. Výsledné segmenty jsou ukládány do výstupního pole.

Jedná-li se o seznam, je jeho obsah rozdělen podle oddělovače `\item` a jednotlivé řádky jsou zpracovány stejným způsobem, jako odstavce.

```
List<LBlockContent> parseBlock(String block, {bool isLastBlock =
    false}) {
  _currentType = LBlockContentType.paragraph;
  List<LBlockContent> blockContent = [];

  for (var segment in block.splitWithDelim(
    RegExp(r'\\(begin|end){(enumerate|itemize)}'),
  )) {
    segment = segment.trim();
    if (segment.isEmpty || _updateType(segment)) {
      continue;
    }
    blockContent.addAll(_parseSegment(segment));
  }

  if (isLastBlock && usedLiterature.isNotEmpty) {
    blockContent.add(LBlockLiteratureContent(usedLiterature.toList
        ()));
  }
  return blockContent;
}
```

Zdrojový kód 8: Překlad bloku

```
bool _updateType(String segment) {
  if (segment.contains('begin{itemize')) {
    _currentType = LBlockContentType.list;
  } else if (segment.contains('begin{enumerate')) {
    _currentType = LBlockContentType.enumeratedList;
  } else if (segment.contains(RegExp(r'\\end{(enumerate|itemize)}')
      )) {
    _currentType = LBlockContentType.paragraph;
  } else {
    return false;
  }
  return true;
}
```

Zdrojový kód 9: Aktualizace aktuálního typu bloku

3.3.4 Výrazy a jejich vyhodnocování

Nejprve jsem k výpočtům přistupoval tak, že jsem definoval třídu *Matice*, která obsahovala dvourozměrné pole čísel. Dále jsem na ní definoval všechny unární operace (výměna řádků, převod na trojúhelníkovou matici, determinant, ...), a binární opearce pomocí přetížení operátorů sčítání, odčítání a násobení. Podobným způsobem jsem definoval také třídu *Vektor*.

Tento přístup fungoval velice dobře a výpočty byly rychlé, ale narazil jsem

na zásadní problém, kvůli kterému jsem byl nucen celou logiku výpočtů přepsat.

Pro výukovou aplikaci je nesmírně důležité krokování jednotlivých operací, jenže to tento způsob řešení neumožňoval a promarnil jsem zbytečně mnoho času snahou generovat při výpočtech datovou strukturu obsahující provedené kroky.

Nakonec jsem zvolil symbolické vyhodnocování výrazů. Definoval jsem rozhraní *Expression* (výraz) s metodou `simplify()`, která má provést jeden krok zjenodušení výrazu a vrátit výsledek. Rozhraní má také metodu `toTeX()`, která, jak je patrné z názvu, vrací reprezentaci výrazu v TeXu.

Také jsem se rozhodl algebraické výpočty definovat jako samostatnou Dart knihovnu, aby bylo možné při vývoji rychleji iterovat díky kratší kompilaci (knihovna byla testovaná pouze s rozhraním příkazové řádky).

3.3.4.1 Odbočka o reprezentaci čísel

Před implementací výrazů je třeba zmínit problematiku reprezentace čísel. Aby při výpočtech nedocházelo ke ztrátě přesnosti vlivem necelých čísel, rozhodl jsem se reprezentovat čísla pomocí zlomků. Nevýhodou je, že takto nelze pracovat s iracionálními čísly, ale to pro výukové účely této aplikace není zapotřebí.

Pro práci se zlomky existuje pro jazyk Dart knihovna *Fraction*, která reprezentuje čitatele a jmenovatele pomocí datového typu integer.

Při použití knihovny *Fraction* jsem však brzy narazil na problém. Při některých výpočtech, např. při výpočtu redukované trojúhelníkové matice, vznikají zlomky s velkým čitatelem a jmenovatelem. Začal jsem tudíž narážet na problém s přesností čísel, který se však projevoval pouze ve webové verzi aplikace. Oproti tomu při testování knihovny samotné vše fungovalo správně.

Důvodem je rozdílná reprezentace čísel v prostředí Dart Native a Dart Web. Dart Native používá pro reprezentaci integeru 64bitový dvojkový doplněk se znaménkem, ale Dart Web reprezentuje integer pomocí 64bitové plovoucí řádové čárky. [5]

V některých případech tedy výpočet v prostředí Dart Native proběhl v pořádku, ale v prostředí Dart Web byla při překročení rozsahu nastavena hodnota některého z operandů na nekonečno a výpočet skončil výjimkou.

Problém jsem vyřešil úpravou knihovny *Fraction* (vytvořením forku). Místo typu *Int* jsem použil pro čitatele a jmenovatele typ *BigInt*, který nabízí integery s volitelnou přesností omezenou pouze pamětí. Novou knihovnu (fork) jsem nazval *BigFraction*.

3.3.4.2 Implementace výrazů – základní struktury

Nejprve bylo třeba definovat struktury, pro které budou zbylé výrazy definované. Je důležité, že samotné struktury jsou také výrazy (implementují rozhraní *Expression*).

První a nejjednodušší strukturou je *Scalar* (skalár), který má číselnou hodnotu (typu *BigFraction*) a metoda `simplify()` vrací původní výraz (vrací sám sebe), protože nelze více zjednodušit.

Další důležitou strukturou je *Vector*, který obsahuje seznam výrazů. Metoda `simplify()` postupně projde seznam výrazů (prvků vektoru), pokud narazí na výraz, který lze zjednodušit, vrací nový vektor se zjednodušeným prvkem. Nelze-li žádný z prvků zjednodušit, metoda vrací původní vektor.

Struktura *Matrix* (matice) je definovaná jako seznam výrazů (vektorů, nebo výrazů jejichž výsledkem je vektor). V konstruktoru se kontroluje, zda v seznamu není neplatný výraz (například množina, definovaná dále) a jsou-li v seznamu vektory, kontroluje se, zda mají všechny stejnou velikost.

Dalšími strukturami jsou *Boolean* (má hodnotu typu boolean, použité jako výsledek lin. nezávislosti vektorů) a *ExpressionSet* (velmi podobné vektoru, ale místo seznamu používá množinu, takže nemůže obsahovat duplikátní prvky), *Variable* a *CommutativeGroup*.

Výrazy *Variable* a *CommutativeGroup* slouží primárně pro zapsání řešení soustavy lineárních rovnic v případě, že má více než jedno řešení a k určení, zda je zobrazení homomorfismem. *Variable* obsahuje název (např. „x“) a index proměnné, *CommutativeGroup* obsahuje komutativní operaci (sčítání nebo násobení) a seznam výrazů, mezi kterými je operace provedena (může obsahovat např. výraz $2 + 3 + 4$).

CommutativeGroup slouží ke zjednodušení práce s výrazy obsahujícími proměnné (např. $2x + y + 3$). Bez využití *CommutativeGroup* by takový výraz byl uložený ve stromu několika výrazů sčítání a násobení. Hlavním benefitem tohoto pomocného výrazu je usnadnění porovnání dvou objektů typu *CommutativeGroup* (významné pro určení, zda je zobrazení homomorfismem).

3.3.4.3 Implementace výrazů – základní operace

Následně bylo třeba definovat výrazy pro základní operace, jako je sčítání, odčítání, násobení a inverzi.

Všechny zmíněné binární operace se skládají ze dvou operandů. Metoda `simplify()` nejprve zjednodušuje operandy, dokud je lze více zjednodušit. Pokud jsou již oba operandy zjednodušené a je pro jejich dvě struktury operace definovaná, vypočítá se a metoda vrací výsledek (například součet dvou skalárů vrací nový skalár s výsledkem).

Zbývající výraz, inverze, se skládá pouze z výrazu, který má být invertován. Operace inverze je definována pouze pro skalár a matici. Inverzní matice se počítá pomocí reciproké matice, podle věty 28. S inverzní maticí trochu předbíhám, jelikož pro její výpočet je třeba definovat několik maticových operací.

Také je třeba definovat výrazy *AreEqual* a *And*. Oba se skládají se ze dvou operandů. Výsledkem výrazu *AreEqual* je výraz *Boolean*, který má hodnotu *true* v případě, že se hodnoty operandů (např. dva skaláry) rovnají. Výraz *And* (logický součin) vrací *Boolean* s hodnotou *true* v případě, že oba jeho operandy jsou výrazy typu *Boolean* s hodnotami *true*.

3.3.4.4 Implementace výrazů – maticové operace

Pro složitější operace je nejprve nutné pro matici definovat elementární řádkové transformace:

- *ExchangeRows* (výměna řádků) – obsahuje matici, nad kterou se operace vykonává a indexy řádků, které se mají vyměnit,
- *MultiplyRowByN* (vynásobení řádku skalárem) – obsahuje matici, index řádku a n,
- *AddRowToRowNTimes* (přičtení n-násobku řádku matice k jinému řádku) – obsahuje matici, indexy a n.

Dalším definovaným výrazem byl převod matice na trojúhelníkový tvar, který je rozdělen do dvou výrazů, *Triangular* a *TriangularDet*, podle toho, zda se jedná o operaci nad maticí nebo determinantem.

Pomocí těchto výrazů je již možné počítat determinant o libovolném rozměru. K tomu slouží výraz *Determinant*, který má pro matice o rozměru $n < 3$ pevně definovaný výpočet a pro větší matice determinant nejprve převede na trojúhelníkový tvar a následně vynásobí hodnoty na diagonále (podle věty 12).

Také jsou definované následující výrazy počítající maticové operace:

- *Transpose* – převod matice na transponovanou,
- *Reduce* – převod matice na redukovaný tvar,
- *Minor* – výpočet minoru matice,
- *AlgSupplement* – výpočet algebraického doplňku matice,
- *Rank* – výpočet hodnosti matice (převede matici na redukovaný tvar a zjistí počet nenulových řádků).

3.3.4.5 Implementace výrazů – řešení soustav lineárních rovnic

Pro práci se soustavami lineárních rovnic jsou definované následující výrazy:

- *IsSolvable* – skládá se z matice soustavy a sloupce (vektoru) pravých stran, podle věty 81 (Frobeniovy) určí, zda je soustava řešitelná,
- *GaussianElimination* – skládá se z rozšířené matice soustavy, řeší soustavu pomocí Gaussovy eliminační metody,
- *SolveWithCramer* – skládá se z matice soustavy a sloupce pravých stran, řeší soustavu pomocí Cramerova pravidla,
- *SolveWithInverse* – skládá se z matice soustavy a sloupce pravých stran, řeší soustavu pomocí inverzní matice (podle důsledku 86).

3.3.4.6 Implementace výrazů – vektorové prostory

Pro výpočet operací souvisejících s vektorovými prostory jsou definované následující výrazy:

- *AreVectorsLinearlyIndependent* – skládá se ze seznamu vektorů (nebo výrazů, jejichž výsledkem je vektor), výsledkem je výraz *Boolean* s hodnotou *true* v případě, že jsou vektory lineárně nezávislé, řešeno obdobně jako v příkladu 46,
- *FindBasis* – skládá se z matice (kde každý řádek představuje jeden z vektorů generujících vektorový prostor), výsledkem je výraz *ExpressionSet* reprezentující bázi vektorového prostoru, báze je získána převodem matice na redukovaný trojúhelníkový tvar,
- *TransformMatrix* – skládá se ze dvou výrazů *ExpressionSet*, které reprezentují dvě báze vektorového prostoru, výsledkem je matice přechodu od první báze ke druhé,
- *TransformCoords* – implementuje transformaci souřadnic od báze k bázi, skládá se z transformační matice a vektoru (souřadnic),
- *Mapping* – implementuje zobrazení, skládá se ze vstupního vektoru a vektoru zobrazení (*mapping* vektoru), např. máme-li zobrazení $(x_1, x_2) \rightarrow (x_2+1, x_1)$, pak vektor (x_2+1, x_1) je vektorem zobrazení (využívá se výrazů *Variable* a *CommutativeGroup*),
- *IsHomomorphism* – skládá se z vektoru zobrazení a velikosti vstupního vektoru, vrací výraz typu *Boolean* s hodnotou *true*, je-li zadané zobrazení homomorfismem.

Závěr

Vytvořený program, respektive webová aplikace, nabízí možnost procházení výukového textu, procvičování formou řešení náhodně generovaných příkladů a možnost krokování uživatelem zadaných výpočtů (tzv. režim kalkulačky).

V současné době je možné v aplikaci simulovat následující operace:

- sčítání, odčítání a násobení matic,
- násobení matice skalárem,
- výpočet determinantu,
- určení hodnosti matice,
- převod na matice na transponovanou,
- výpočet inverzní matice,
- převod matice na trojúhelníkový nebo redukovaný tvar,
- určení lineární (ne)závislosti vektorů,
- nalezení báze vektorového prostoru,
- výpočet matice přechodu od báze k bázi,
- transformace souřadnic od báze k bázi,
- určit, zda je zobrazení homomorfismem,
- řešení soustav lineárních rovnic pomocí Gaussovy eliminační metody, Cramerova pravidla a inverzní matice.

Aplikace je navržena tak, aby bylo možné ji snadno rozšířit o další sekce, viz kapitola 3.3.2. Navíc díky zvolené technologii (frameworku Flutter), ji lze snadno rozšířit i na další zatím nepodporované platformy (jako např. iOS, nebo Linux).

V budoucnu bych aplikaci rád rozšířil i na mobilní platformy. Konkrétně bych rád aplikaci vydal na Google Play pro telefony se systémem Android a na App store pro telefony se systémem iOS.

Dále bych do budoucna rád vylepšil uživatelské rozhraní, například krokování operací (aby zobrazení jednotlivých kroků bylo přehlednější a více interaktivní s lepšími vysvětlivkami). Také bych rád vylepšil zobrazování výukového textu a předělal interpretaci TeXu, aby byla robustnější a podporovala více možností formátování.

Conclusions

The developed program, or web application, offers the possibility of browsing the educational text, practicing by solving randomly generated exercises and the possibility of going through user-specified calculations step-by-step (the so-called calculator mode).

Currently, the following operations can be simulated in the application:

- matrix addition, subtraction and multiplication,
- multiplying matrix by a scalar,
- calculating the determinant,
- determining the rank of a matrix,
- converting a matrix to a transposed matrix,
- calculating the inverse of a matrix,
- converting a matrix to triangular or reduced form,
- determining the linear (in)dependence of vectors,
- finding the basis of a vector space,
- calculating the base-to-base transform matrix,
- transformation of coordinates from base to base,
- determining, whether a mapping is a homomorphism,
- solving systems of linear equations using Gaussian elimination method, Cramer's rule and inverse matrix.

The application is designed in such a way that it can be easily extended with additional sections, see chapter 3.3.2. Moreover, thanks to the chosen technology (Flutter framework), it can be easily extended to other platforms thar are not supported yet (such as iOS or Linux).

In the future I would also like to release mobile version of the application. I would like to publish it on Google Play for Android devices and on App store for iOS devices.

In the future I would also like to improve the user interface, for example, the stepping operations (to make the displayed steps clearer and more interactive with better explanations of the operations). I would also like to improve the way the educational text is displayed and redesign the TeX interpretation to be more robust and support more formatting options.

A Obsah elektronických dat

Aktuální zdrojový kód je také dostupný v git repozitáři na portále GitHub: github.com/jimmyl0l3c/dp_algebra/. Stav, v jakém byl zdrojový kód odevzdaný spolu s diplomou prací, značí label „v1.0.0“.

`algebra_lib/`
: Adresář obsahující zdrojové kódy knihovny pro jazyk Dart. Knihovna obsahuje implementace algebraických výrazů popsaných v kapitole 3.3.4.

`backend/`
: Adresář obsahující zdrojové kódy Django aplikace sloužící jako backend výukové aplikace, viz kapitola 3.2.

`ci_cd/`
: Adresář obsahující konfigurační soubory pro nasazení webové aplikace včetně backendu a databáze s využitím Docker kontejnerů. Také obsahuje bash skript *run.sh* pro nasazení.

`flutter_app/`
: Adresář obsahující zdrojové kódy frontendu, respektive webové aplikace popsané v kapitole 3.3.

`text/`
: Adresář s textem práce ve formátu PDF, včetně zdrojových souborů pro typografický systém LaTeX potřebných pro bezproblémové vytvoření PDF dokumentu textu.

`README.txt`
: Textový soubor obsahující instrukce k nasazení webové aplikace v produkčním prostředí a spuštění aplikace ve vývojovém prostředí, včetně všech požadavků pro bezproblémový provoz. Dále obsahuje instrukce pro načtení zálohy databáze ze souboru *dump.sql*.

`dump.sql`
: Záloha PostgreSQL databáze obsahující výukový text.

`db.sqlite3`
: Databáze formátu Sqlite obsahující testovací data, využívána pro vývoj.

Literatura

[1] Bečvář, Jindřich. *Lineární algebra*. 5. vyd. Praha: Matfyzpress, 2019. ISBN 978-80-7378-378-5.

[2] Hort, Daniel; Rachůnek, Jiří. *Algebra I*. Olomouc: Univerzita Palackého, 2003. ISBN 80-244-0631-4.

[3] Chajda, Ivan; Kolařík, Miroslav. *Matematika 1: lineární algebra*. Olomouc: Univerzita Palackého, 2021.

[4] *Django (version 4.1)*. Lawrence, Kansas: Django Software Foundation, 2022. Dostupný z: ⟨https://www.djangoproject.com/⟩.

[5] *Dart programming language*. 2023. Dostupný z: ⟨https://dart.dev⟩.

[6] *Flutter – Build apps for any screen*. 2023. Dostupný z: ⟨https://flutter.dev⟩.

[7] Emanovský, Petr; Kühr, Jan. *Cvičení z algebry pro 1. ročník (I)*. Olomouc: Univerzita Palackého v Olomouci, 2007. ISBN 978-80-244-1833-9.