EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103006/I/2020/36097107977012228

GAUSSOVA ELIMINAČNÁ METÓDA RIEŠENIA SYSTÉMOV LINEÁRNYCH ROVNÍC VYUŽITÍM JAZYKA JAVA

Diplomová práca

2020 **Bc. Natália Kováčová**

EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

GAUSSOVA ELIMINAČNÁ METÓDA RIEŠENIA SYSTÉMOV LINEÁRNYCH ROVNÍC VYUŽITÍM JAZYKA JAVA

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra matematiky a aktuárstva

Vedúci záverečnej práce: PaedDr. Zsolt Simonka, PhD.

Bratislava 2020 **Bc. Natália Kováčová**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracovala samostatne a že som uviedla všetku použitú literatúru.

Dátum: **.....**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Natália Kováčová
Študijný program:	informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor:	ekonómia a manažment
Typ záverečnej práce:	Inžinierska záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Gaussova eliminačná metóda riešenia systémov lineárnych rovníc využitím jazyka Java

Anotácia: Práca bude venovaná využitiu programovacieho jazyka Java v lineárnej algebre. Súčasťou práce bude vytvorenie programu, ktorý postupným zadávaním ekvivalentných riadkových úprav rieši systém m lineárnych rovníc s n neznámymi. Program má slúžiť jednak ako učebná pomôcka pri štúdiu lineárnej algebry, a tiež ako efektívna pomôcka v kontrolnej fáze vyučovacieho procesu. V tejto súvislosti požiadavky kladené na jeho tvorbu sú jednoduchosť ovládania, názornosť a dobrá čitateľnosť výstupov, ako aj zabezpečenie jeho funkčnosti pre prípad nevhodných vstupov. Demonštrovanie možností praktického využitia vytvoreného programu bude súčasťou záverečnej práce.

Vedúci:	PaedDr. Zsolt Simonka, PhD.
Katedra:	KMA FHI - Katedra matematiky a aktuárstva FHI
Vedúci katedry:	Ing. Michal Páleš, PhD.
Dátum zadania:	11.10.2018
Dátum schválenia:	23.10.2018

doc. RNDr. Jozef Fecenko, CSc.
vedúci katedry

ABSTRAKT

KOVÁČOVÁ, Natália: *Gaussova eliminačná metóda riešenia systémov lineárnych rovníc využitím jazyka Java* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra matematiky a aktuárstva. – Vedúci záverečnej práce: PaedDr. Zsolt Simonka, PhD. – Bratislava: FHI EU, 2020, 58 strán.

Cieľom záverečnej práce je vytvoriť program v jazyku Java, ktorý rieši systémy *m* lineárnych rovníc s *n* neznámymi využitím Gaussovej eliminačnej metódy. Práca je rozdelená do štyroch kapitol. Obsahuje 31 obrázkov a 2 tabuľky. Prvá kapitola je venovaná najmä definícii pojmov lineárnej algebry a charakteristike jazyka Java. V druhej kapitole je charakterizovaný cieľ práce a v tretej kapitole metodika práce a metódy skúmania. Štvrtá kapitola obsahuje popis implementácie vytvorenej aplikácie a ukážku funkcionality. Výsledkom riešenia danej problematiky je funkčná aplikácia na riešenie systémov lineárnych rovníc.

Kľúčové slová: Gaussova eliminačná metóda, Gauss-Jordanova eliminačná metóda, systémy lineárnych rovníc, Java

ABSTRACT

KOVÁČOVÁ, Natália: *Gaussian elimination method of solving linear equations using Java programming* - University of Economics in Bratislava. Faculty of Economic Informatics; Department of Mathematics and Actuarial Science. – Diploma thesis supervisor: PaedDr. Zsolt Simonka, PhD. – Bratislava: FHI EU, 2020, 58 pages.

The goal of the thesis is to create a program in Java that solves systems of *m* linear equations with *n* unknowns using the Gaussian elimination method. Thesis is divided into four chapters. It contains 31 images and 2 tables. First chapter deals with definition of linear algebra terms and description of Java language. Second chapter describes the aim of the thesis and the third chapter defines work methodology and research methods. Fourth chapter contains a description of application implementation and a demonstration of the functionality. The output of the thesis is an application for solving systems of linear equations.

Keywords: Gaussian elimination, Gauss-Jordan elimination, system of linear equations, Java

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek

JVM - Java Virtual Machine

JRE - Java Runtime Environment

JDK - Java Development Kit

LTS – Long term support

GUI – Graphical user interface

AWT - Abstract Window Toolkit

Úvod

Systémy lineárnych rovníc sú základnou súčasťou lineárnej algebry, ktorá sa využíva nielen v oblastiach matematiky, ale aj v iných vedeckých disciplínach a v praxi. V súčasnosti sa na riešenie komplexnejších úloh lineárnej algebry využívajú rôzne softvérové nástroje, ktoré značne urýchľujú a zjednodušujú proces riešenia. Existuje viacero najmä webových aplikácii na riešenie úloh lineárnej algebry, ktoré ale iba vypočítajú a vypíšu výsledok riešenia, prípadne zobrazia postup.

Cieľom záverečnej práce je vytvoriť program v jazyku Java, ktorý rieši systém m lineárnych rovníc s n neznámymi postupným zadávaním ekvivalentných úprav. Program má slúžiť jednak ako učebná pomôcka pri štúdiu lineárnej algebry a tiež ako efektívna pomôcka v kontrolnej fáze vyučovacieho procesu.

Začiatok prvej kapitoly je venovaný teórii lineárnej algebry, definícii systémov lineárnych rovníc a ich riešiteľnosti. Ďalej je tu vysvetlený pojem matica, stupňovitý tvar matice a inverzná matica. V ďalšej časti tejto kapitoly je uvedená analýza a porovnanie aplikácii zameraných na riešenie systémov lineárnych rovníc. Posledná časť obsahuje charakteristiku programovacieho jazyka Java a využitých technológii.

V druhej kapitole je sformulovaný cieľ práce a jednotlivé čiastkové ciele potrebné na dosiahnutie hlavného cieľa.

Tretia kapitola obsahuje popis metód riešenia systémov lineárnych rovníc a metodiky vývoja aplikácie. V tejto časti sú špecifikované aj požiadavky na aplikáciu a návrh používateľského rozhrania.

V poslednej kapitole je vysvetlená implementácia hlavných algoritmov použitých v aplikácii a ukážka využitia aplikácie na konkrétnych príkladoch.

1 Súčasný stav riešenej problematiky doma a v zahraničí

V prvej kapitole sa budeme venovať definícii potrebných pojmov z teórie lineárnej algebry. Ďalej uvedieme analýzu a porovnanie aplikácii zameraných na riešenie systémov lineárnych rovníc. V poslednej časti tejto kapitoly budeme charakterizovať programovací jazyk Java, ktorý využijeme na vývoj aplikácie.

1.1 Lineárna algebra

Lineárna algebra je oblasť matematiky, ktorá študuje vektory, matice, vektorové a lineárne priestory a systémy lineárnych rovníc. Vektory a matice charakterizujú systémy lineárnych rovníc a tie zase môžu modelovať problém reálneho sveta.

Jednou z najdôležitejších úloh lineárnej algebry je riešenie systémov lineárnych rovníc. Veľká časť matematických problémov, ktoré sa vyskytujú vo vedeckých alebo priemyselných aplikáciách, v určitej fáze spejú k vyriešeniu lineárneho systému. S použitím metód modernej matematiky je často možné redukovať komplexnejší problém na jediný systém lineárnych rovníc.

Hoci je lineárna algebra odvetvím matematiky, využíva sa aj v mnohých iných oblastiach, napríklad v štatistike, ekonomike, elektrotechnike a fyzike. S vývojom počítačov a so zvýšením významu simulácií a modelovania sa výrazne rozšírilo použitie lineárnej algebry aj v ďalších disciplínach. Lineárnu algebru používajú inžinieri a vedci na navrhovanie a analýzu zložitých systémov. Stavební inžinieri ju používajú na navrhovanie a analýzu nosných štruktúr, ako sú mosty, strojní inžinieri na navrhovanie a analýzu systémov odpruženia a elektrotechnici na navrhovanie a analýzu elektrických obvodov. [4]

1.2 Matice

Matica je množina čísel usporiadaných do riadkov a stĺpcov. Veľkosť alebo rozmer matice je definovaný ako $m \times n$, kde m je počet riadkov a n je počet stĺpcov. Všeobecný tvar matice je

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & \cdots & \cdots & a_{mn} \end{pmatrix}$$

Matica $\mathbf{A}$, pre ktorú platí $m = n$, sa nazýva *štvorcová matica n-tého stupňa*.

Štvorcová matica **A** *n*-tého stupňa, ktorej prvky $a_{ij} = 1$, pre $i = j$ a prvky $a_{ij} = 0$ pre $i \neq j$ sa nazýva *jednotková matica stupňa n*, označujeme ju **E**, resp. $\mathbf{E}_n$. [1]

Vektor je usporiadaná *n*-tica $(a_1, a_2, \dots, a_n) \in R^n$. Vektor môže byť *riadkový* v tvare $\overline{a} = (a_1, a_2, \dots, a_n)$, kde čísla $a_1, \dots, a_n$ sú zložky vektora a n predstavuje počet zložiek, alebo *stĺpcový* v tvare $\overline{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$. [1]

1.2.1 Stupňovitý tvar matice

Prvok a_{ij} matice **A** je vedúci prvok i-teho riadku matice **A**, ak $a_{ij} \neq 0$, a $j = 1$ alebo $a_{il} = 0$ pre všetky $1 \leq l < j$. Inak povedané, *vedúci prvok nenulového riadku je prvý nenulový prvok tohto riadku*. Nulový riadok nemá vedúci prvok. [3]

Matica **A** je v *redukovanom stupňovitom tvare,* ak spĺňa nasledujúce podmienky:

a) Ak $r_i(\mathbf{A}) \neq 0$ a $r_k(\mathbf{A}) = 0$, tak $i < k$; t.j. každý nenulový riadok matice **A** leží nad každým jej nulovým riadkom.
b) Ak a_{ij}, a_{kl} sú vedúce prvky i-teho, resp. k-teho riadku a $i < k$, tak aj $j < l$; t.j. vedúci prvok vyššieho riadku leží viac vľavo než vedúci prvok nižšieho riadku.
c) Ak a_{ij} je vedúci prvok i-teho riadku, tak $a_{ij} = 1$; t.j. vedúci prvok každého nenulového riadku je 1.
d) Ak a_{ij} je vedúci prvok i-teho riadku, tak $a_{kj} = 0$ pre každé $k \neq i$, t.j. v stĺpci, v ktorom sa nachádza vedúci prvok nejakého riadku, sú všetky ostatné prvky rovné 0.

Pokiaľ matica **A** spĺňa iba podmienky a) a b), hovoríme, že je v *stupňovitom tvare*. [3] *Stupňovitý tvar* môžme nazvať aj ako *Gaussov tvar* a *redukovaný stupňovitý tvar* ako *Jordanov tvar.*

Príklady matíc v stupňovitom tvare (**G**) a v redukovanom stupňovitom tvare (**J**):

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

1.2.2 Inverzná matica

Štvorcovú maticu **A** *n*-tého stupňa nazývame *regulárnou*, ak jej hodnosť $h(\mathbf{A}) = n$. Ak $h(\mathbf{A}) < n$, hovoríme, že matica **A** je *singulárna*. Štvorcová matica stupňa *n* je regulárna práve vtedy, ak je ekvivalentná s jednotkovou maticou $\mathbf{E}_n$. [1]

Maticu **B** nazývame *inverznou maticou* k matici **A** stupňa *n*, ak platí:

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A} = \mathbf{E}_n$$

Maticu, ku ktorej existuje inverzná matica, nazývame *invertibilnou maticou*. Inverznú maticu k matici **A** (ak existuje), označujeme $\mathbf{A}^{-1}$. Štvorcová matica **A** stupňa *n* je invertibilná práve vtedy, ak je regulárna. [1]

1.3 Systémy lineárnych rovníc

Systémom *m* lineárnych rovníc s *n* neznámymi nazývame množinu rovníc v tvare

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

kde $x_1, x_2, x_3, \ldots, x_n$ sú neznáme a a_{ij}, b_i sú reálne čísla, pričom $i = 1,2,\ldots,m$ a $j = 1,2,\ldots,n$. Čísla a_{ij} sa nazývajú koeficienty systému rovníc a čísla b_i sú absolútne členy alebo pravé strany systému rovníc. [1]

Systém lineárnych rovníc je *nehomogénny*, ak aspoň jeden absolútny člen $b_i \neq 0$. Systém lineárnych rovníc je *homogénny*, ak $b_i = 0$ pre všetky i=1,2,...,m. To znamená, že homogénny systém má tvar [1]

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= 0 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= 0 \end{aligned}$$

K systému lineárnych rovníc môžeme priradiť dve matice:

a) maticu **A** typu $m \times n$, ktorej prvkami sú koeficienty systému rovníc a nazývame ju maticou systému rovníc

$$\mathbf{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

b) blokovú maticu typu $m \times (n+1)$, ktorej prvkami sú matica systému **A** a stĺpec pravých strán systému $\bar{b}$, teda

$$\overline{\mathbf{A}} = \left(\begin{array}{ccc|c} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} & b_m \end{array}\right) = (\mathbf{A} \mid \bar{b}), \text{ kde } \bar{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in L_m$$

Matica $\overline{\mathbf{A}}$ sa nazýva *rozšírená matica systému lineárnych rovníc* a stĺpcový vektor $\bar{b} \in L_m$ sa nazýva *vektor pravých strán systému lineárnych rovníc*. [1]

Riešením systému lineárnych rovníc je n-tica čísel $(c_1, c_2, c_3, \ldots, c_n)$, ktorá po dosadení za neznáme $x_1, x_2, \ldots, x_n$, do všetkých rovníc systému vytvorí m rovností čísel. *Systém lineárnych rovníc je riešiteľný*, ak má aspoň jedno riešenie. V opačnom prípade hovoríme, že je *neriešiteľný*. [1]

1.3.1 Riešiteľnosť systému lineárnych rovníc

O riešiteľnosti systému rozhoduje Frobeniova veta.

Veta 1.3.1.1 (Frobeniova veta) Systém lineárnych rovníc je riešiteľný práve vtedy, ak hodnosť matice systému A sa rovná hodnosti rozšírenej matice systému $\bar{\mathbf{A}}$, t.j. ak $h(\mathbf{A}) = h(\bar{\mathbf{A}})$. [1]

Z vety 1.3.1.1 pre nehomogénny systém m rovníc s n neznámymi $\mathbf{A} \cdot \bar{x} = \bar{b}$ vyplýva, že:

a) ak $h(\mathbf{A}) = h(\bar{\mathbf{A}})$, tak systém je riešiteľný.

b) ak $h(\mathbf{A}) \neq h(\bar{\mathbf{A}})$, tak systém nie je riešiteľný.

a pre nehomogénny systém $\mathbf{A} \cdot \bar{x} = \bar{0}$ vyplýva, že systém je vždy riešiteľný, pretože $h(\mathbf{A}) = h(\mathbf{A} \mid \bar{0}) = h(\mathbf{A})$.

Hodnosť matice je číslo $h \geq 0$, ktoré udáva maximálny počet lineárne nezávislých vektorov matice **A**. Hodnosť matice určíme elementárnymi úpravami matice na *Gaussov* alebo *Jordanov* tvar.

Frobeniova veta hovorí len o existencii riešenia. Nasledujúca veta umožňuje určiť aj počet riešení v prípade, že systém je riešiteľný.

Veta 1.3.1.2 Nech $\mathbf{A} \cdot \bar{x} = \bar{b}$ je systém *m* lineárnych rovníc s *n* neznámymi. Platí:

a) ak $h(\mathbf{A}) = h(\bar{\mathbf{A}}) = n$, tak systém má jedno riešenie,

b) ak $h(\mathbf{A}) = h(\bar{\mathbf{A}}) < n$, tak systém má nekonečne veľa riešení závislých od n-$h(\mathbf{A})$ parametrov (voľných neznámych).

Z vety 1.3.1.2 vyplýva, že homogénny systém je vždy riešiteľný, pretože vždy platí $h(\mathbf{A}) = h(\bar{\mathbf{A}})$. To znamená, že má vždy jedno nulové riešenie. Okrem nulového riešenia môže mať aj iné riešenia.

Dôsledok vety 1.3.1.2 Nech $\mathbf{A} \cdot \bar{x} = \bar{0}$ je homogénny systém *m* lineárnych rovníc s *n* neznámymi, potom platí:

a) ak $h(\mathbf{A}) = n$, tak systém má jediné riešenie$(x_1 = 0, x_2 = 0, \dots, x_n = 0)$ ktoré sa nazýva triviálne riešenie

b) ak $h(\mathbf{A}) < n$, tak systém má nekonečne veľa riešení závislých od n-$h(\mathbf{A})$ parametrov (voľných neznámych). [1]

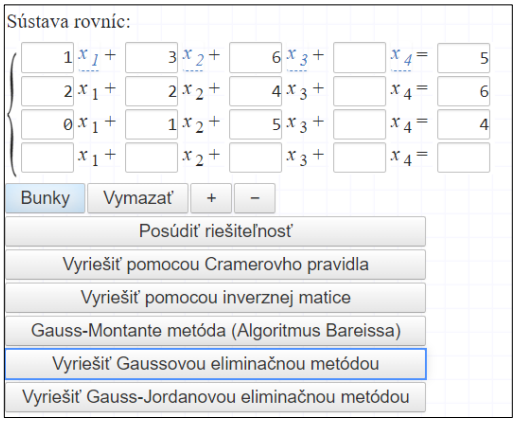
1.4 Aplikácie zamerané na riešenie systémov lineárnych rovníc

Existuje niekoľko aplikácii zameraných na riešenie systémov lineárnych rovníc, pričom väčšina z nich rieši aj iné úlohy lineárnej algebry. Nasledujúce podkapitoly obsahujú analýzu a zhodnotenie troch vybraných webových aplikácii, ktoré riešia systémy *m* lineárnych rovníc s *n* neznámymi pomocou Gaussovej alebo Gauss-Jordanovej eliminačnej metódy.

1.4.1 Matrix calc

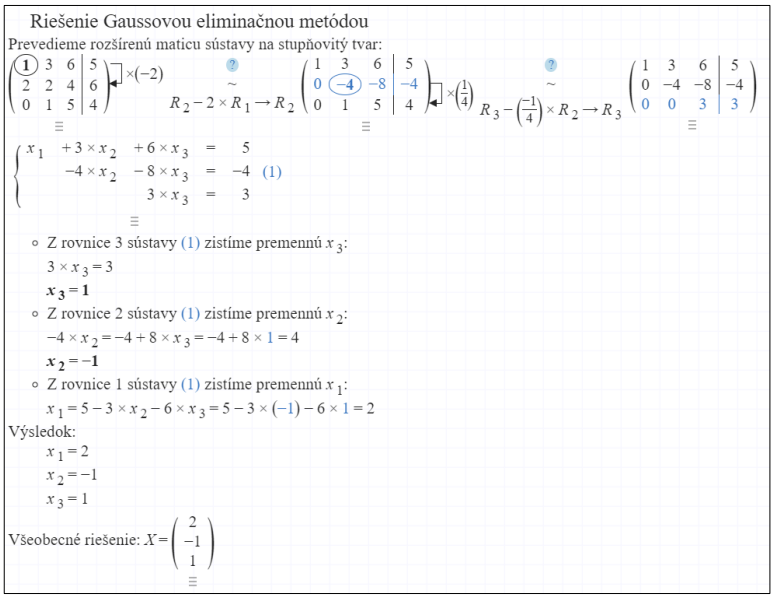
Prvá aplikácia, ktorú budeme analyzovať, sa nachádza na webovej stránke https://matrixcalc.org/sk/slu.html. Je dostupná v 25 jazykoch, vrátane slovenčiny. Okrem riešenia systémov lineárnych rovníc umožňuje riešiť rôzne maticové operácie, výpočet

determinantu a vlastných vektorov. Pre systémy lineárnych rovníc dokáže posúdiť ich riešiteľnosť alebo ich vyriešiť pomocou Gaussovej a Gauss-Jordanovej eliminačnej metódy, inverznej matice a Cramerovho pravidla. Hodnoty koeficientov rovníc sa zadávajú do vopred vygenerovaných polí. Ak majú rovnice menšie rozmery ako počet polí, tieto polia sa ponechajú prázdne. Taktiež je možné nastaviť väčší alebo menší počet polí. Zadávané čísla môžu byť v tvare zlomkov alebo desatinných čísel. Obrázok 1 zobrazuje zadávanie systému rovníc.



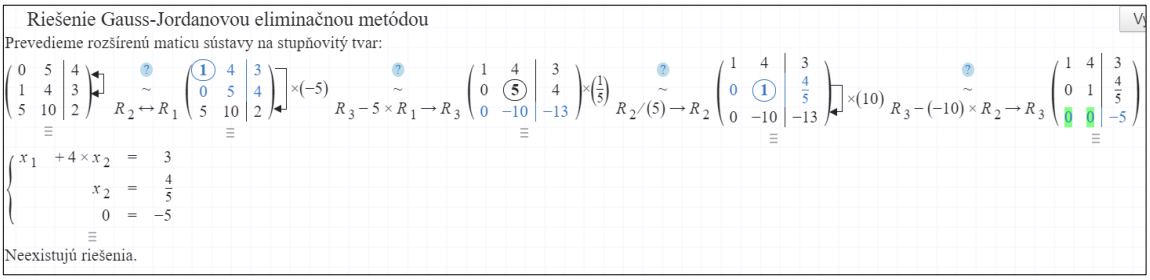
Obrázok 1 - *Matrix calc vstupy* [5]

Po zadaní vstupov a voľbe riešenia Gaussovou eliminačnou metódou sa zobrazí postup s výsledkom systému rovníc, ktorý je na obrázku 2. Tento postup zobrazuje jednotlivé elementárne úpravy hlavne názorne, ale aj slovne. Aplikácia je schopná riešiť aj systémy, ktoré majú nekonečne veľa riešení. V takom prípade vypíše závislosť každej neznámej od ostatných. Po zadaní nového systému rovníc je zachovaná aj história už vyriešených systémov.



Obrázok 2 - *Matrix calc Gaussova metóda výpočet* [5]

Pri riešení Gauss-Jordanovou eliminačnou metódou v prípade neexistujúceho riešenia nie je výsledná matica úplne transformovaná na redukovaný stupňovitý tvar, ako je vidieť z obrázku 3.

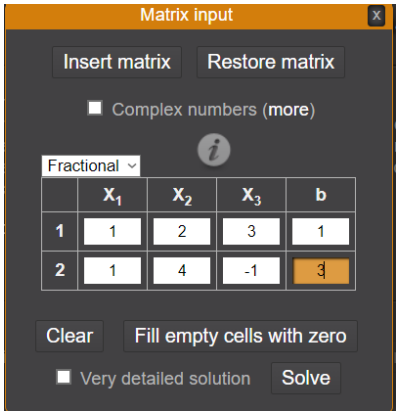


Obrázok 3 - *Matrix calc Gauss-Jordanova metóda výpočet* [5]

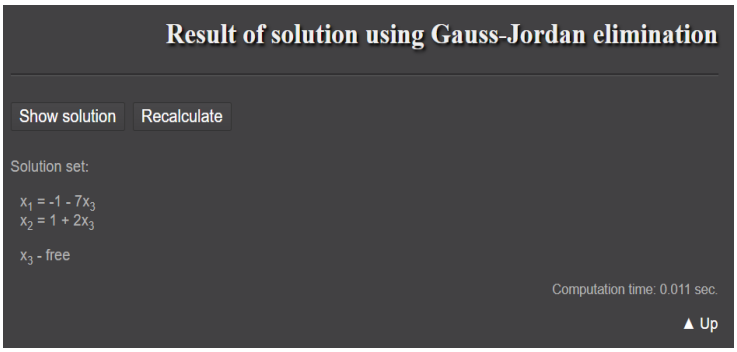
1.4.2 Matrix reshish

Aplikácia sa nachádza na webovej stránke https://matrix.reshish.com. Je dostupná v 10 jazykoch. Umožňuje vykonávať rôzne maticové operácie a riešiť systémy lineárnych rovníc Gauss-Jordanovou eliminačnou metódou, Cramerovým pravidlom a maticovou metódou. Významnou vlastnosťou tejto aplikácie je možnosť používať komplexné čísla pri akejkoľvek metóde a taktiež po použití jednej z metód môžete pokračovať vo výpočte pomocou inej metódy s pôvodnou alebo výslednou maticou. Aplikácia je schopná riešiť systémy s jedným riešením, ako aj systémy, ktoré majú nekonečne veľa riešení.

Pri riešení systémov Gauss-Jordanovou eliminačnou metódou sa v prvom kroku zvolí počet riadkov a stĺpcov rozšírenej matice, pričom najvyšší možný počet je 100. V ďalšom kroku sa zadávajú do nového okna koeficienty rovníc, kde je aj možnosť zvoliť si typ vstupov (zlomky alebo desatinné čísla) a tiež veľmi podrobné riešenie. Zadávanie vstupov a výsledok systému rovníc sú zobrazené na obrázkoch 4 a 5.



Obrázok 4 - *Matrix reshish zadávanie vstupov* [6]



Obrázok 5 - *Matrix reshish zadávanie výsledok* [6]

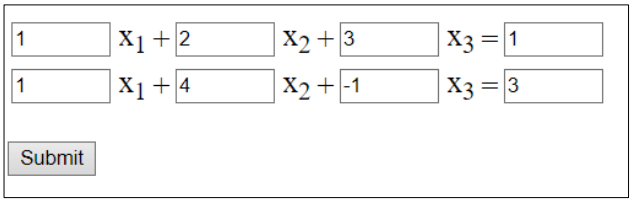
Po zobrazení výsledku zadaného systému je možné vypísať celý postup riešenia, alebo vypočítať systém odznova inou metódou. V prípade, že sme pri zadávaní vstupov nezvolili veľmi podrobné riešenie, vypísaný postup na obrázku 5 je stručnejší. Neobsahuje napríklad informáciu o tom, akým číslom sú riadky násobené a ku ktorým riadkom sú pripočítavané, čo je zahrnuté v podrobnom postupe.

Obrázok 6 - *Matrix reshish postup* [6]

1.4.3 Linear Algebra Toolkit

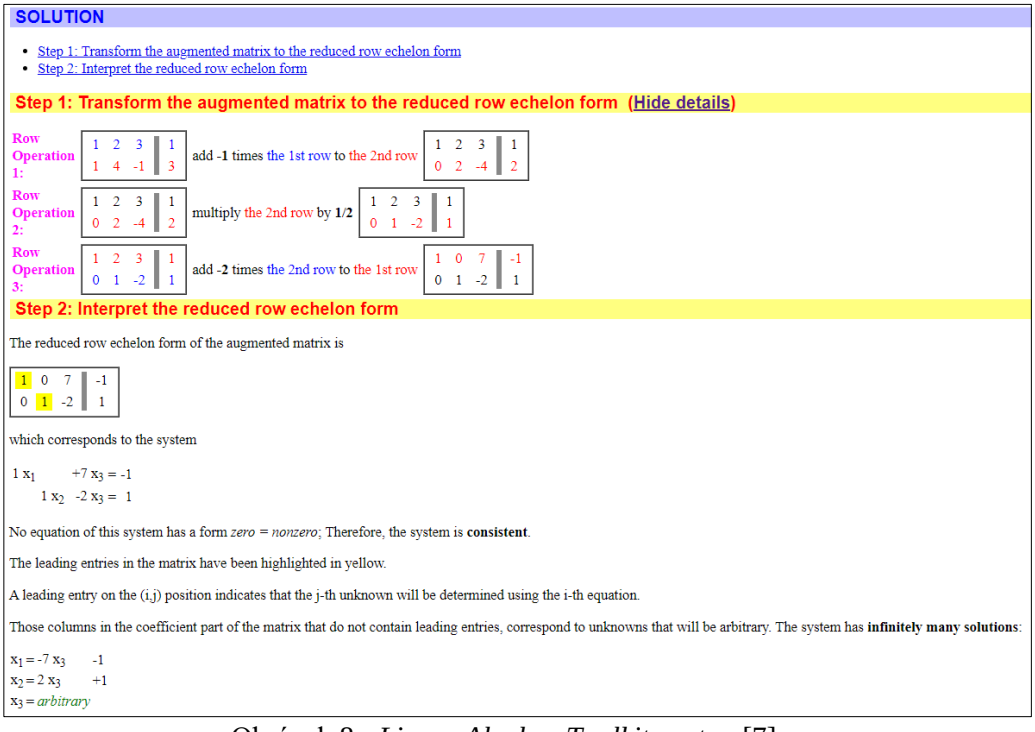
Posledná aplikácia, ktorú budeme analyzovať, sa nachádza na webovej stránke http://www.math.odu.edu/~bogacki/cgi-bin/lat.cgi#bottom. Obsahuje moduly, ktoré majú pomôcť študentom pri učení a precvičovaní úloh lineárnej algebry, ako je riešenie systémov lineárnych rovníc, výpočet determinantu alebo nájdenie lineárnej závislosti vektorov. Modul *Systems of linear equations and matrices* okrem riešenia systémov lineárnych rovníc taktiež umožňuje transformáciu matice na stupňovitý a redukovaný stupňovitý tvar a nájdenie inverznej matice.

Pri riešení systémov lineárnych rovníc sa v prvom kroku zvolí počet rovníc a neznámych. Maximálny počet je 12. V ďalšom kroku sa zadávajú koeficienty rovníc, ktoré sú na obrázku 6.



Obrázok 7- *Linear Algebra Toolkit vstupy* [7]

Systémy rovníc sú riešené pomocou Gauss-Jordanovej eliminačnej metódy. Postup riešenia zadaného systému rovníc, ktorý je na obrázku 7, je dostatočne podrobný. Intepretácia výsledku obsahuje aj vysvetlenie riešiteľnosti a počtu riešení systému.



Obrázok 8 - *Linear Algebra Toolkit postup* [7]

1.4.4 Porovnanie analyzovaných aplikácii

Väčšina webových aplikácii zameraných na riešenie systémov lineárnych rovníc dokáže riešiť iba systémy s rovnakým počtom rovníc a neznámych. Z tohto dôvodu sme vybrali na analýzu aj aplikácie, ktoré riešia systémy iba Gauss-Jordanovou metódou namiesto Gaussovej, ale dokážu riešiť aj systémy *m* rovníc o *n* neznámych. Tabuľka 1 obsahuje porovnanie troch analyzovaných webových aplikácii.

Tabuľka 1- *Porovnanie analyzovaných aplikácii*

Aplikácia	Metódy riešenia	Postup riešenia	Výsledok sústavy	Výhody	Nevýhody
Matrix calc	Gaussova Gauss-Jordanova	podrobné kroky riešenia znázornené aj pomocu šípok a znakov	bez vysvetlenia	všetky funkcie sú v rámci jednej stránky (okna) jednoduché a prehľadné rozhranie história všetkých zadaných rovníc možnosť použiť inú metódu pre zadaný systém po výpočte komplexné čísla	nie je možnosť zvoliť si presný rozmer pre systémy *m*x*n*
Matrix reshish	Gauss-Jordanova	podrobné kroky riešenia	bez vysvetlenia	možnosť použiť inú metódu pre zadaný systém po výpočte komplexné čísla	postup sa nezobrazí hneď s výsledkom podrobný postup je potrebné zvoliť pri zadávaní vstupov
Linear Algebra Toolkit	Gauss-Jordanova	podrobné kroky riešenia farebne znázornené	vysvetlenie riešiteľnosti a počtu riešení	podrobne popísaný postup a vysvetlený výsledok jednoduché a prehľadné rozhranie	starší dizajn v porovnaní s ostatnými aplikáciami

1.5 Java

Java je objektovo orientovaný programovací jazyk. Vychádza z C ++, ktorý je rozšírením jazyka C. Veľká časť vlastností Javy je zdedená z týchto dvoch jazykov. Syntax Javy je odvodená z jazyka C a mnoho objektovo orientovaných funkcií je ovplyvnených jazykom C ++. Zámerom bolo, aby syntax bola známa pre programátorov týchto jazykov. Inkrementálny vývoj s objektovo orientovanými komponentmi v kombinácii s jednoduchosťou jazyka Java umožňuje rýchlo vyvíjať aplikácie a ľahko ich meniť. Štúdie zistili, že vývoj v jazyku Java je rýchlejší ako v jazykoch C alebo C ++. [8] Java bola vytvorená skupinou inžinierov zo spoločnosti Sun Microsystems v roku 1991. Primárnou motiváciou bola potreba platformovo nezávislého jazyka, ktorý by mohol byť využívaný na vytváranie softvéru pre rôzne spotrebné elektronické zariadenia. [9]

Java je kompilovaný a interpretovaný jazyk. Zdrojový kód Javy sa mení na jednoduché binárne inštrukcie, podobne ako obyčajný strojový kód mikroprocesora. Zatiaľ čo zdrojový kód C alebo C ++ je obmedzený na natívne pokyny pre konkrétny model procesora, v Jave je kompilovaný do univerzálneho formátu - inštrukcií pre virtuálny stroj. [10]

Tri najzákladnejšie časti programovacieho prostredia Java sú Java Virtual Machine (JVM), Java Runtime Environment (JRE) a Java Development Kit (JDK).

JVM poskytuje runtime prostredie potrebné na spustenie Java programov. Zjednodušene povedané, JVM funguje ako počítač vo vnútri počítača, ktorý je určený špeciálne na spúšťanie Java programov. Vždy keď sa spustí Java program, vytvorí sa inštancia JVM. Poskytuje bezpečnosť a izoláciu pre spustený program a bráni kolízii s inými programami v systéme. Keď JVM prevezme program na spustenie, zdrojový kód je skonvertovaný do formy známej ako bytecode.

JRE je súbor programov, ktorý obsahuje JVM a knižnice potrebné na spúšťanie programov v JVM. Zahŕňa všetky základné triedy Javy, ako aj knižnice pre interakciu s hostiteľským systémom a pomocné programy.

JDK je balík nástrojov pre vývoj aplikácií a komponentov v jazyku Java. Zahŕňa prostredie JRE, interpreter, kompilátor a ďalšie nástroje potrebné pre vývoj. [11]

Základnou jednotkou Java kódu je trieda. Rovnako ako pri iných objektovo orientovaných jazykoch, triedy sú aplikačné komponenty, ktoré obsahujú spustiteľný kód a údaje. Skompilované triedy sú distribuované v univerzálnom binárnom formáte, ktorý obsahuje bytecode a ďalšie informácie o triede. Trieda môže obsahovať metódy (funkcie), premenné, inicializačný kód a aj iné triedy. Samostatné triedy, ktoré popisujú jednotlivé časti komplexého programu, sú často zoskupené do balíkov (package), ktoré sú užitočné hlavne pri organizácii väčších projektov. Rozhranie (interface) je abstraktná trieda, ktorá sa používa na zoskupenie súvisiacich metód. [10]

Jazyk Java prešiel od svojho vzniku v roku 1996 postupnou revíziou, ale nie kompletnou zmenou. Znamená to, že niektoré z pôvodných návrhov, ktoré boli zavedené na konci 90. rokov, stále ovplyvňujú tento jazyk. Java 8 pridala najradikálnejšie zmeny za takmer desať rokov. Táto verzia bola významnou inováciou jazyka Java z dôvodu zahrnutia novej jazykovej funkcie: lambda výrazov, ktoré zásadne ovplyvnili spôsob písania kódu. Lambda výrazy pridali prvky funkcionálneho programovania do Javy. Môžu zjednodušiť a znížiť množstvo zdrojového kódu potrebného na vytvorenie určitých konštruktov, napríklad niektorých typov anonymných tried. Ďalšou zmenou bolo, že sa do jazyka pridal nový operátor (->). Od roku 2018 sú nové verzie Javy vydávané každých šesť mesiacov, aby sa umožnilo rýchlejšie zavádzanie nových funkcií. [9]

V septembri 2018 bola vydaná Java 11. Je to prvá verzia, ktorá sa považuje za „long term support“ (LTS) verziu, čiže verziu s dlhodobou podporou. Výhodou LTS je, že platiaci zákazníci dostanú pravidelné aktualizácie aj po vydaní ďalšej hlavnej verzie. Pridáva relatívne málo nových funkcií, ktoré sú priamo viditeľné pre vývojárov – sú to napríklad malá zmena syntaxe pre lambda výrazy, nové rozhrania API a schopnosť spúšťať jednosúborové aplikácie bez potreby použitia kompilátora.. Existujú niektoré ďalšie vnútorné zmeny, ale toto vydanie je určené predovšetkým na stabilizáciu. [12]

Knižnice sú ďalšou významnou časťou ekosystému Javy. Knižnica je kolekcia metód a tried, ktoré poskytujú určitú funkcionalitu. Znamená to, že programátori môžu využiť tieto vytvorené metódy vo svojom programe, čo do istej miery uľahčuje proces vývoja. Java obsahuje štandardné knižnice, ktoré sú súčasťou JDK a externé knižnice. [8]

Často využívaná externá knižnica je Apache Commons. Apache Commons je vlastne celý projekt zameraný na vytváranie Java knižníc, ktorý obsahuje viac ako 40

knižníc. Jednou z nich je aj Apache Commons Math. Táto knižnica obsahuje matematické a štatistické komponenty, ktoré nie sú zahrnuté v programovacom jazyku Java. [13] Umožňuje riešiť aj niektoré úlohy lineárnej algebry, ako napríklad operácie s maticami, vektormi a tiež systémy lineárnych rovníc. Dokáže ale riešiť iba systémy s rovnakým počtom rovníc a neznámych a iba v prípade, že majú jediné riešenie.

1.5.1 Vývojové prostredie

Vývojové prostredie je softvér na vytváranie aplikácií, ktorý kombinuje bežné vývojárske nástroje do jedného grafického používateľského rozhrania. Zvyčajne obsahuje editor zdrojového kódu, kompilátor alebo interpreter a debugger. Používateľ píše a edituje zdrojový kód v editore kódu. Kompilátor prekladá zdrojový kód do jazyka, ktorý je čitateľný pre počítač. Debugger testuje kód na nájdenie a vyriešenie chýb. [14]
Najznámejšie vývojové prostredia pre jazyk Java sú NetBeans, Eclipse a IntelliJ IDEA. Na vývoj našej aplikácie sme použili vývojové prostredie IntelliJ IDEA.

IntelliJ IDEA je dostupné v Ultimate a Community verzii. Ultimate je platená verzia, ale pre študentov a učiteľov je zadarmo. Je to plná verzia, ktorá obsahuje všetky funkcie a nástroje tohto vývojového prostredia. Community verzia je bezplatná, ale neobsahuje nástroje na vývoj webových aplikácii a databázové nástroje. Je teda vhodná najmä na vývoj desktopových a android aplikácii. [15]

Významným prvkom väčšiny vývojových prostredí je dokončovanie kódu. Je to funkcia, ktorá analyzuje a navrhuje časti kódu pre vývojárov. IntelliJ IDEA poskytuje inteligentné dokončovanie kódu. Dokáže rozlišovať medzi veľkým množstvom jazykov, ako napríklad Java, Scala, Groovy, SQL, JavaScript. Okrem návrhov premenných, tried a názvov metód dokáže tiež navrhovať metódy, parametre, vlastnosti, názvy súborov a ďalšie. Inteligentné dokončenie kódu navrhuje vety na základe kontextu a potrieb používateľa. [16]

Okrem identifikovania chybných výrazov sa IntelliJ snaží pomôcť vylepšiť vytváraný kód tým, že ho na pozadí analyzuje. V týchto analýzach identifikuje chyby a možné vylepšenia. Druhy analýz, ktoré vykonáva v kóde, sú plne konfigurovateľné - tieto konfigurácie sa nazývajú inšpekcie a sú jednou z najvýkonnejších funkcií tohto vývojového prostredia. Pomocou inšpekcií môže IntelliJ nájsť veľký rozsah existujúcich chýb, možných bugov alebo vylepšení. Napríklad identifikovať premennú, ktorá nie je

využitá alebo navrhnúť zjednodušenie výrazu. [16]

IntelliJ obsahuje širokú škálu nástrojov pre refaktoring. Refaktoring je proces reštrukturalizácie zdrojového kódu bez zmeny jeho správania. Zvyšuje čitateľnosť kódu a redukuje jeho zložitosť. Ak je vnútorná štruktúra kódu lepšia a prehľadnejšia, kód sa ľahšie udržuje a rozširuje. Niekedy je proces refaktoringu náročný na čas, pretože v mnohých súboroch je potrebné urobiť veľa úprav, čo spôsobí, že sa môžu vyskytnúť chyby pri kompilácii. IntelliJ pomáha vykonať refaktoring kódu rýchlejšie a bezpečnejšie. V závislosti od časti kódu automaticky navrhne refaktoring, alebo je možné si vybrať jednu z možností refaktoringu manuálne z ponuky. Medzi nástroje refaktoringu patrí napríklad nájdenie a nahradenie duplicitného kódu, bezpečné vymazanie a zmena dátového typu prvku v celom projekte. [17]

1.5.2 Grafické používateľské rozhranie

Grafické používateľské rozhranie (GUI - skratka pre Graphical User Interface) je pojem používaný nielen v Jave, ale vo všetkých programovacích jazykoch, ktoré podporujú vývoj GUI. Grafické používateľské rozhranie predstavuje pre používateľov vizuálne zobrazenie aplikácie. Pozostáva z grafických komponentov (napr. tlačidiel, ikon, okien), prostredníctvom ktorých môže používateľ komunikovať so stránkou alebo aplikáciou. [18]

Programy s grafickým používateľským rozhraním sú riadené udalosťami. To znamená, že program reaguje na akcie používateľa - tieto akcie sa nazývajú udalosti (eventy). Príklady udalostí sú stlačenie klávesy, pohyb myši, stlačenie tlačidla a výber položky z menu. V programe je potrebné špecifikovať, na ktoré udalosti musí program reagovať a aká by mala byť reakcia. V Jave sa to vykonáva implementáciou tzv. listeners (poslucháčov), ktorí čakajú na nastanie špecifických druhov udalostí. Ak je implementovaný listener pre konkrétny typ udalosti, runtime systém automaticky informuje listenera o tejto udalosti. Listener potom vykoná požadovanú akciu. Udalosti sa spracúvajú v poradí, v akom nastali. Existuje viacero typov udalostí v Jave, ktoré sú spojené s rôznymi zdrojmi, akými sú tlačidlá, menu alebo myš. Udalosti obsahujú informácie o tom, čo spustilo udalosť a kde. [18]

Na vytvorenie grafického používateľského rozhrania v jazyku Java sa používa Swing alebo JavaFX. Naša aplikácia bola vytvorená s použitím nástroja Swing.

Swing je nástroj na vytvorenie GUI v jazyku Java. Je to kolekcia tried a rozhraní, ktoré obsahujú bohatú sadu vizuálnych komponentov, ako sú napríklad tlačidlá, textové polia, posuvníky, zaškrtávacie polia, tabuľky a ďalšie. Spoločne môžu byť tieto ovládacie prvky použité na vytvorenie výkonných a ľahko použiteľných grafických rozhraní. Swing je vlastne rozšírenie staršej knižnice AWT. Abstract Window Toolkit (AWT) je pôvodný platformovo závislý nástroj na tvorbu GUI v Jave. Swing je postavený na základných knižniciach AWT, s pridaním veľkého množstva komponentov a iných prvkov. Swing komponenty sú písané v Jave a preto je Swing platformovo nezávislý. [19] Základnými komponentami sú ovládacie prvky ako napríklad:

- Button – tlačidlo, pomocou ktorého sa zachytávajú udalosti
- Label – priestor na zobrazenie textu alebo obrázkov
- TextField – textové pole na zadávanie jednoriadkového textu
- Table – tabuľka, slúži na zobrazenie údajov v riadkoch a stĺpcoch

Tieto komponenty nemôžu byť zobrazené samostatne, ale musia byť súčasťou kontajneru. Takýmto kontajnerom je napríklad Panel, ktorý slúži predovšetkým ako schránka na umiestnenie iných komponentov. Je súčasťou najvyššieho kontajneru - Frame, ktorý predstavuje okno aplikácie. Kontajner je grafický komponent, ktorý v sebe môže držať a kresliť ostatné grafické komponenty. Akým spôsobom sú komponenty v kontajneri rozložené, určuje tzv. správca rozloženia (LayoutManager).

1.5.3 Výhody jazyka Java

- *Objektovo orientované programovanie* - objektovo orientované programovanie je spojené s pojmami ako trieda, objekt, dedičnosť, zapuzdrenie, abstrakcia a polymorfizmus, ktoré umožňujú vytvárať modulárne programy a znovu použiteľný kód. Výhodami sú hlavne možnosť znova použiť objekty v iných programoch; zabraňovanie chybám tým, že objekty skryjú informácie, ku ktorým by nemal byť ľahký prístup a prehľadnejšie programy.
- *Jednoduchosť* - Java je vysokoúrovňový programovací jazyk, čo znamená, že sa úzko podobá ľudskému jazyku. Na rozdiel od jazykov nižšej úrovne, ktoré sa podobajú strojovému kódu, musia byť vysokoúrovňové jazyky konvertované pomocou kompilátorov alebo interpreterov. Tým sa uľahčuje písanie a čítanie jazyka, čím sa zjednodušuje vývoj.

- *Platformová nezávislosť* – platformová nezávislosť znamená, že je možné vytvoriť Java program na platforme Windows, kompilovať ho do bytecode a spustiť tento program na akejkoľvek inej platforme, ktorá podporuje JVM, napríklad Mac OS alebo Linux. V tomto prípade JVM slúži ako úroveň abstrakcie medzi kódom a hardvérom.[20]
- *Automatická správa pamäte* – v Jave nie je potrebné manuálne písanie kódu pre správu pamäte vďaka automatickej správe pamäte a tzv. garbage collectora, čo je aplikácia, ktorá automaticky zaisťuje alokáciu a dealokáciu pamäte. Garbage collector dokáže vyhľadať objekty, na ktoré už program neodkazuje a odstrániť ich. Je to tiež jedna z funkcií, vďaka ktorej sú programy napísané v jazyku Java menej náchylné na chyby ako v jazykoch, ktoré nepodporujú túto funkciu, napr C++.
- *Multithreading* – multithreading je schopnosť programu vykonávať niekoľko úloh súčasne. V Jave je táto schopnosť integrovaná, zatiaľ čo v iných jazykoch je potrebné zavolať procedúry špecifické pre operačný systém, aby sa umožnil multithreading. [8]

1.5.4 Nevýhody jazyka Java

- *Výkon* – spustenie programov v Jave o niečo pomalšie a náročnejšie na pamäť ako v natívne kompilovaných jazykoch, napríklad C alebo C ++. Väčšina vysokoúrovňových jazykov vrátane Javy sa musí vyrovnať so slabším výkonom v dôsledku kompilácie a abstrakcie virtuálneho stroja. Ďalším dôvodom je napríklad aj spomínaný garbage collector, čo je užitočná funkcia, ktorá ale môže viesť k problémom s výkonom.
- *Komplexný kód* – komplexný kód znamená, že obsahuje veľa slov, keďže syntax v Jave sa snaží napodobniť prirodzený jazyk. Vďaka tomu je jazyk transparentnejší a jednoduchší na naučenie, ale menej kompaktný.
- *Platená komerčná licencia* - od roku 2019 je potrebné platiť za Java Standard Edition 8 pri použití na obchodné a komerčné účely. Na získanie všetkých nových aktualizácií a opráv musia zákazníci zaplatiť podľa počtu používateľov alebo procesorov. [20]

2 Cieľ práce

Hlavný cieľ práce:

Cieľom záverečnej práce je vytvoriť program v jazyku Java, ktorý rieši systémy *m* lineárnych rovníc s *n* neznámymi využitím Gaussovej eliminačnej metódy.

Čiastkové ciele práce:

1. Charakteristika aplikácii zameraných na riešenie systémov lineárnych rovníc a ich porovnanie
2. Výber vývojového prostredia a technológie na tvorbu používateľského rozhrania v Jave
3. Vytvorenie a implementácia algoritmu Gaussovej eliminačnej metódy a vývoj používateľského rozhrania
4. Komparácia aplikácie s podobnými existujúcimi aplikáciami

3 Metodika práce a metódy skúmania

V tejto kapitole sú charakterizované metódy riešenia systémov lineárnych rovníc a metodika vývoja aplikácie, kde je bližšie popísaná fáza špecifikácie požiadaviek a návrhu aplikácie.

3.1 Metódy riešenia systémov lineárnych rovníc

Existuje niekoľko metód riešenia systémov lineárnych rovníc. Pre systémy s menším počtom rovníc a neznámych sa najčastejšie používa sčitovacia alebo dosadzovacia metóda. Pre väčšie systémy môžeme použiť eliminačné metódy – Gaussovu alebo Gauss-Jordanovu eliminačnú metódu, na ktoré sa zameriavame v našej práci.

3.1.1 Gaussova eliminačná metóda

Gaussova eliminačná metóda je jedna z najznámejších a najpoužívanejších metód na riešenie systémov lineárnych rovníc.

Princíp tejto metódy spočíva vo vykonávaní postupných úprav rozšírenej matice systému na Gaussov tvar, z ktorého vieme zistiť jeho riešenie. Tieto úpravy sa nazývajú *elementárne riadkové úpravy*:

- výmena riadkov,
- vynásobenie riadku nenulovým číslom,
- pripočítanie násobku riadku k inému riadku. [2]

Elementárne riadkové úpravy použité na rozšírenej matici nezmenia riešenie zodpovedajúceho systému rovníc. Pomocou týchto úprav transformujeme rozšírenú maticu na Gaussov tvar. [2]

Na prevedenie rozšírenej matice na Gaussov tvar použijeme nasledujúci algoritmus:

1. Nájdeme prvý nenulový stĺpec zľava. Prvok na najvyššej pozícii v tomto stĺpci je vedúci prvok. Ak je to potrebné, vymeníme riadky tak, aby v pozícii vedúceho prvku bolo nenulové číslo.
2. Použijeme elementárne riadkové úpravy na vynulovanie prvkov v stĺpci pod vedúcim prvkom.

3. Zopakujeme kroky 1 a 2 pre zvyšné riadky s tým, že ignorujeme riadok, v ktorom sa nachádzal vedúci prvok. Opakujeme postup dovtedy, kým nezostanú žiadne ďalšie riadky na úpravu. Výsledkom je matica v Gaussovom tvare. [2]

4. V prípade, že systém je riešiteľný, na zistenie hodnôt neznámych realizujeme spätný chod Gaussovej eliminačnej metódy. Vyriešime rovnicu s jednou neznámou a získanú hodnotu dosadíme za túto neznámu do ďalšej rovnice, ktorú vyriešeime a získame tak hodnotu ďalšej neznámej. Tento postup opakujeme, kým nezistíme hodnoty všetkých neznámych.

3.1.2 Gauss-Jordanova eliminačná metóda

Druhá eliminačná metóda, nazývaná Gauss-Jordanova eliminačná metóda je rozšírením Gaussovej eliminačnej metódy. Pomocou elementárnych riadkových úprav transformujeme rozšírenú maticu systému na Jordanov tvar.

Použijeme rovnaký algoritmus ako pri Gaussovej eliminačnej metóde, rozdiel je v dvoch pridaných krokoch:

1. Vydelíme každý nenulový riadok vedúcim prvkom tak, aby mal vedúci prvok hodnotu 1.

2. Použijeme elementárne riadkové úpravy na vynulovanie prvkov v stĺpci nad vedúcim prvkom. Výsledkom je matica v Jordanovom tvare. [2]

Túto metódu použijeme aj na výpočet inverznej matice. Ak maticu **A** stupňa *n* upravíme pomocou postupnosti elementárnych riadkových úprav na jednotkovú maticu, tak matica **A** je invertibilná a tou istou postupnosťou elementárnych úprav jednotkovej matice dostaneme inverznú maticu $\mathbf{A}^{-1}$ k matici **A**. [1]

3.2 Metodika vývoja aplikácie

Pri vývoji aplikácie sme použili inkrementálny prístup vývoja. Inkrementálny vývoj je založený na myšlienke vytvoriť počiatočnú implementáciu aplikácie, získať spätnú väzbu od používateľa a rozvíjať túto implementáciu prostredníctvom ďalších verzií, až kým sa nevytvorí požadovaný systém. Inkrementálny vývoj softvéru je základnou súčasťou agilných prístupov. Tento prístup odráža spôsob, akým riešime problémy. Málokedy vypracujeme kompletné riešenie problému vopred, ale smerom k riešeniu postupujeme

v niekoľkých krokoch a keď zistíme, že sme urobili chybu, vrátime sa späť. Pri inkrementálnom vývoji softvéru je jednoduchšie a výhodnejšie vykonávať zmeny počas vývoja. Funkcionalita systému je rozdelená na prírastky (inkrementy) a každý prírastok alebo verzia systému obsahuje niektoré z požadovaných funkcií. Počiatočné prírastky systému vo všeobecnosti zahŕňajú najdôležitejšiu alebo najnutnejšiu požadovanú funkcionalitu. To znamená, že používateľ môže vyhodnotiť systém v relatívne skorom štádiu vývoja, aby zistil, či poskytuje všetko čo je vyžadované. Ak nie sú splnené všetky požiadavky, potom je potrebné zmeniť iba aktuálny prírastok a prípadne môže byť definovaná nová funkcionalita pre neskoršie prírastky. [21]

Proces vývoja softvéru pozostáva z postupnosti viacerých činností s cieľom špecifikovať, navrhovať, implementovať a testovať softvérový systém. Základné činnosti vývoja sú usporiadané odlišným spôsobom pri rôznych vývojových modeloch. Vo vodopádovom modeli sú usporiadané postupne, zatiaľ čo pri inkrementálnom vývoji môžu byť tieto činnosti vykonávané striedavo. Spôsob, akým sú tieto činnosti vykonávané, závisí od typu vyvíjaného softvéru a skúseností a schopností vývojárov. Proces vývoja softvéru pozostáva z nasledujúcich fáz:

1. Špecifikácia – fáza špecifikácie softvéru zahŕňa niekoľko krokov. Prvým krokom je získavanie požiadaviek, ktoré väčšinou prebieha diskusiou s potenciálnymi používateľmi aplikácie. Následne sa získané požiadavky analyzujú, aby sa zistilo, či sú jednoznačne a dostatočne definované. V ďalšom kroku sa informácie získané pri analýze požiadaviek zapisujú do dokumentu, ktorý definuje súbor požiadaviek. V poslednom kroku sa overuje, či sú požiadavky realizovateľné a konzistentné.
2. Návrh – návrh predstavuje popis štruktúry softvéru, dátových modelov, systémových komponentov a návrh používateľského rozhrania.
3. Implementácia - po dokončení požiadaviek a návrhu je ďalšou fázou implementácia alebo vývoj softvéru. V tejto fáze sa začne programovanie softvéru vo vybranom programovacom jazyku podľa požiadaviek a návrhu definovaných v predchádzajúcich fázach.
4. Validácia - cieľom validácie je nájsť chyby v systéme a overiť, či sa aplikácia správa podľa očakávania a vyhovuje všetkým špecifikáciám. Hlavnou metódou validácie je testovanie. Pri testovaní sa môžu zistiť chyby v systéme, ktoré sa opravia a vytvorí sa nová verzia softvéru, ktorú je potrebné znova overiť. Tento cyklus sa opakuje, až kým všetky požiadavky nie sú otestované a všetky chyby odstránené.

5. Nasadenie a údržba – po odstránení chýb a overení, že softvér spĺňa všetky požiadavky, je softvér nasadený do prevádzky, alebo v našom prípade poskytnutý používateľom. Pri väčších softvérových projektoch nasleduje fáza údržby, ktorá zahrňuje monitorovanie výkonnosti softvéru a poskytovanie aktualizácii a opráv.

3.2.1 Špecifikácia požiadaviek

Pred začatím vývoja sme si určili funkčné a nefunkčné požiadavky, ktoré musí naša aplikácia spĺňať. Funkčné požiadavky špecifikujú funkcie, ktoré vytváraný systém musí byť schopný vykonávať, aby splnil požiadavky používateľov. Nefunkčné požiadavky popisujú, akým spôsobom by mal systém fungovať.

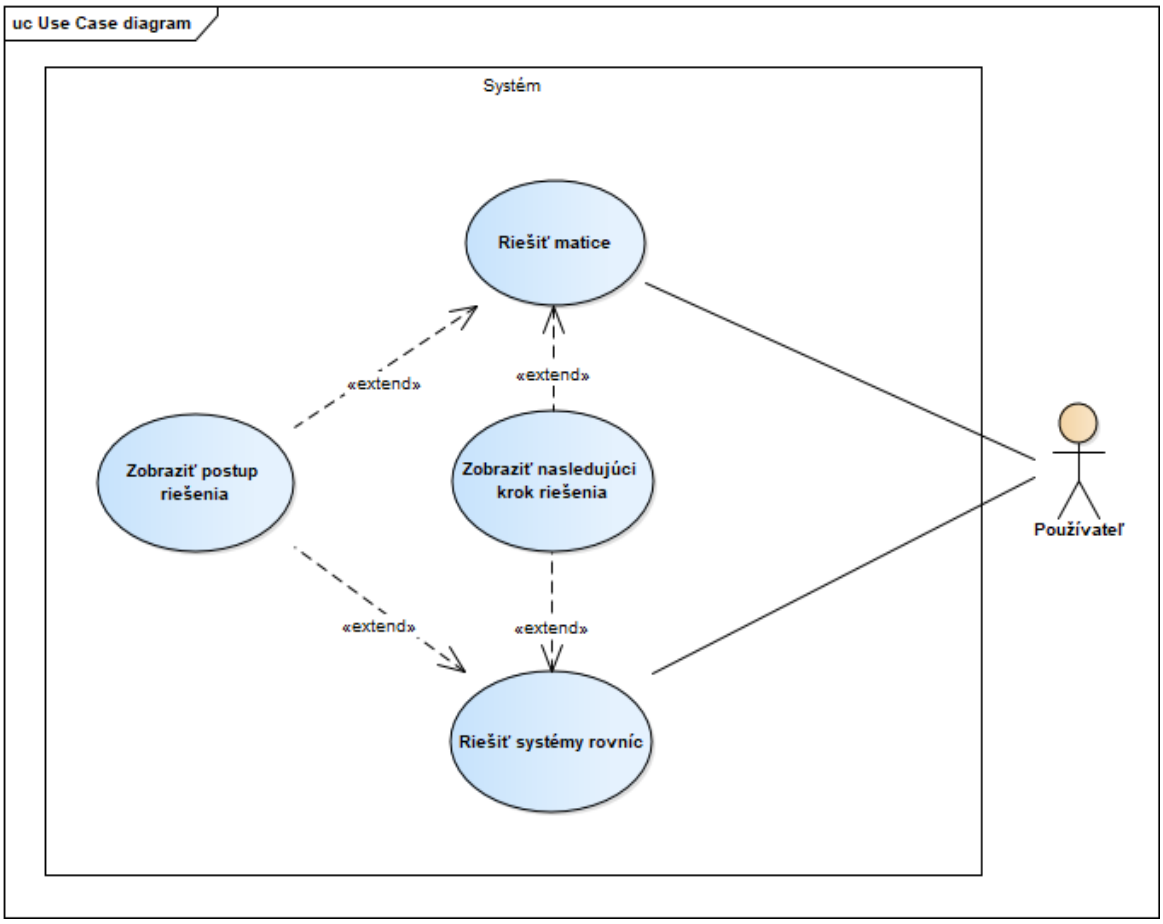
Funkčné požiadavky:

- zadávanie rozmeru a koeficientov systému rovníc
- voľba ekvivalentných úprav
- zobrazenie nasledujúceho kroku riešenia
- zobrazenie výsledku riešenia

Nefunkčné požiadavky:

- jednoduché ovládanie
- názornosť a dobrá čitateľnosť výstupov
- zabezpečenie funkčnosti v prípade nevhodných vstupov

Pri špecifikácii požiadaviek je vhodné vytvoriť diagram prípadov použitia (Use Case diagram). Prípady použitia vyjadrujú spôsob interakcie medzi používateľmi a systémom pomocou grafického modelu. Súbor prípadov použitia predstavuje možné interakcie, ktoré sú definované v požiadavkách. Aktérmi môžu byť ľudia alebo iné systémy. Interakcie sú reprezentované ako pomenované elipsy. Každý prípad použitia by mal byť zdokumentovaný textovým popisom, prípadne aj scenárom. Na obrázku 9 je zobrazený Use Case diagram našej aplikácie.



Obrázok 9 - *Use Case diagram*

V Use Case diagrame je jediným aktérom „Používateľ“, keďže všetci používatelia využívajú aplikáciu rovnakým spôsobom. Prípady použitia zobrazujú jednotlivé funkcie, ktoré môže používateľ v aplikácii vykonávať. Nasledujúca tabuľka obsahujú scenár prípadu použitia *Riešiť systémy rovníc*. Scenár prípadu použitia *Riešiť matice* nebudeme uvádzať, keďže obsahuje kroky, ktoré sú približne rovnaké ako pri systémoch rovníc.

Tabuľka 2 - *Prípad použitia Riešiť systémy rovníc*

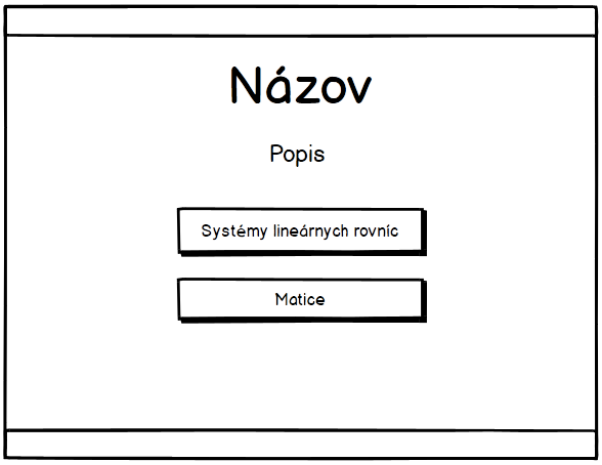
Use Case	Riešiť systémy rovníc		
Vstupná podmienka	Používateľ si zvolil jednu z metód riešenia systémov rovníc		
Výstupná podmienka	Vyriešenie zadaného systému rovníc		
Hlavný tok	**Krok**	**Rola**	**Akcia**
	1	Aktér	Zadá počet rovníc a neznámych a potvrdí
	2	Systém	Vygeneruje textové polia na zadávanie koeficientov
	3	Aktér	Zadá koeficienty rovníc a potvrdí
	4	Systém	Zobrazí zadaný systém rovníc a možnosti riešenia
	5	Aktér	Zvolí ekvivalentnú úpravu a zadá potrebné vstupy
	6	Systém	Vykoná ekvivalentnú úpravu a zobrazí výstup, v prípade konečného kroku aj výsledok
Alternatívny tok	**Krok**	**Rola**	**Akcia**
	5.1	Aktér	Zvolí zobrazenie nasledujúceho kroku
	5.2	Systém	Zobrazí nasledujúci krok riešenia, v prípade konečného kroku aj výsledok
Tok výnimiek	**Krok**	**Rola**	**Akcia**
	1.1	Aktér	Nezadá počet rovníc alebo neznámych a potvrdí
	1.2	Systém	Upozorní na chýbajúce vstupy
	2.1	Aktér	Zadá nesprávne hodnoty koeficientov a potvrdí
	2.2	Systém	Upozorní na nesprávne hodnoty vstupov
	5.1	Aktér	Zvolí ekvivalentnú úpravu a zadá nesprávne vstupy
	5.2	Systém	Upozorní na nesprávne hodnoty vstupov

3.2.2 Návrh používateľského rozhrania

Po špecifikácii všetkých požiadaviek je ďalším krokom návrh rozhrania a obsahu jednotlivých obrazoviek aplikácie. Na zobrazenie základnej štruktúry používateľského rozhrania sa používa *wireframe*. Wireframe je jednoduchá vizuálna reprezentácia rozvrhnutia prvkov aplikácie. Cieľom je zobraziť jednotlivé komponenty aplikácie, ich vzájomné usporiadanie a vzťahy. Nezobrazuje konkrétne grafické riešenia ani farby jednotlivých prvkov aplikácie. Na vytvorenie wireframu sme použili nástroj Balsamiq Wireframes.

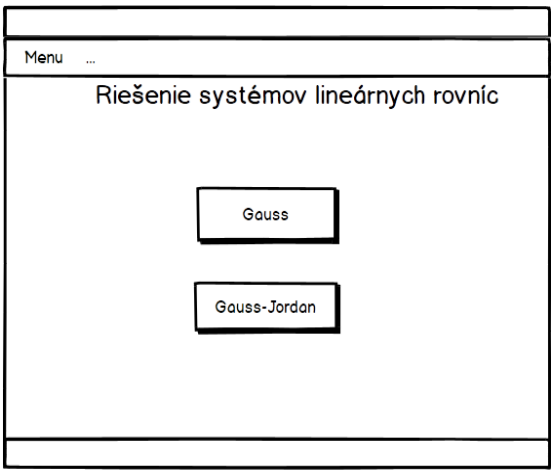
Úvodné okno obsahuje názov a popis aplikácie a dve tlačidlá. Tieto tlačidlá predstavujú dve témy, ktoré aplikácia rieši:

- systémy lineárnych rovníc,
- matice

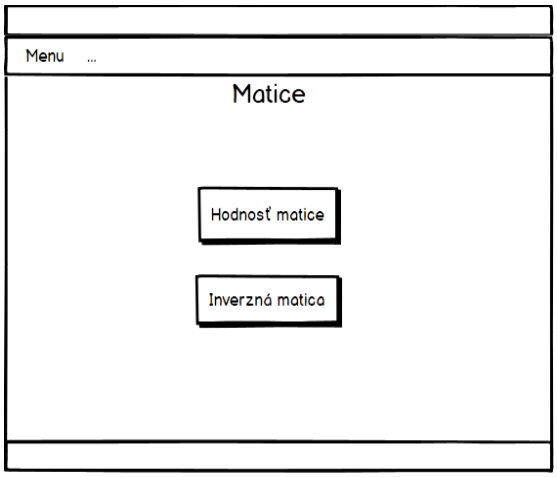


Obrázok 10 - *Wireframe úvodná obrazovka*

Po zvolení jednej z úloh sa zobrazí nové okno, ktoré je na obrázkoch 11 a 12. Toto okno obsahuje ďalšie dve tlačidlá. Pri systémoch lineárnych rovníc sú to tlačidlá, ktoré umožňujú výber metódy riešenia a pri maticiach výber typu úlohy.

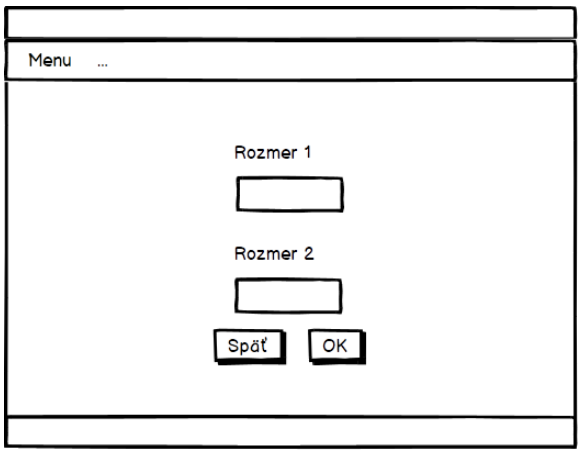


Obrázok 11 - *Wireframe Systémy lineárnych rovníc*

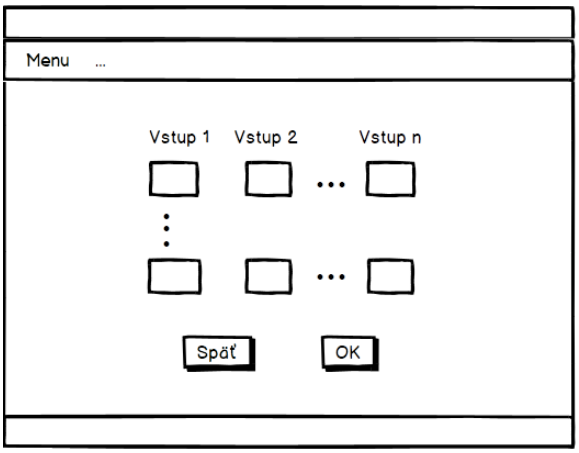


Obrázok 12- *Wireframe Matice*

Po zvolení metódy riešenia alebo typu úlohy sa zobrazí okno, ktoré je na obrázku 13. V tomto okne sa zadávajú rozmery systému rovníc alebo matice, v závislosti od vybranej úlohy. Po zadaní rozmerov sa v ďalšom okne zadávajú vstupné hodnoty do textových polí. Tieto okná taktiež obsahujú tlačidlo *Späť*, ktoré umožňuje návrat do predchádzajúceho okna a tlačidlo *OK*, ktorým sa potvrdia zadané vstupy.

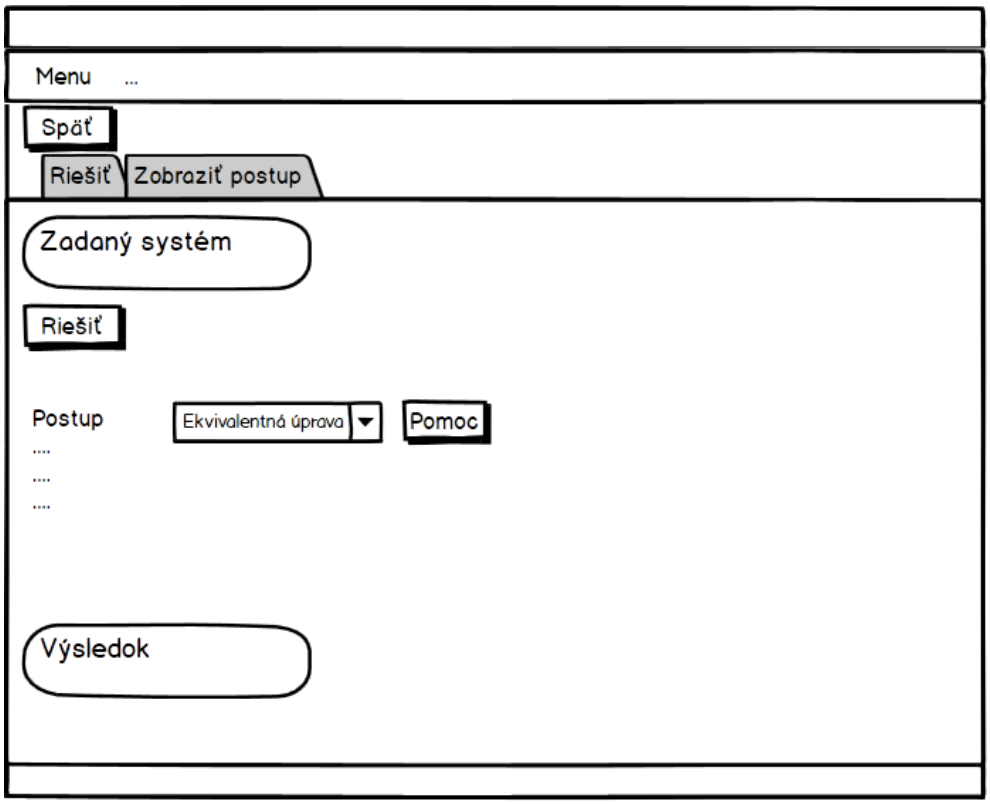


Obrázok 13 - *Wireframe zadávanie rozmerov*



Obrázok 14 - *Wireframe zadávanie hodnôt*

Po zadaní vstupov sa zobrazí posledné okno, v ktorom sú dve záložky. V prvej sa rieši zadaná úloha a v druhej je možné zobraziť vypočítaný postup. Úlohu je možné riešiť pomocou ekvivalentných úprav, ktoré sú na výber v kombinovanom poli, alebo kliknutím na tlačidlo *Pomoc*, ktoré zobrazí nasledujúci krok riešenia. Po poslednom kroku riešenia sa zobrazí výsledok. Všetky okná okrem úvodného taktiež obsahujú *Menu*, pomocou ktorého je možné sa vrátiť na začiatok alebo si zvoliť novú úlohu.



Obrázok 15 - *Wireframe riešenie*

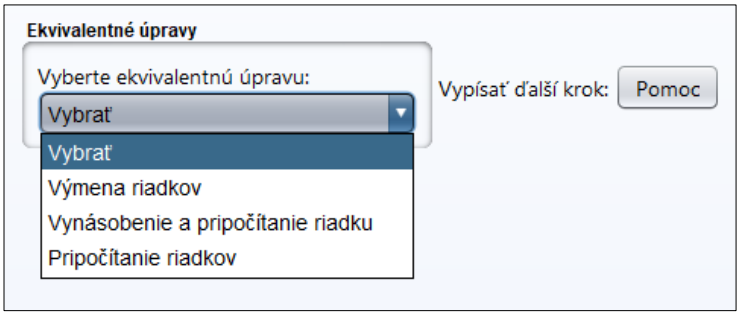
4 Výsledky práce a diskusia

V tejto kapitole sa budeme venovať implementácii aplikácie. Uvedieme niektoré dôležité metódy a algoritmy a ukážku všetkých funkcií aplikácie na konkrétnych príkladoch.

4.1 Implementácia aplikácie

Aplikácia rieši dve kategórie úloh lineárnej algebry. Hlavnou kategóriou sú *Systémy lineárnych rovníc*. Obsahuje aj doplnok *Matice*, ktorý umožňuje riešiť hodnosť matice a inverznú maticu.

Prvým krokom po výbere úlohy je zadávanie rozmerov systému alebo matice. Rozmery sa zadávajú do textových polí, ktoré umožňujú zadávať iba celé čísla. V prípade, že používateľ nevyplní jedno alebo žiadne z textových polí a klikne na tlačidlo pre potvrdenie údajov, zobrazí sa chybové hlásenie. Chybové hlásenie informuje o tom, čo je potrebné vykonať, aby táto chyba nenastala. Pri systémoch lineárnych rovníc sa zadáva počet rovníc a počet neznámych. Pri hodnosti matice je to počet riadkov a počet stĺpcov matice a pri inverznej matici sa zadáva iba jeden rozmer, keďže matica musí byť štvorcová. Po úspešnom zadaní rozmerov sa zobrazí nové okno, v ktorom sú vygenerované textové polia na zadávanie koeficientov systému alebo prvkov matice podľa zadaných rozmerov. V prípade systémov lineárnych rovníc sa vygeneruje jeden stĺpec textových polí navyše pre pravú stranu systému. Do týchto textových polí je možné zadávať iba celé čísla alebo zlomky v tvare 1/2. Hodnoty v týchto poliach sú štandardne nastavené na 0. V prípade, že používateľ zadá nesprávny tvar vstupov a potvrdí ich pomocou tlačidla, zobrazí sa chybové hlásenie. Po úspešnom zadaní vstupov sa hodnoty uložia do dvojrozmerného poľa, ktoré predstavuje maticu. V poslednom okne je možné vidieť zadaný systém/maticu a tlačidlo *Riešiť*, ktoré po kliknutí zobrazí panel s ekvivalentnými úpravami. Následne sa toto tlačidlo zmení na tlačidlo *Riešiť odznova,* ktoré umožňuje začať riešiť zadaný príklad odznova v novej záložke, pričom doteraz vytvorený postup zostane zachovaný. Riešenie úlohy je realizované pomocou výberu ekvivalentných úprav matice alebo zobrazením ďalšieho kroku riešenia. Ekvivalentné úpravy pre Gaussovu eliminačnú metódu sú zobrazené na obrázku 16. Gauss-Jordanova metóda obsahuje jednu úpravu navyše – vydelenie riadku.



Obrázok 16 - *Ekvivalentné úpravy*

Po kliknutí na jednu z možností sa zobrazia textové polia pre zadávanie vstupov potrebných na vykonanie úpravy matice a tlačidlo *OK* na potvrdenie vstupov. Pri výmene riadkov je potrebné zadať čísla riadkov, ktoré chceme vymeniť. V prípade násobenia a pripočítania riadku sa zadávajú čísla riadkov a číslo, ktorým bude vybraný riadok vynásobený. Pri pripočítavaní riadkov je potrebné zadať čísla riadkov, ktoré chceme pripočítať a pri delení číslo riadku a číslo, ktorým chceme vydeliť daný riadok. Textové polia určené na zadávanie čísel riadkov umožňujú zadávať iba celé čísla a sú zabezpečené aj pre prípad zadania neexistujúcich alebo rovnakých riadkov matice. V takomto prípade sa zobrazí chybové hlásenie podobne ako pri zadávaní vstupov v predchádzajúcich oknách. Textové pole pre zadávanie čísel, ktorými chceme vynásobiť alebo vydeliť riadok, umožňuje zadávať iba celé nenulové čísla. Po zadaní potrebných údajov a úspešnom potvrdení vybranej úpravy bude táto úprava vykonaná na matici a následne sa upravená matica zobrazí v okne ako ďalší krok postupu. Ak používateľ nevie, aká ďalšia ekvivalentná úprava by mala nasledovať, je možné pomocou tlačidla *Pomoc* zobraziť nasledujúci krok riešenia alebo celý postup. Riešenie je realizované pomocou algoritmov Gaussovej a Gauss-Jordanovej eliminačnej metódy. Implementácia týchto metód bude vysvetlená v nasledujúcich podkapitolách.

4.1.1 Gaussova eliminačná metóda v Jave

Na implementáciu algoritmu Gaussovej eliminačnej metódy sme použili nasledujúcu metódu:

```
public void gaussEliminacia (int m, int n, Double[][] A) {
    int i, j, k,l = 0;
    for (i = 0; i < m; i++) {
        l = i;
        //ak prvok je 0
        if (A[i][i] == 0) {
            //skontroluj ci je v stlpci pod prvkom 1 alebo -1
            if (check1(i,m,A)){
                k = return1(i,m,A);
                vymen(i,k,n, A);
```

```
            }
            //ak nie je, tak vymen s prvym nenulovym prvkom
            else{
                for (k = i + 1; k < m; k++) {
                    if  (A[k][i] !=0) {
                        vymen(i,k,n, A);
                        break;
                    }
                }
            }
            //pokial prvok je stale 0
            while (A[i][l] == 0 && l < n-2){
                l++;
            if (check1(l,m,A)){
                    k = return1(l,m,A);
                    vymen(i,k,n, A);
                }
            else{
                for (k = i + 1; k < m; k++){
                    if (A[i][l] == 0 && A[k][l] !=0){
                        vymen(i, k, n, A);
                    }
                }
            }
        }
    }
    //ak veduci prvok nie je 1 alebo -1
    if (A[i][i] != 1 && A[i][i] != -1) {
        if (obj.check1(i, m, A)) {
            k = obj.return1(i, m, A);
            vymen(i, k, n, A);
        }
    }
    //eliminacia
    for (k = i + 1; k < m; k++) {
        double podiel = -(A[k][l] / A[i][l]);

        if (podiel == 1){
            for (j = 0; j < n; j++) {
                A[k][j] = A[k][j] + A[i][j];
            }
        }
        else if (podiel != 0) {
            for (j = 0; j < n; j++) {
                A[k][j] = A[k][j] + podiel * A[i][j];
            }
        }
    }
    if(l == n-2){
        break;
    }
}

}
```

Premenná *m* predstavuje počet riadkov matice, *n* je počet stĺpcov matice a pole *A* predstavuje maticu. Metóda obsahuje hlavný cyklus *for*, ktorý sa vykoná od $i = 0$ po *m*. V prvej *if* podmienke sa overí, či je prvok a_{ii} matice *A* rovný 0. Ak áno, tak sa zavolá sa metóda, ktorá pomocou cyklu skontroluje, či je v stĺpci pod prvkom a_{ii} číslo 1 alebo -1. Ak je táto podmienka splnená, tak sa zavolá ďalšia metóda, ktorá vráti index riadku, v ktorom je číslo 1 alebo -1 a uloží ho do premennej *k*. Následne sa riadky *i* a *k* vymenia

navzájom pomocou nasledujúcej metódy:

```
public void vymen(int i, int k, int n, Double[][] A) {
    for (int j = 0; j < n; j++) {
        double t;
        t = A[i][j];
        A[i][j] = A[k][j];
        A[k][j] = t;

    }
}
```

Ak v stĺpci pod prvkom a_{ii} nie je číslo 1 alebo -1, pomocou *for* cyklu sa prehľadá stĺpec pod týmto prvkom a ak nájde nenulový prvok, vymení riadok *i* s riadkom *k*, ktorý obsahuje nenulový prvok. Nasleduje cyklus *while*, ktorý je vytvorený pre prípad, ak je celý stĺpec nulový. Pokiaľ sa prvok a_{il} (premenná l je nastavená zo začiatku na hodnotu i) bude rovnať 0 a zároveň $l < n - 2$, zvýši sa hodnota premennej l o 1, čo znamená, že sa posunieme na prvok o jeden stĺpec napravo. Znova sa vykoná ten istý algoritmus prehľadávania stĺpca na nájdenie nenulového prvku. Tento postup sa bude vykonávať, pokiaľ bude platiť podmienka cyklu.

Ak prvá *if* podmienka nebola splnená, t.j. prvok $a_{ii} \neq 0$, v druhej *if* podmienke sa overí, či tento prvok je 1 alebo -1. Ak nie, znova sa zisťuje, či je v stĺpci pod týmto prvkom číslo 1 alebo -1 a ak áno, riadky sa vymenia.

Ďalším krokom je eliminácia, ktorá sa realizuje pomocou *for* cyklu od $k = i + 1$, keďže je potrebné eliminovať stĺpec pod vedúcim prvkom, až po *m*. Do premennej *podiel* sa uloží záporný podiel prvku a_{kl} (prvok pod vedúcim prvkom, ktorý chceme vynulovať) a vedúceho prvku a_{il}. Ak $podiel = 1$, bude riadok i iba pripočítaný k riadku k, keďže nie je v tomto prípade potrebné násobenie. Ak $podiel \neq 0$, bude riadok i vynásobený podielom a pripočítaný k riadku k. Po eliminácii sa overuje posledná podmienka $l = n - 2$. Ak je táto podmienka splnená, znamená to, že index stĺpca l sa rovná indexu predposledného stĺpca matice a už nie je možné vykonávať ďalšie iterácie, keďže posledný stĺpec predstavuje pravú stranu systému rovníc. V tomto prípade sa ukončí vykonávanie hlavného cyklu pomocou kľúčového slova *break*. Pri riešení hodnosti matice použijeme rovnaký algoritmus na získanie matice v stupňovitom tvare, ale posledná podmienka bude $l = n - 1$, keďže pri maticiach vykonávame úpravy na všetkých stĺpcoch matice. Pre zobrazenie iba jedného nasledujúceho kroku sa použije po každej ekvivalentnej úprave kľúčové slovo *return*, ktoré ukončí vykonávanie metódy.

Po každej vykonanej úprave sa overí, či je matica v stupňovitom tvare. Toto

overenie je vykonané pomocou nasledujúcej metódy.

```
public boolean checkVysledok(int m, int n, Double[][] A) {
    int count = 0;
    ArrayList<Integer> nuloveRiadky = new ArrayList<>();
    ArrayList<Integer> nenuloveRiadky = new ArrayList<>();
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (A[i][j] == 0.0) {
                count++;
            }
        }
        if (count == n) {
            nuloveRiadky.add(i);
        } else {
            nenuloveRiadky.add(i);
        }
        count = 0;
    }

    boolean riadky = false;
    if (!nuloveRiadky.isEmpty()) {
        for (int i = 0; i < nenuloveRiadky.size(); i++) {
            for (int j = 0; j < nuloveRiadky.size(); j++) {
                if (nuloveRiadky.get(j) > nenuloveRiadky.get(i)) {
                    riadky = true;
                } else {
                    riadky = false;
                }
            }
        }
    } else {
        riadky = true;
    }

    ArrayList<Integer> list = new ArrayList<>();
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n-1; j++) {
            if (A[i][j] != 0.0) {
                list.add(j);
                break;
            }
        }
    }
    boolean zoradene = Ordering.natural().isStrictlyOrdered(list);
    if (riadky && zoradene) {
        return true;
    }
    return false;
}
```

V prvom *for* cykle sa počítajú nulové prvky pomocou premennej *count*. Po každom riadku sa overí, či počet nulových prvkov v riadku je rovný počtu stĺpcov. Ak áno, znamená to, že celý riadok je nulový a do ArrayListu *nuloveRiadky* sa pridá index tohto riadku. Ak nie, tak sa tento index pridá do ArrayListu *nenuloveRiadky*. Tento cyklus sa bude opakovať, až pokiaľ neprejde všetky riadky. Ak existujú nulové riadky, t.j. list *nuloveRiadky* nie je prázdny, v ďalšom *for* cykle sa overí, či všetky prvky v liste *nuloveRiadky* sú väčšie ako v *nenuloveRiadky*. Ak áno, znamená to, že všetky nulové

riadky sa nachádzajú pod nenulovými. Premenná *riadky* sa nastaví na hodnotu *true*. Ak nie, premenná *riadky* bude *false*. Ak matica nemá žiadne nulové riadky, potom *riadky* = *true*. V poslednom *for* cykle sa hľadá prvý nenulový prvok v každom riadku. Ak prvok $a_{ij} \neq 0$, uloží sa index stĺpca, v ktorom sa tento prvok nachádza, do ArrayListu *list* a vnorený cyklus sa ukončí. Po nájdení všetkých vedúcich prvkov sa pomocou metódy *isStrictlyOrdered* overí, či sú všetky prvky v ArrayListe *list* zoradené od najmenšieho po najväčší. Táto metóda je súčasťou externej knižnice *Guava*. Vracia hodnotu *true* alebo *false*, ktorá sa uloží do premennej *zoradene*. Nakoniec sa overí, či premenné *riadky* a *zoradene* majú hodnotu *true*. Ak áno, znamená to, že sú splnené podmienky pre stupňovitý tvar matice a táto metóda vráti hodnotu *true*. Ak aspoň jedna z podmienok nie je splnená, metóda vráti *false*.

Ak je matica v stupňovitom tvare, zavolá sa metóda, ktorá vypíše výsledok systému rovníc alebo hodnosť matice v závislosti od typu úlohy. Pred výpisom výsledku sa najskôr ale zisťuje, či je systém riešiteľný pomocou nasledujúcej metódy.

```
public boolean checkRiesitelnost(int m, int n, Double[][] A) {
    int count = 0;
    int pocetNenulovychA = m;
    int pocetNenulovychAR = m;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n - 1; j++) {
            if (A[i][j] == 0.0) {
                count++;
            }
        }
        if (count == n - 1) {
            pocetNenulovychA--;
        }
        count = 0;
    }
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (A[i][j] == 0.0) {
                count++;
            }
        }
        if (count == n) {
            pocetNenulovychAR--;
        }
        count = 0;
    }
    if (pocetNenulovychA = pocetNenulovychAR) {
        return true;
    }
    return false;
}
```

Premenná *pocetNenulovychA* predstavuje počet nenulových riadkov matice systému **A** a premenná *pocetNenulovychAR* predstavuje počet nenulových prvkov rozšírenej matice **Ā**. Tieto premenné sú zo začiatku nastavené na hodnotu *m*. Metóda má dva hlavné *for* cykly. V prvom sa počítajú nulové prvky každého riadku matice systému **A** pomocou premennej *count*. Po každom riadku sa overí, či počet nulových prvkov v tomto riadku je rovný *n-1*, čo znamená, že tento riadok (bez posledného stĺpca) je nulový. Ak áno, hodnota premennej *pocetNenulovychA* sa zníži o 1. V druhom cykle sa počítajú nulové prvky každého riadku rozšírenej matice **Ā.** Po každom riadku sa overí, či počet nulových prvkov v tomto riadku je rovný *n.* Ak áno, hodnota premennej *pocetNenulovychAR* sa zníži o 1. Nakoniec sa overí podmienka *pocetNenulovychA = pocetNenulovychAR.* Ak je to pravda, metóda vráti hodnotu *true*, v opačnom prípade *false*. Ak počet nenulových riadkov matice **A** sa rovná počtu nenulových riadkov matice **Ā,** systém je riešiteľný. Ak systém nie je riešiteľný, vypíše sa táto informácia spolu so zdôvodnením do panelu s výsledkom.

Ak je systém riešiteľný, ďalej je potrebné zistiť počet riešení pomocou nasledujúcej metódy.

```
public boolean checkNekonecno(int m, int n, Double[][] matica) {
    int count = 0;
    int pocetNenulovych = m;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (A[i][j] == 0.0) {
                count++;
            }
        }
        if (count == n) {
            pocetNenulovych--;
        }
        count = 0;
    }
    if (pocetNenulovych < n - 1) {
        return true;
    }
    return false;
}
```

Táto metóda obsahuje jeden hlavný *for* cyklus, ktorý využíva rovnaký algoritmus na počítanie nulových prvkov každého riadku rozšírenej matice **Ā** ako pri predchádzajúcej metóde. Na konci sa overí, či počet nenulových riadkov je menší ako *n-1*, t.j. ako počet neznámych. Ak áno, metóda vráti hodnotu *true*, teda systém má nekonečne veľa riešení. To bude vypísané aj do panelu s výsledkom spolu s hodnotami neznámych. V opačnom prípade vráti *false*, čo znamená, že systém má jedno riešenie. V tomto prípade sa zavolá metóda pre dosadzovanie do neznámych, ktorá vypočíta výsledné riešenie.

4.1.2 Gauss-Jordanova eliminačná metóda v Jave

Na implementáciu algoritmu Gauss-Jordanovej eliminačnej metódy sme použili nasledujúcu metódu:

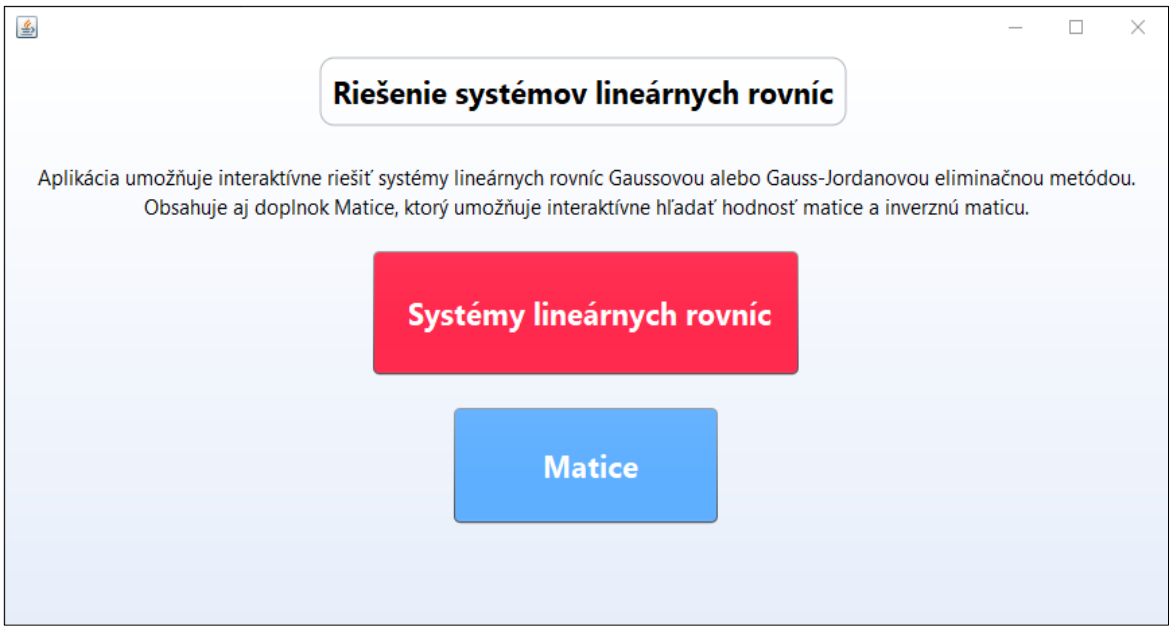
```
public void gaussJordan(int m, int n, Double[][] A){

 int l = 0;
        for(int i = 0; i < m; i++){
                l = i;
                //ak veduci prvok je nula, tak vymen riadky
                if (A[i][i] == 0) {
                for (int k = i + 1; k < m; k++) {
                        if (A[k][i] != 0) {
                                vymen(i, k, n, A);
                                break;
                        }
                }

                        while(A[i][l] == 0 && l < n-2) {
                                l++;
                                for (int k = i + 1; k < m; k++){
                                        if (A[i][l] == 0 && A[k][l] !=0){
                                                vymen(i, k, n, A);
                                        }
                                }
                        }
                }
                        double lead = A[i][l];
                        //vydelit kazde nenulove cislo v riadku veducim prvkom
                        if(lead != 1 && lead !=0) {
                                for (int j = 0; j < n; j++) {
                                        A[i][j] = A[i][j] / lead;
                                }

                        }
                        //eliminacia
                        for(k = 0; k < m; k++){
                                if (k!= i){
                                        double h = -(A[k][l]);
                                            if(h ==1) {

                                                    for (int j = 0; j < n; j++) {
                                                A[k][j] = A[k][j] + A[i][j];
                                                    }
                                            }

                                           else if(h !=0){
                                                 for(int j = 0; j < n; j++){
                                                A[k][j] = A[k][j] + h*A[i][j];
                                                 }
                                           }
                                }
                        }
                }
            if(l == n-2){
                break;
            }
    }
}
```

Metóda obsahuje hlavný cyklus *for*, ktorý sa vykoná od $i = 0$ po m. Zo začiatku je postup rovnaký ako pri Gaussovej eliminačnej metóde, rozdiel je v tom, že nie je potrebné

hľadať prvky s hodnotou 1 alebo -1, ale na výmenu stačí akýkoľvek nenulový prvok. Ak už je vedúci prvok identifikovaný, uloží sa jeho hodnota do premennej *lead*. Následne sa každý prvok v riadku *i*, ktorý nie je rovný 0 alebo 1, vydelí vedúcim prvkom. Posledným krokom je eliminácia, ktorá prebieha v cykle od $k = 0$, keďže musí prejsť všetky riadky, až po *m*. Podmienkou je, že $k \neq i$, pretože nemôžeme eliminovať vedúci prvok. Ak je táto podmienka splnená, do premennej *h* sa uloží záporná hodnota prvku a_{kl}, teda prvku, ktorý je potrebné eliminovať. Ak $h = 1$, bude riadok *i* iba pripočítaný k riadku *k* a ak $h \neq 0$, bude riadok *i* vynásobený číslom *h* a pripočítaný k riadku *k*. Po eliminácii sa overuje posledná podmienka $l = n - 2$ rovnako ako pri Gaussovej eliminačnej metóde. Ak je táto podmienka splnená, ukončí sa vykonávanie hlavného cyklu pomocou kľúčového slova *break*.

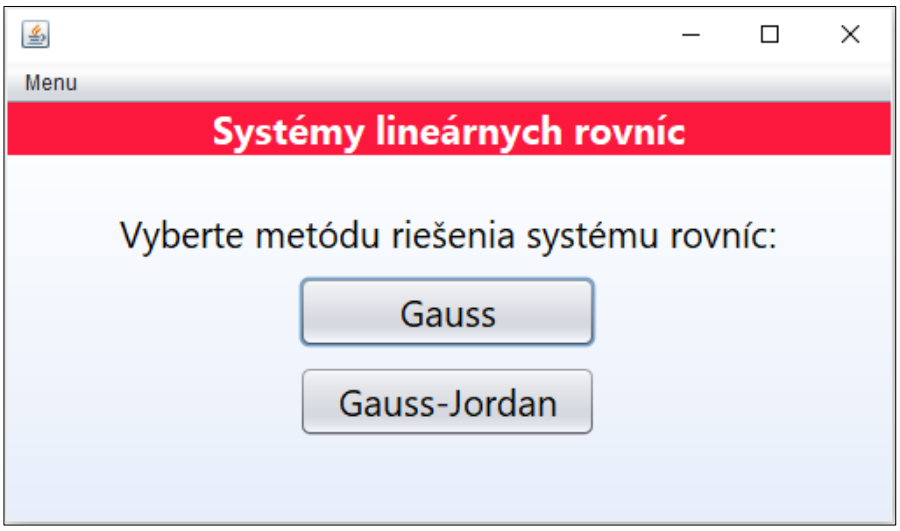
4.2 Ukážka použitia aplikácie

V tejto podkapitole si ukážeme riešenie každého typu úlohy na konkrétnych príkladoch v aplikácii. Na úvodnej obrazovke, ktorá je na obrázku 17, je možné zvoliť si jednu z kategórii úloh pomocou dvoch tlačidiel. Tieto tlačidlá sú rozlíšené farbou aj veľkosťou, keďže matice sú iba doplnkom v aplikácii.

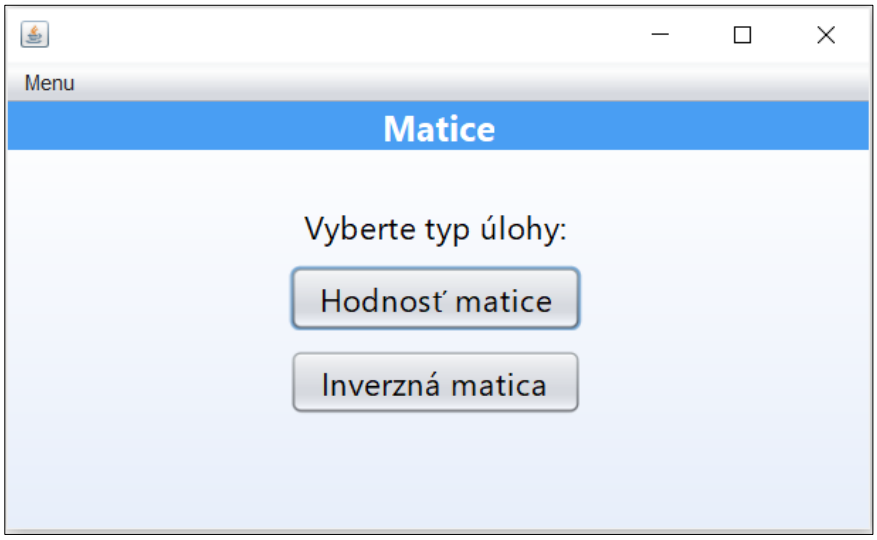


Obrázok 17 - *Úvodné okno*

Po kliknutí na jedno z tlačidiel sa zobrazí nové okno, ktoré je na obrázkoch 18 a 19. Toto okno obsahuje názov kategórie úlohy vo farbe príslušného tlačidla z úvodného okna. V prípade výberu systémov lineárnych rovníc je na výber metóda riešenia – Gaussova alebo Gauss-Jordanova metóda. Pri zvolení kategórie matíc je možné vybrať typ úlohy – hodnosť matice alebo inverznú maticu.



Obrázok 18 - *Systémy lineárnych rovníc*



Obrázok 19 – *Matice*

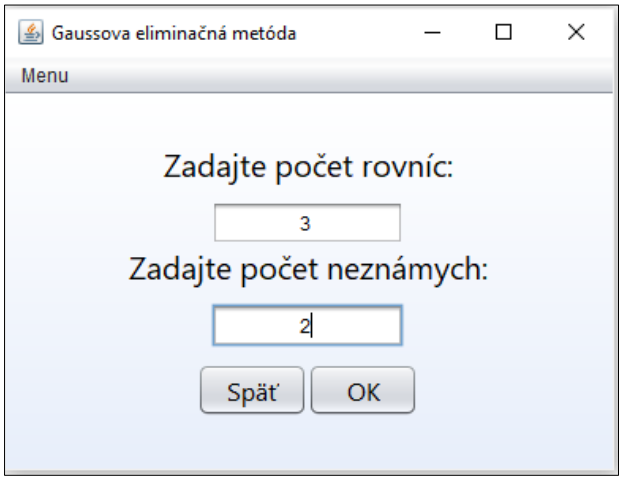
Riešenie týchto úloh v aplikácii ukážeme na nasledujúcich príkladoch. Na prvom príklade budeme demonštrovať riešenie systému lineárnych rovníc Gaussovou eliminačnou metódou.

Príklad 1: Riešme Gaussovou eliminačnou metódou systém lineárnych rovníc

$$\begin{aligned} -5x_2 &= -4 \\ x_1 + 4x_2 &= 3 \\ 5x_1 + 10x_2 &= 7 \end{aligned}$$

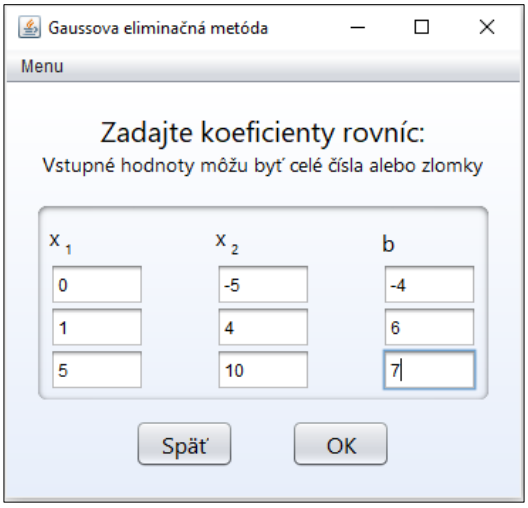
Riešenie:

Po zvolení Gaussovej eliminačnej metódy zadáme v novom okne počet rovníc a neznámych a potvrdíme kliknutím na tlačidlo *OK*.



Obrázok 20 - *Gaussova metóda zadávanie rozmerov*

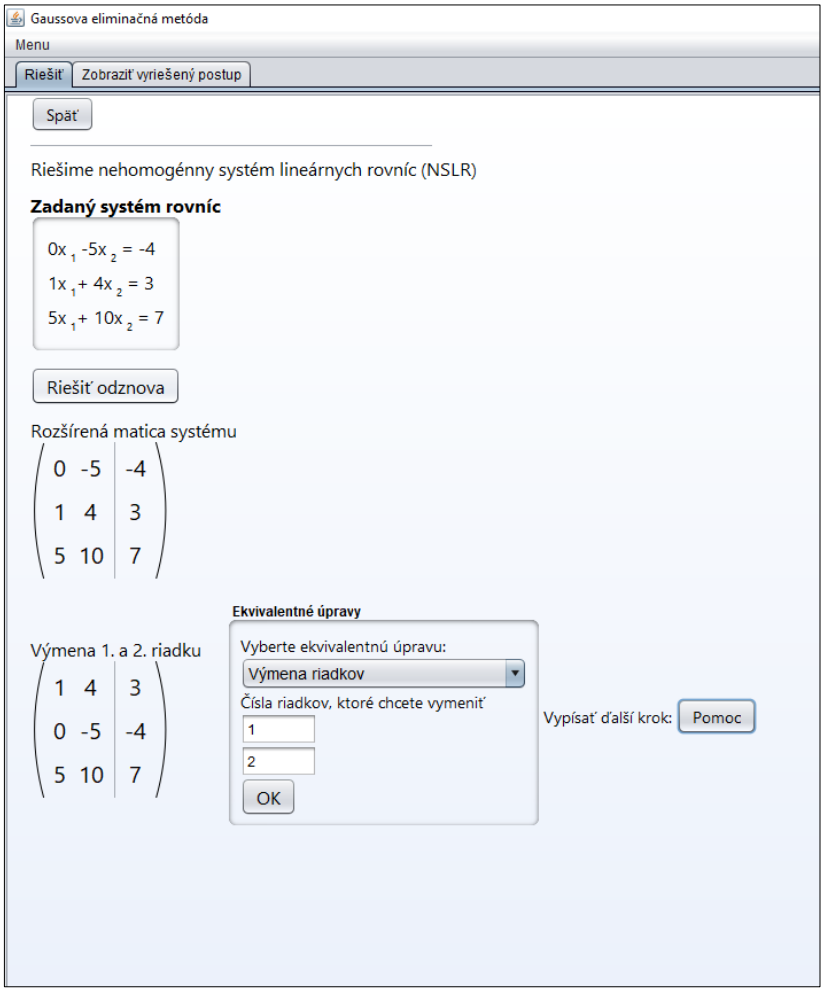
V ďalšom okne zadávame koeficienty rovníc systému a po zadaní všetkých hodnôt znova potvrdíme tlačidlom *OK*.



Obrázok 21 - *Gaussova metóda zadávanie koeficientov*

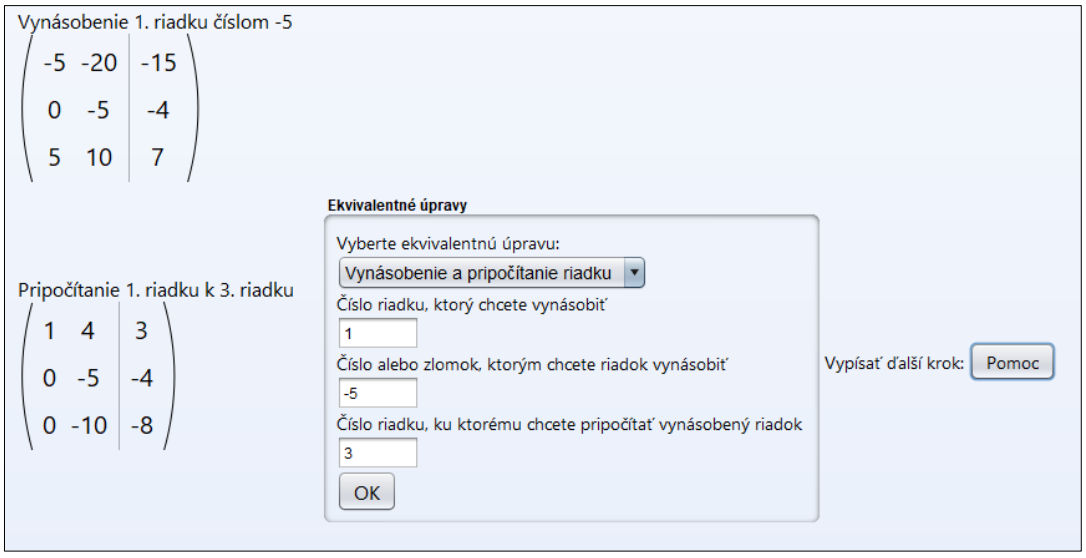
V poslednom okne bude prebiehať samotné riešenie zadaného príkladu. Po kliknutí na tlačidlo *Riešiť* sa zobrazí rozšírená matica systému a panel s ekvivalentnými úpravami.

Ako prvú úpravu zvolíme výmenu riadkov a zadáme čísla riadkov 1 a 2.



Obrázok 22 - *Gaussova metóda - riešenie krok 1*

Ďalšou úpravou bude vynásobenie a prípočítanie riadku. Chceme vynásobiť 1. riadok číslom -5 a pripočítať k 3. riadku.

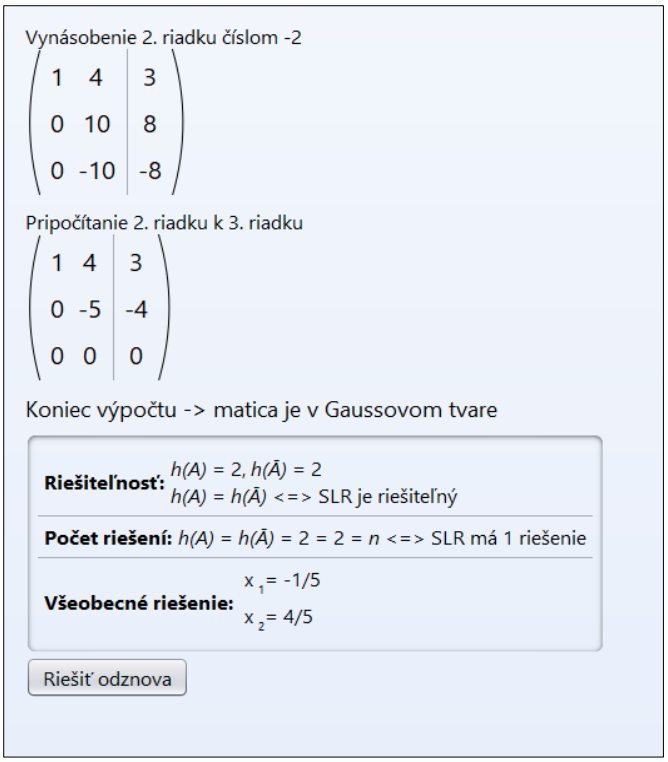


Obrázok 23 - *Gaussova metóda - riešenie krok 2*

Posledná úprava je vynásobenie 2. riadku číslom -2 a pripočítanie k 3. riadku. Po vykonaní tejto úpravy bude matica v Gaussovom tvare a zobrazí sa panel s výsledkom. Zadaný systém má 1 riešenie:

$$x_1 = -\frac{1}{5}$$

$$x_2 = \frac{4}{5}$$



Obrázok 24 - *Gaussova metóda - výsledok*

Na druhom príklade ukážeme riešenie systému lineárnych rovníc Gauss-Jordanovou eliminačnou metódou.

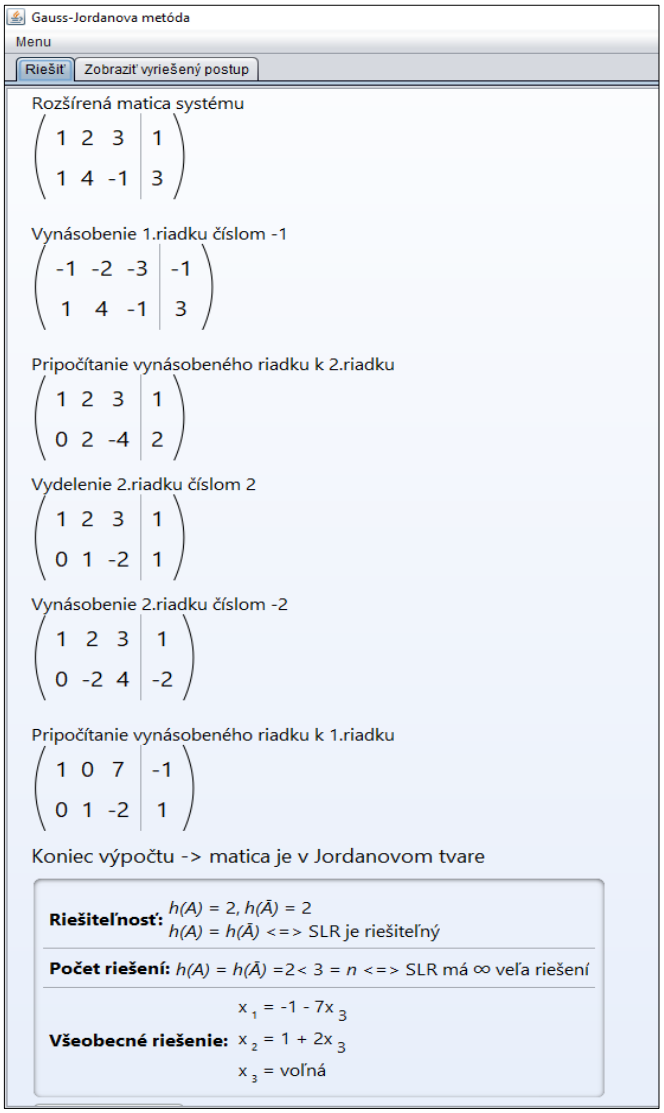
Príklad 2: Riešme Gauss-Jordanovou eliminačnou metódou systém lineárnych rovníc

$$x_1 + 2x_2 + 3x_3 = 1$$
$$x_1 + 4x_2 - x_3 = 3$$

Riešenie:

Počet rovníc, neznámych a koeficienty systému zadávame rovnakým spôsobom ako pri Gaussovej eliminačnej metóde. V prvom kroku riešenia vynásobíme 1. riadok číslom 1 a pripočítame k 2. riadku. Následne vydelíme 2. riadok číslom 2 a v poslednom kroku vynásobíme 2. riadok číslom -2 a pripočítame k 1. riadku. Matica je teraz v Jordanovom tvare. Zadaný systém má nekonečne veľa riešení.

$$x_1 = -1 - 7x_3$$
$$x_2 = 1 + 2x_3$$
$$x_3 = voľná$$



Obrázok 25 - *Gauss-Jordanova metóda – výsledok*

Riešenie hodnosti matice budeme demonštrovať na príklade 3.

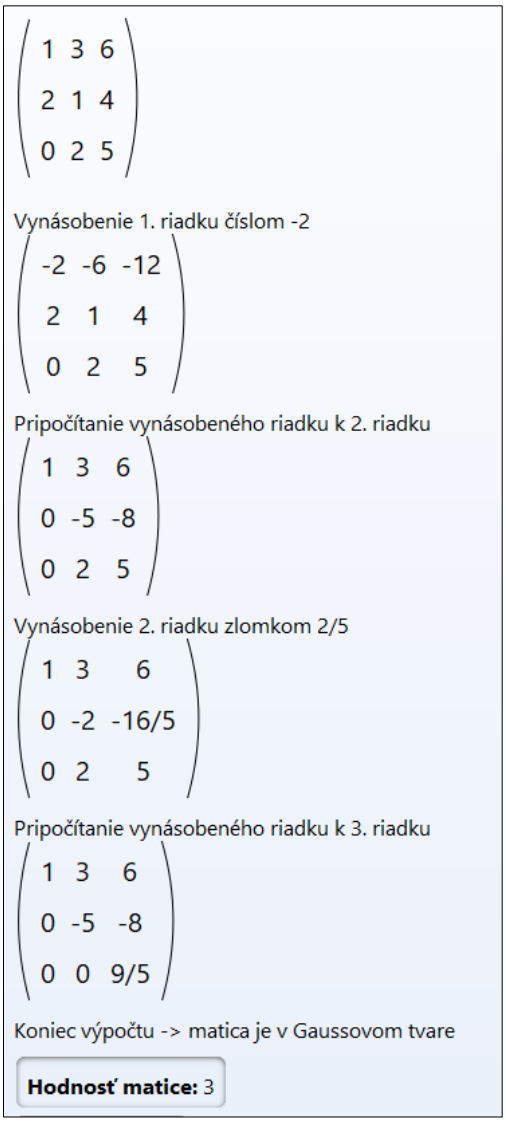
Príklad 3: Určme hodnosť matice **A**

$$\mathbf{A} = \begin{pmatrix} 1 & 3 & 6 \\ 2 & 1 & 4 \\ 0 & 2 & 5 \end{pmatrix}$$

Riešenie:

Pri riešení hodnosti matice je potrebné najskôr zadať počet riadkov a stĺpcov matice a prvky matice. Následne môžeme riešiť úlohu výberom ekvivalentných úprav rovnako ako pri systémoch lineárnych rovníc. Pri riešení hodnosti matice využijeme Gaussovu

eliminačnú metódu. V prvom kroku vynásobíme 1. riadok číslom -2 a pripočítame k 2. riadku. V druhom kroku vynásobíme 2. riadok zlomkom $\frac{2}{5}$, pripočítame k 3. riadku a dostaneme tak maticu v Gaussovom tvare. Z tohto tvaru môžeme určiť, že hodnosť matice je 3.



Obrázok 26 - *Hodnosť matice – výsledok*

Na príklade 4 si ukážeme poslednú úlohu, ktorou je nájdenie inverznej matice.

Príklad 4: Určme inverznú maticu k matici **A**

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 0 & 4 \end{pmatrix}$$

Riešenie:

Pri riešení inverznej matice zadávame iba jeden rozmer, keďže matica musí byť štvorcová. Po zadaní prvkov sa k matici **A** vytvorí jednotková matica. V prvom kroku riešenia vydelíme 2. riadok číslom 4. V ďalšom kroku vynásobíme 2. riadok číslom -3

a pripočítame k 1. riadku. Matica je teraz v Jordanovom tvare a určili sme tak inverznú maticu $\mathbf{A^{-1}}$.

$$\mathbf{A^{-1}} = \begin{pmatrix} 1 & -\frac{3}{4} \\ 0 & \frac{1}{4} \end{pmatrix}$$

Obrázok 27 - *Inverzná matica – výsledok*

4.3 Porovnanie vytvorenej aplikácie s existujúcimi aplikáciami

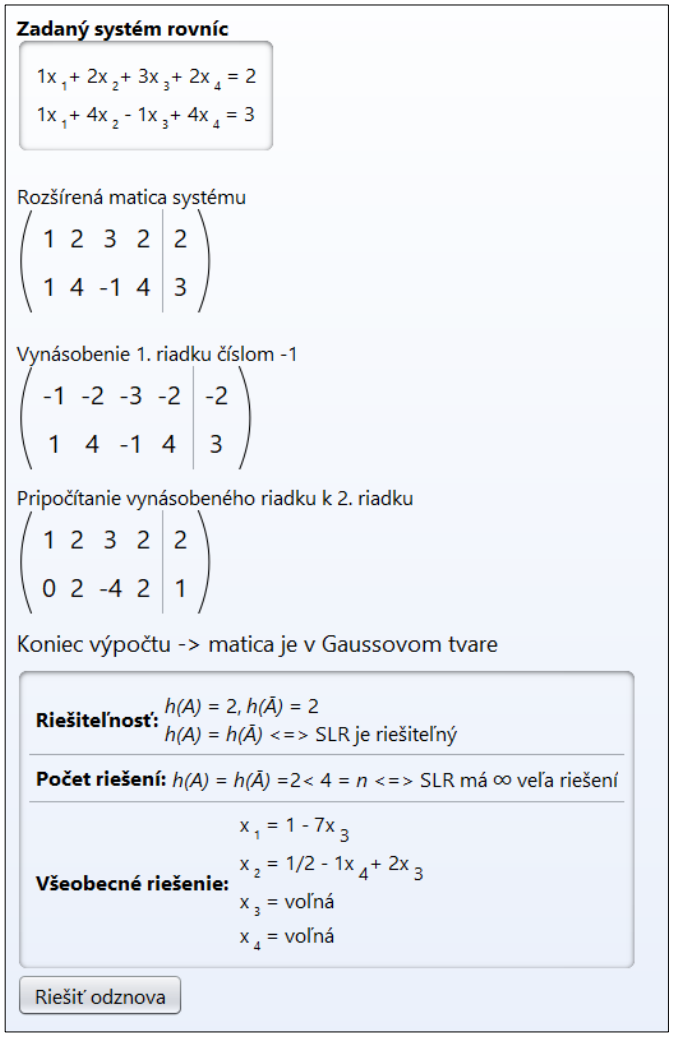
V kapitole 1.4 sme analyzovali tri existujúce webové aplikácie zamerané na riešenie systémov lineárnych rovníc. Všetky tri aplikácie dokážu riešiť systémy *m* lineárnych rovníc s *n* neznámymi. Okrem výsledku zadaného príkladu umožňujú zobraziť aj podrobný postup riešenia. Hlavnou výhodou našej vytvorenej aplikácie v porovnaní s ostatnými je možnosť zadávať ekvivalentné úpravy matice a vyriešiť tak zadaný príklad.

Aplikácie *Matrix reshish* a *Linear Algebra Toolkit* neumožňujú riešiť systémy rovníc Gaussovou, ale iba Gauss-Jordanovou eliminačnou metódou. Aplikácia *Matrix calc* umožňuje riešiť systémy pomocou oboch metód. Na nasledujúcom príklade porovnáme riešenie systému lineárnych rovníc Gaussovou eliminačnou metódou v aplikácii *Matrix calc* s riešením v našej aplikácii.

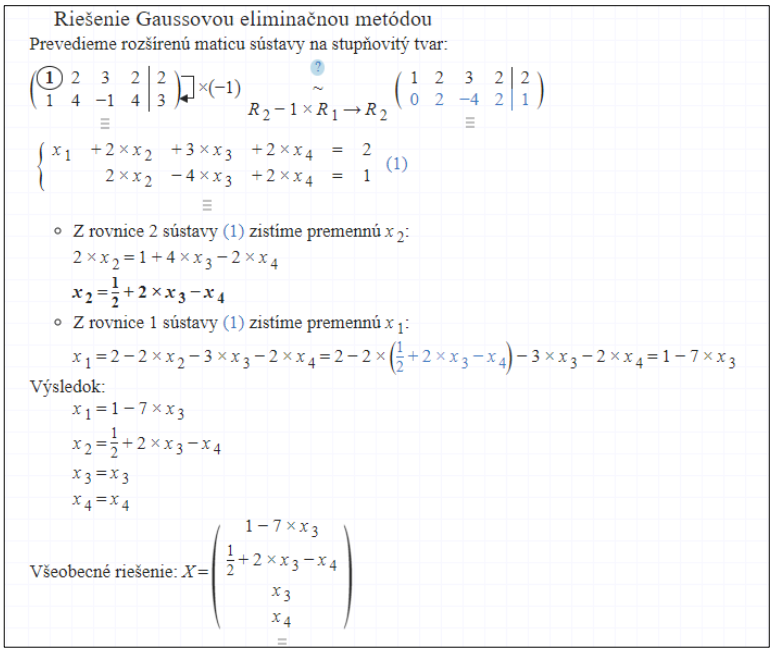
Príklad 5: Riešme Gaussovou eliminačnou metódou systém lineárnych rovníc

$$x_1 + 2x_2 + 3x_3 + 2x_4 = 2$$
$$x_1 + 4x_2 - x_3 + 4x_4 = 3$$

Riešenie:

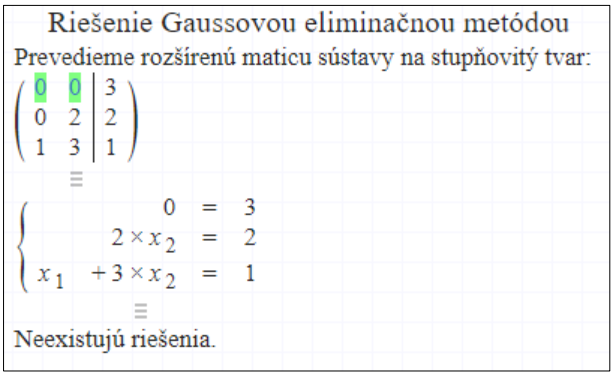


Obrázok 28 – *Príklad 5 v našej aplikácii*

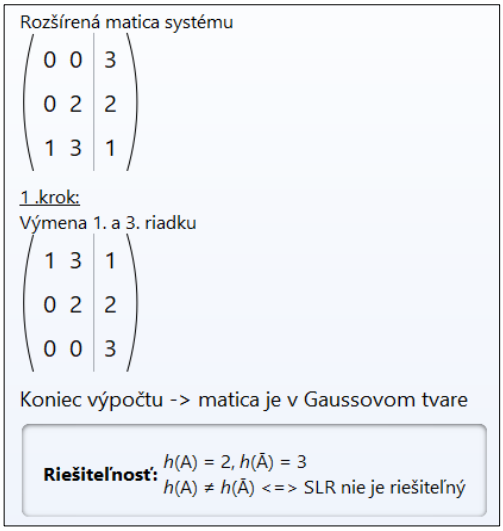


Obrázok 29 – *Príklad 5 v aplikácii Matrix calc*

V našej aplikácii sme príklad riešili postupným zadávaním ekvivalentných úprav a po získaní matice v Gaussovom tvare sa zobrazil výsledok. V aplikácii *Matrix calc* sa po zadaní vstupov vypíše riešenie spolu s výsledkom. Jednotlivé kroky riešenia sú popísané dostatočne v oboch aplikáciách, rozdielom je, že v našej aplikácii uvádzame aj maticu po vynásobení riadku. Výhodou aplikácie *Matrix calc* je zobrazenie postupu získania hodnôt neznámych z Gaussovho tvaru matice. Výhodou našej aplikácie je zdôvodnenie riešiteľnosti a počtu riešení systému rovníc. Nedostatkom aplikácie *Matrix calc* je, že pri niektorých systémoch rovníc nie je rozšírená matica úplne prevedená na Gaussov tvar, napríklad ak už vieme určiť riešenie. Takýto príklad je na obrázkoch 30 a 31.



Obrázok 30 – *Gaussov tvar Matrix calc*



Obrázok 31 – *Gaussov tvar naša aplikácia*

Záver

Cieľom záverečnej práce bolo vytvoriť aplikáciu v jazyku Java, ktorá rieši systémy *m* lineárnych rovníc s *n* neznámymi využitím Gaussovej eliminačnej metódy. Riešenie je realizované postupným zadávaním ekvivalentných úprav.

V prvej kapitole sme definovali niektoré základné pojmy z teórie lineárnej algebry. Taktiež sme spravili analýzu a porovnanie aplikácii zameraných na riešenie systémov lineárnych rovníc. Na porovnanie sme si vybrali tri webové aplikácie, ktoré dokážu riešiť systémy *m* lineárnych rovníc s *n* neznámymi. Z týchto troch aplikácii iba jedna umožňuje riešenie systémov pomocou oboch eliminačných metód. Všetky tri aplikácie dokážu zobraziť aj celý postup riešenia spolu s výsledkom. V závere prvej kapitoly sme sa venovali charakteristike programovacieho jazyka Java a technológii využitých pri tvorbe aplikácie.

V druhej kapitole sme stanovili hlavný cieľ práce a čiastkové ciele potrebné na splnenie tohto cieľa. Prvý čiastkový cieľ, ktorým bola analýza a porovnanie aplikácii zameraných na riešenie systémov lineárnych rovníc, sme splnili v prvej kapitole. Na vývoj aplikácie sme si vybrali vývojové prostredie IntelliJ IDEA a na tvorbu používateľského rozhrania sme použili balík nástrojov Swing. Následne sme implementovali algoritmus Gaussovej a Gauss-Jordanovej eliminačnej metódy v aplikácii pomocou jazyka Java a vytvorili sme používateľské rozhranie aplikácie. Nakoniec sme porovnali našu vytvorenú aplikáciu s webovou aplikáciou, ktorú sme analyzovali v prvej kapitole. Splnením všetkých čiastkových cieľov sme tak naplnili hlavný cieľ práce.

V tretej kapitole sme na úvod vysvetlili algoritmus Gaussovej a Gauss-Jordanovej eliminačnej metódy. Následne sme definovali požiadavky na aplikáciu a vytvorili sme návrh používateľského rozhrania.

Štvrtá kapitola je venovaná výsledkom práce. V úvode tejto kapitoly sme opísali funkcionalitu aplikácie a niektoré použité metódy. V ďalšej časti je ukážka riešenia príkladov v aplikácii. V závere kapitoly sme porovnali vytvorenú aplikáciu s vybranou existujúcou aplikáciou.

Výsledkom práce je aplikácia, ktorá umožňuje interaktívne riešiť systémy *m* lineárnych rovníc o *n* neznámych Gaussovou alebo Gauss-Jordanovou eliminačnou

metódou. Taktiež umožňuje riešiť hodnosť matice a inverznú maticu pomocou týchto metód. Aplikácia je určená najmä študentom lineárnej algebry, ktorí ju môžu využiť na riešenie systémov lineárnych rovníc a prípadne na porovnanie svojho postupu s vyriešeným postupom v aplikácii. Hlavným prínosom aplikácie a taktiež rozdielom v porovnaní s inými podobnými aplikáciami je možnosť zadávania ekvivalentných úprav matice pri riešení.

Zoznam použitej literatúry

[1] SAKÁLOVÁ, Katarína – SIMONKA, Zsolt – STREŠŇÁKOVÁ, Anna. *Matematika. Lineárna algebra*. Bratislava : Ekonóm, 2010. 264 s. ISBN 978-80-225-2906-8

[2] KUTTLER, Ken. *A First Course in Linear Algebra*. Calgary : Lyryx, 2017. 608 s. ISBN 978-1542895521

[3] ZLATOŠ, Pavol. *Lineárna algebra a geometria.* Bratislava : Marenčin PT, 2011. 744 s. ISBN 978-80-8114-111-9

[4] FORD, William - *Numerical Linear Algebra with Applications.* Cambridge : Academic Press, 2014. 628 s. ISBN 9780123947840

[5] MATRIXCALC. *Riešenie sústav lineárnych rovníc* [online]. [cit. 2020-03-09]. Dostupné na: https://matrixcalc.org/en/slu.html

[6] MATRIX RESHISH. *Gauss-Jordan Elimination Calculator* [online]. [cit. 2020-03-10]. Dostupné na: https://matrix.reshish.com/gauss-jordanElimination.php

[7] LINEAR ALGEBRA TOOLKIT. *Solving a system of linear equations* [online]. [cit. 2020-03-11]. Dostupné na:

[8] FLANAGAN, David – EVANS, Ben. *Java in a Nutshell.* 7th Edition. Sebastopol : O'Reilly Media, Inc., 2018. 458 s. ISBN 9781492037255.

[9] SCHILDT, Herbert. *Java: The Complete Reference.* Eleventh Edition. New York : McGraw-Hill, 2018. 1248 s. ISBN 9781260440249.

[10] LEUCK, Daniel. – NIEMEYER, Patrick. – LOY, Marc. *Learning Java*. 5th Edition. Sebastopol : O'Reilly Media, Inc., 2020. 505 s. ISBN 9781492056270.

[11] OBARE, Miles et al. *Java Fundamentals*. Birmingham : Packt Publishing, 2019. 408 s. ISBN 9781789801736

[12] HORSTMANN, Cay S. *Core Java Volume I—Fundamentals.* 11th Edition. New Jersey : Prentice Hall, 2018. 928 s. ISBN 978-0135166307

[13] APACHE COMMONS. *Commons Math: The Apache Commons Mathematics Library* [online]. [cit. 2020-03-20]. Dostupné na: http://commons.apache.org/proper/commons-math/

[14] ROUSE, M. *What is integrated development environment(IDE)*? [online]. 2018 [cit. 2020-03-21]. Dostupné na: https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

[15] JETBRAINS. *Ultimate vs Community – Compare Editions* [online]. [cit. 2020-03-20]. Dostupné na: https://www.jetbrains.com/idea/features/editions_comparison_matrix.html

[16] ASSUMPCAO, Hudson O. *Getting Started with IntelliJ IDEA*. Birmingham : Packt Publishing, 2013. 114 s. ISBN 9781849699617

[17] KROCHMALSKI, Jaroslaw. *IntelliJ IDEA Essentials*. Birmingham : Packt Publishing, 2014, 276 s. ISBN 9781784396930

[18] FISCHER, Paul. *An Introduction to Graphical User Interfaces with Java Swing*. Dorchester : Addison-Wesley, 2005. 306 s. ISBN 0321 22070 6

[19] WOOD, Dave et al. *Java Swing*. 2nd Edition. Sebastopol : O'Reilly Media, Inc., 2002. 1280 s. ISBN 9780596004088

[20] ALTEXSOFT. *The Good and the Bad of Java Programming* [online]. [cit. 2020-04-13]. Dostupné na: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/

[21] SOMMERVILLE, Ian. *Software Engineering*. 10th Edition. London : Pearson Education, 2016, 811 s. ISBN 978-1-292-09613-1

Prílohy

Príloha č. 1: Spustiteľná aplikácia
Príloha č. 2: Zdrojový kód aplikácie