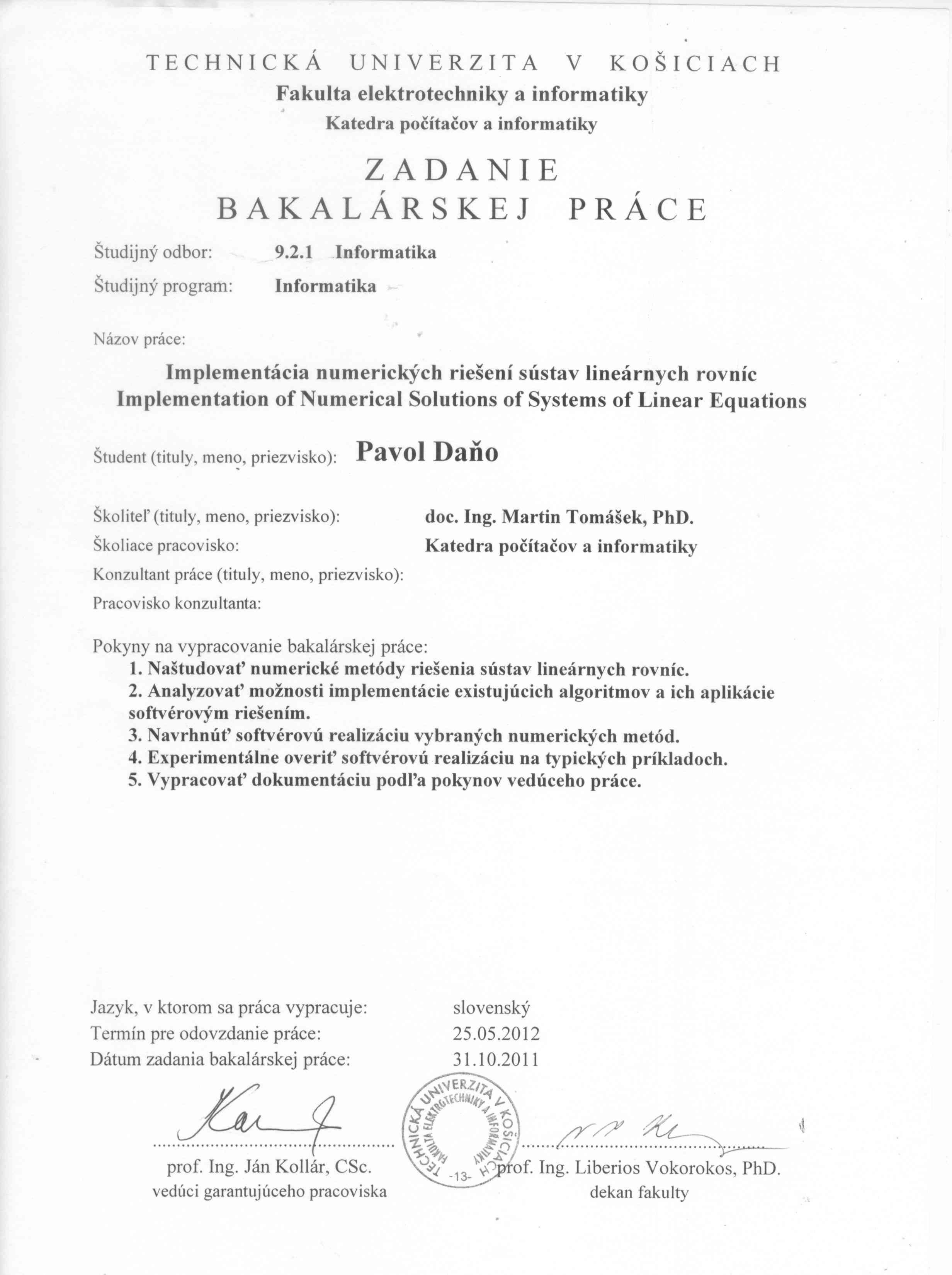
|  |  |
| --- | --- |
| **TECHNICKÁ UNIVERZITA V KOŠICIACH**  **FAKULTA ELEKTROTECHNIKY A INFORMATIKY** | |
| Implementácia numerických riešení sústav lineárnych rovníc  **Bakalárska práca** | |
| **2012** | **Pavol DAŇO** |

|  |  |  |
| --- | --- | --- |
| **TECHNICKÁ UNIVERZITA V KOŠICIACH**  **FAKULTA ELEKTROTECHNIKY A INFORMATIKY** | | |
| Implementácia numerických riešení sústav lineárnych rovníc  Bakalárska práca | | |
|  | | |
| Študijný program: | | Informatika |
| Študijný odbor: | | 9.2.1 Informatika |
| Školiace pracovisko: | | Katedra počítačov a informatiky (KPI) |
| Školiteľ: | | doc. Ing. Martin Tomášek, PhD. |
|  | |  |
| **2012 Košice** | **Pavol DAŇO** | |

|  |
| --- |
| **Abstrakt v SJ**  Primárnym cieľom tejto bakalárskej práce je implementácia numerických riešení systémov lineárnych rovníc. So zámerom splniť stanovené požiadavky autor vychádza z teoretickej roviny matematickej algebry. Oboznamuje sa so všeobecnými aj špecifickými metódami a vďaka nadobudnutým poznatkom analyzuje možnosti implementácie. Tým plynule prechádza na návrh jednotlivých algoritmov a ich konečnej softvérovej realizácie. Vo finálnej fáze spracovania problematiky popisuje primárne funkcie vytvoreného programu, jeho možnosti a obmedzenia. Záver jadra práce predstavujú ukážkové príklady použitia programu.  **Kľúčové slová**  Matica, lineárny, rovnica, Gauss, Jacobi, Gauss-Seidel, iterácia, relaxácia, Cholesky, metóda |

|  |
| --- |
| **Abstrakt v AJ**  The primary aim of the Bachelor´s thesis is an implementation of Numerical Solutions of Systems of Linear Equations. In order to fulfill defined tasks, author comes out of algebra theory. He familiarizes himself with general, as well as, specific methods and analyzes implementation possibilities according to benefited knowledge. Author fluently passes through algorithm design and its definite software realization. In the final stage of dealing with this topic, primary functions of created software product, it´s potential and limitations are described. Stone of the thesis is closed with trial demonstrations of program usage.  **Kľúčové slová v AJ**  Matrix, linear, equation, Gauss, Jacobi, Gauss-Seidel, iteration, relaxation, Cholesky, method |



**Čestné vyhlásenie**

Vyhlasujem, že som celú bakalársku prácu vypracoval samostatne s použitím odbornej literatúry uvedenej na konci práce.

Košice, 24. máj 2012 ..........................................

*vlastnoručný podpis*

**Poďakovanie**

Týmto by som chcel vysloviť poďakovanie RNDr. Jánovi Bušovi, CSc. a RNDr. Ivanovi Daňovi, PhD. za cenné rady a motiváciu. Moja veľká vďaka taktiež patrí doc. Ing. Martinovi Tomášekovi, PhD. za venovaný čas a najmä možnosť riešiť túto tému na katedre informatiky.

**Predhovor**

Myšlienkou implementácie riešení sústav lineárnych rovníc som sa začal zaoberať počas absolvovania povinného kurzu *Matematická štatistika, numerika a pravdepodobnosť*, kde práve druhá tematická časť výučby zahŕňala kapitoly venované numerickým riešeniam sústav rovníc pozostávajúcich z lineárnych prvkov. Na jednej strane mi chuť bral nedostatok času na širokospektrálne pochopenie všetkých aspektov riešení v spomínanom období. Na strane druhej, očividná prepojenosť medzi algoritmami ponímanými pohľadom informatiky (ktorú študujem) a exaktnými postupmi numerickej matematiky ma priťahovala oveľa výraznejšie. Zásluhu pripisujem taktiež RNDr. Ivanovi Daňovi, PhD., ktorého motivujúce prednášky položili základný kameň môjho záujmu a po následnej konzultácii aj RNDr. Jánovi Bušovi, CSc., ktorému nadšenie zo záujmu o matematickú tému absolventa nematematickej katedry zjavne nechýbalo. Veľká vďaka patrí doc. Ing. Martinovi Tomášekovi, PhD., ktorý mi umožnil venovať sa tejto téme pod svojou záštitou.

Obsahovou náplňou sú teoretické základy použitých i alternatívnych metód, ich analýza a následné transformovanie navrhnutých algoritmov do formy softvérovej realizácie, ako vytýčeného cieľa predkladanej práce.

Je pravdou, že existuje viacero programových nástrojov pre manipuláciu so sústavami rovníc (*MatLab* - komerčný, *Octave* - nekomerčná verzia), no minimálne 1 charakteristika vytvoreného programu - Linear Systems Solver - predstavuje rozdiel oproti spomínaným softvérom, a tým je dynamické generovanie popisu riešenia realizovanej metódy nad zadaným systémom do formy FO a PDF súboru, čo vyplýva z požiadaviek, ktoré boli pre program definované. Naplnením požadovaných vlastností sa prvotná čisto empirická povaha programu preklenula do roviny edukatívnej, čo umožňuje absolventom vyššie spomínaného edukatívneho kurzu osvojiť si preberané postupy jednoduchšie a rýchlejšie vrátane možnosti porovnávania priebežných i konečných výsledkov výpočtov.

**Obsah**

[Zoznam obrázkov 11](#_Toc325667133)

[Zoznam tabuliek 13](#_Toc325667134)

[Zoznam symbolov a skratiek 14](#_Toc325667135)

[Matematické symboly a skratky 14](#_Toc325667136)

[Ostatné symboly a skratky 14](#_Toc325667137)

[Slovník termínov 15](#_Toc325667138)

[Úvod 16](#_Toc325667139)

[1 Numerické metódy riešenia sústav lineárnych rovníc 18](#_Toc325667140)

[1.1 Priame metódy 20](#_Toc325667141)

[1.1.1 Inverzná matica a Cramerovo pravidlo 20](#_Toc325667142)

[1.1.2 Gaussova eliminačná metóda 21](#_Toc325667143)

[1.1.3 Modifikovaná Gaussova eliminačná metóda s výberom hlavného prvku 24](#_Toc325667144)

[1.2 Nepriame metódy 25](#_Toc325667145)

[1.2.1 Jacobiho metóda 25](#_Toc325667146)

[1.2.2 Gauss-Seidelova metóda 26](#_Toc325667147)

[1.3 Porovnanie priamych metód 27](#_Toc325667148)

[1.4 Porovnanie nepriamych metód 27](#_Toc325667149)

[1.5 Ďalšie alternatívne postupy 28](#_Toc325667150)

[1.5.1 Relaxačná metóda 28](#_Toc325667151)

[2 Návrh algoritmov 30](#_Toc325667152)

[2.1 Algoritmus pre Gaussovu eliminačnú metódu 30](#_Toc325667153)

[2.2 Algoritmus pre modifikovanú Gaussovu eliminačnú metódu s čiastočným výberom hlavného prvku 31](#_Toc325667154)

[2.3 Algoritmus pre Jacobiho metódu 33](#_Toc325667155)

[2.4 Algoritmus pre Gauss-Seidelovu metódu 34](#_Toc325667156)

[3 Programová realizácia 35](#_Toc325667157)

[3.1 Funkcionalita programu 35](#_Toc325667158)

[3.2 Výber programovacieho prístupu a jazyka, implementácia 36](#_Toc325667159)

[3.3 Programové moduly 37](#_Toc325667160)

[4 Experimentálne overenie činnosti programu 40](#_Toc325667161)

[4.1 Vstupy 40](#_Toc325667162)

[4.1.1 Vstupy pre generátor matíc 40](#_Toc325667163)

[4.1.2 Vstupy časti programu, ktorá rieši matice 40](#_Toc325667164)

[4.2 Výstupy 42](#_Toc325667165)

[4.2.1 Výstup generátora matíc 42](#_Toc325667166)

[4.2.2 Výstupy časti programu, ktorá rieši matice 43](#_Toc325667167)

[4.3 Porovnanie rýchlosti programu pre konkrétne metódy 45](#_Toc325667168)

[4.4 Ukážkové príklady 46](#_Toc325667169)

[5 Záver 51](#_Toc325667170)

[Zoznam použitej literatúry 52](#_Toc325667171)

[Prílohy 53](#_Toc325667172)

[A. Používateľská príručka 54](#_Toc325667173)

[A.1 Začíname 55](#_Toc325667174)

[A.1.1 Spustenie aplikácie 56](#_Toc325667175)

[A.1.2 Hlavné menu 56](#_Toc325667176)

[A.1.3 Pár informácií nezaškodí 57](#_Toc325667177)

[A.2 Pracujeme s maticami 58](#_Toc325667178)

[A.2.1 Generátor matíc 58](#_Toc325667179)

[A.2.2 Podprogram pre riešenie sústav lineárnych rovníc 61](#_Toc325667180)

[A.3 Končíme 64](#_Toc325667181)

[A.4 Programujeme vlastné metódy 65](#_Toc325667182)

[A.4.1 Vytvorenie projektu 65](#_Toc325667183)

[A.4.2 Príprava 66](#_Toc325667184)

[A.4.3 Implementácia 70](#_Toc325667185)

[A.4.4 Zavedenie vytvorenej metódy do systému Linear Systems Solver 71](#_Toc325667186)

[B. Systémová príručka 73](#_Toc325667187)

[B.1 Základné informácie 74](#_Toc325667188)

[B.1.1 Výber programovacieho jazyka 74](#_Toc325667189)

[B.1.2 Výber vývojového prostredia 74](#_Toc325667190)

[B.1.3 Príprava prostredia 74](#_Toc325667191)

[B.1.4 Spúšťanie 76](#_Toc325667192)

[B.2 Dokumentácia 77](#_Toc325667193)

[B.2.1 core.App 77](#_Toc325667194)

[B.2.2 core.basicAspects.Informer 79](#_Toc325667195)

[B.2.3 core.DynamicHTMLGenerator 80](#_Toc325667196)

[B.2.4 core.DynamicPDFGenerator 81](#_Toc325667197)

[B.2.5 core.FOtoPDFconverter 88](#_Toc325667198)

[B.2.6 core.Generator 89](#_Toc325667199)

[B.2.7 exceptions.BaseDirectoryCreationException 95](#_Toc325667200)

[B.2.8 exceptions.MatrixSizeException 96](#_Toc325667201)

[B.2.9 exceptions.NegativeValueException 97](#_Toc325667202)

[B.2.10 exceptions.ReverseRangeValuesException 98](#_Toc325667203)

[B.2.11 linearSystems.data.MatrixD 98](#_Toc325667204)

[B.2.12 linearSystems.data.MatrixExecutionsHelper 101](#_Toc325667205)

[B.2.13 linearSystems.data.MatrixI 104](#_Toc325667206)

[B.2.14 linearSystems.data.VectorD 106](#_Toc325667207)

[B.2.15 linearSystems.data.VectorI 108](#_Toc325667208)

[B.2.16 linearSystems.dataHandlers.MatrixReader 110](#_Toc325667209)

[B.2.17 linearSystems.dataHandlers.MatrixWriter 111](#_Toc325667210)

[B.2.18 linearSystems.dataHandlers.MethodsClassLoader 113](#_Toc325667211)

[B.2.19 linearSystems.dataHandlers.MethodsClassLoader.JavaFileFilter 115](#_Toc325667212)

[B.2.20 linearSystems.exceptions.  
dominanceExceptions.ColumnlyIndominantMatrixException 116](#_Toc325667213)

[B.2.21 linearSystems.exceptions.  
dominanceExceptions.DiagonalyIndominantMatrixException 117](#_Toc325667214)

[B.2.22 linearSystems.exceptions.  
dominanceExceptions.RowlyIndominantMatrixException 118](#_Toc325667215)

[B.2.23 linearSystems.exceptions.MatrixEditableNotException 119](#_Toc325667216)

[B.2.24 linearSystems.exceptions.NoMethodsAsJavaClassesException 119](#_Toc325667217)

[B.2.25 linearSystems.exceptions.WrongDoubleFormatException 120](#_Toc325667218)

[B.2.26 linearSystems.interfaces.LinearSystemsSolvable Interface Reference 121](#_Toc325667219)

[B.2.27 linearSystems.interfaces.Stepable Interface Reference 122](#_Toc325667220)

[B.2.28 linearSystems.interfaces.ToleranceSetable Interface Reference 123](#_Toc325667221)

[B.2.29 linearSystems.methods.CholeskyMethod 124](#_Toc325667222)

[B.2.30 linearSystems.methods.GaussianMethod 129](#_Toc325667223)

[B.2.31 linearSystems.methods.GaussianMethodAdapted 133](#_Toc325667224)

[B.2.32 linearSystems.methods.GaussSeidelMethod 136](#_Toc325667225)

[B.2.33 linearSystems.methods.JacobiMethod 140](#_Toc325667226)

[B.2.34 rounder.DoubleRounder 145](#_Toc325667227)

[B.2.35 swingUI.AboutFrame 146](#_Toc325667228)

[B.2.36 swingUI.BasicGeneratorFrame 148](#_Toc325667229)

[B.2.37 swingUI.BasicGeneratorFrame.OpenFileButton 155](#_Toc325667230)

[B.2.38 swingUI.BasicSolverFrame 155](#_Toc325667231)

[B.2.39 swingUI.BasicSolverFrame.OpenFileButton 161](#_Toc325667232)

[B.2.40 swingUI.ParentFrame 162](#_Toc325667233)

[B.2.41 swingUI.StartFrame 163](#_Toc325667234)

Zoznam obrázkov

Obr. 1 Porovnanie konvergencie štyroch rôznych iteračných metód na riešenie sústav lineárnych rovníc, prostá iterácia, Jacobiho, Gauss-Seidelova a superrelaxačná metóda (pozri 1.5.1) pre náhodne generovanú maticu [9] 28

Obr. 2 Diagram prípadov použitia aplikácie Linear Systems Solver 36

Obr. 3 Základné komponenty programu Linear Systems Solver (abstrakcia) 37

Obr. 4 Diagram reprezentujúci komunikáciu medzi systémom Linear Systems Solver a FOP technológiou 38

Obr. 5 Sekvenčný diagram programu Linear Systems Solver 39

Obr. 6 Aplikačné okno generátora matíc 40

Obr. 7 Sústava (1) zapísaná v súbore podľa uvedených požiadaviek (bez zarovnávania) 41

Obr. 8 Sústava (1) zapísaná v súbore podľa uvedených požiadaviek (so zarovnaním) 41

Obr. 9 Aplikačné okno modulu pre riešenie matíc 42

Obr. 10 Výstup generátora matíc pre maticu 5x6 s využitím 2 desatinných miest 42

Obr. 11 Zobrazenie výstupu generátora matíc 43

Obr. 12 Aplikačné okno programu po ukončení výpočtu 44

Obr. 13 Dynamicky vytváraný popis postupu výpočtu pre používateľom zadávanú maticu 44

Obr. 14 Demonštrácia riešenia príkladu 1 pomocou Linear System Solver 46

Obr. 15 Zápis matice z príkladu 2 do súboru 47

Obr. 16 Nájdenie riešenia k príkladu pomocou programu Linear Systems Solver 48

Obr. 17 Riešenie príkladu získané programom Linear Systems Solver 49

Obr. 10 Obsah adresára LinearSystemsSolver 55

Obr. 11 Hlavné menu programu Linear Systems Solver 56

Obr. 12 Okno so stručnými informáciami o projekte Linear Systems Solver 57

Obr. 13 Aplikačné okno generátora matíc 58

Obr. 14 Prehľadávač súborov 59

Obr. 15 Ukážkový príklad pre generovanie matice 60

Obr. 16 Generovaná matica uložená v súbore 60

Obr. 17 Aplikačné okno podprogramu pre riešenie matíc 61

Obr. 18 Položky pre nastavenie charakteristík výpočtu 62

Obr. 19 Nezaškrtnuté položky pre rôzne zobrazenia výsledku výpočtu 62

Obr. 20 Ukážka PDF výstupu pre Jacobiho metódu 63

Obr. 21 Korektné ukončenie aplikácie 64

Obr. 22 Vytvorenie nového Java projektu 65

Obr. 23 Vytvorený projekt v Package Exploreri 65

Obr. 24 Vloženie nového balíka linearSystems.methods do adresára src 66

Obr. 25 Náš projekt po vytvorení nového balíka linearSystems.methods 66

Obr. 26 Náš projekt po vytvorení triedy reprezentujúcej novú metódu 67

Obr. 27 Obsah triedy MyNewMethod 67

Obr. 28 Vloženie knižnice linearSystems.jar 68

Obr. 29 Náhľad na projekt v Eclipse po pridaní knižnice linearSystems.jar 68

Obr. 30 Rozšírenie triedy MyNewMethod o implementovanie rozhrania LinearSystemsSolvable 69

Obr. 31 Automatické vloženie metód potrebných na správnu funkciu rozširovaného programu 70

Obr. 32 Demonštračný príklad implementácie 70

Obr. 33 Premiestnenie vytvorenej metódy do adekvátneho adresára programu Linear Systems Solver 71

Obr. 34 Aplikácie vytvorenej metódy v bežiacom programe 72

Obr. 35 Náhľad na pripravený projekt v záložke Package Explorer 75

Obr. 36 Nastavenie classpath (Java build path) priamo v Eclipse 75

Obr. 37 Hlavné menu aplikácie Linear Systems Solver 76

Obr. 38 Diagram dedičnosti pre linearSystems.interfaces.LinearSystemsSolvable 121

Obr. 39 Diagram dedičnosti pre linearSystems.interfaces.Stepable 122

Obr. 40 Diagram dedičnosti pre linearSystems.interfaces.ToleranceSetable 123

Obr. 41 Diagram dedičnosti pre linearSystems.methods.CholeskyMethod 124

Obr. 42 Diagram dedičnosti pre linearSystems.methods.GaussianMethod 129

Obr. 43 Diagram dedičnosti pre linearSystems.methods.GaussianMethodAdapted 133

Obr. 44 Diagram dedičnosti pre linearSystems.methods.GaussSeidelMethod 136

Obr. 45 Diagram dedičnosti pre linearSystems.methods.JacobiMethod 140

Obr. 46 Diagram dedičnosti pre swingUI.AboutFrame 146

Obr. 47 Diagram dedičnosti pre swingUI.BasicGeneratorFrame 148

Obr. 48 Diagram dedičnosti pre swingUI.BasicSolverFrame 155

Obr. 49 Diagram dedičnosti pre swingUI.ParentFrame 162

Obr. 50 Diagram dedičnosti pre swingUI.StartFrame 163

Zoznam tabuliek

Tab. 1 Časy trvania generovania matíc vhodných pre príslušný druh metódy 45

Tab. 2 Časy trvania výpočtu riešenia rôznych metód pre matice rôzneho typu a rôznych rozmerov 45

Tab. 3 Tabuľka známych informácií vyplývajúcich zo slovnej úlohy 48

Zoznam symbolov a skratiek

Matematické symboly a skratky

 determinant matice (tiež det **A** alebo |**A**|)

 všeobecný kvantifikátor

**0** nulová matica

**A** matica A

**A**-1inverzná matica A

**A***r*rozšírená matica A

**A***T*transponovaná matica A

GEM **G**aussova **e**liminačná **m**etóda

*h(***A***)* hodnosť matice A

Ostatné symboly a skratky

€ Euro

AOP **A**spect-**O**riented **P**rogramming

API **A**pplication **P**rogramming **I**nterface

FOP **F**ormatting **O**bject **P**rocessor

GB **G**iga**B**yte

GHz **G**iga**H**ertz

HTML **H**yper**t**ext **M**arkup **L**anguage

JDK **J**ava **D**evelopment **K**it

MB **M**ega**B**yte

ms **milisekunda**, 10-3 sekundy

OOP **O**bject-**O**riented **P**rogramming

OS **O**perating **S**ystem

PDF **P**ortable **D**ocument **F**ormat

PL **P**resentation **L**anguage

s **sekunda**, jednotka času v sústave SI

Slovník termínov

**Balík** je množina spolu súvisiacich tried a rozhraní zabezpečujúca riadenie prístupu a manažment mien.

**Framework** je programová štruktúra, ktorá slúži ako podpora pri programovaní, vývoji a organizácií iných programových projektov. Môže obsahovať podporné programy, knižnice , podporu pre návrhové vzory alebo doporučené postupy pri vývoji.

**Javadoc** je nástroj na generovanie API (zbierka funkcií a tried ktoré určujú akým spôsobom sa majú funkcie knižníc volať zo zdrojového kódu programu) dokumentácie v HTML formáte z dokumentačných komentárov v zdrojovom kóde.

**Matica** je systémom *m* x *n* čísel (reálnych alebo komplexných), ktoré sú usporiadané do obdĺžnikového formátu o *m* riadkoch a *n* stĺpcoch.

**Open source** je vo všeobecnosti akákoľvek informácia dostupná verejnosti, za podmienky, že možnosť jej slobodného šírenia zostane zachovaná. Open-source softvér musí mať dostupný zdrojový kód. Koncoví používatelia musia mať právo voľne používať, modifikovať a šíriť softvér, ako aj právo baliť a predávať produkt.

**Riedka matica** je matica, ktorej počet nenulových prvkov je výrazne menší než počet jej všetkých prvkov.

**Súborový systém** je spôsob ukladania a organizovania počítačových súborov a údajov, ktoré obsahujú, a to tak, aby k nim bol umožnený jednoduchý prístup.

**Transponovaná matica** je matica, ktorá vznikne z matice **A** typu *m* . *n* zámenou riadkov a stĺpcov.

**Vedúci prvok riadku matice** je prvý nenulový prvok. Ak všetky prvky riadku matice sú nulové, potom matica nemá vedúci prvok.

**Vektor** je usporiadanou *n*-ticou čísel, kde *n* vyjadruje jeho rozmer.

Úvod

Sústavy lineárnych rovníc. Význam existencie problémov tohto typu v reálnom svete dokazuje fakt, že so štandardnými postupmi hľadania ich riešení sa stretávame nielen na technicky ale aj netechnicky zameraných vysokých školách. Akokoľvek, napriek tomu, že mnohí si osvoja základné poznatky a postupy realizácie vyučovaných metód, málokto si je aj vedomý reálneho použitia rozoberanej problematiky v technickej (ba aj netechnickej) praxi.

Aplikácia metód rozoberaných v bakalárskej práci, ktorú práve čítate, má širokú základňu. Sústavy lineárnych rovníc sa uplatňujú v oblastiach fyziky (charakteristika elektrických obvodov, optika, kvantová mechanika), ekonomiky a podnikania (modelovanie ekonomických procesov), mechaniky a elektroniky (lineárne stavové rovnice), virtuálnej reality a kybernetiky (napr. transformácie z 3-dimenzionálneho priestoru do 2-dimenzionálneho priestoru obrazovky, robotický pohyb), atď.

Téma sústav lineárnych rovníc je známa asi od roku 300 pred naším letopočtom – 200 nášho letopočtu, kedy sa ňou prvýkrát zaoberal čínsky matematik Jiu Zhang Suan Shu. Stále sa však vynakladá úsilie o efektívnejšie algoritmy riešenia sústav, čo však nie je cieľom predkladanej bakalárskej práce.

Hlavnou úlohou je implementovať dosiaľ známe metódy riešenia sústav lineárnych rovníc tak, aby boli nielen prostriedkom dosiahnutia hľadaného riešenia, ale aj prínosom pre samotného používateľa z výchovno-vzdelávacieho pohľadu. Laicky povedané, pre ľubovoľnú riešiteľnú sústavu rovníc má program nájsť správne riešenie vrátane možnosti zobrazenia celého postupu riešenia, čo ocenia pravdepodobne najmä študenti počas príprav na zápočtové previerky či skúšky.

Zdrojmi informácií sú najmä internetové zdroje, aj keď väčšina implementovaných algoritmov vychádza z teoretického základu matematických kníh uvedených v zozname použitej literatúry.

Tomu však predchádza séria kapitol, ktoré sú na seba nadväzujúce. V prvej kapitole je formulovaná úloha s uvedením bližších požiadaviek, ktoré majú byť počas práce naplnené. Druhá kapitola, ako už názov napovedá, je skutočnou analýzou problematiky numerických metód zameraných na riešenie sústav lineárnych rovníc, kde sa už hlbšie rozoberajú východiská pre položenie teoretických základov, podmienky algoritmizácie a dopad na samotnú implementáciu. Matematické definície aj abstraktnejší popis najpoužívanejších postupov tak, ako ho vníma autor práce, je náplňou tretej kapitoly. V jej prvej podkapitole sa venuje presným metódam, v druhej zasa metódam približným, tretia podkapitola je sumou sumáru nadobudnutých poznatkov (s poukázaním na vlastnosti potenciálne súvisiacich s programovou realizáciou). Štvrtá kapitola je transformáciou získaných vedomostí do formy algoritmov vyjadrených pomocou zadefinovaného pseudojazyka. Funkcionalitu, komponenty, teda architektúru a ďalšie možnosti rozšírenia programu, môžete nájsť charakterizované v piatej tematickej časti, ktorá úzko súvisí s demonštráciou práce programu na základných príkladoch, čo predstavuje poslednú kapitolu predkladanej práce.

1. Numerické metódy riešenia sústav lineárnych rovníc

Sústavou lineárnych rovníc je systém *m* rovníc o *n* neznámych, tvaru [1]:

 (1)

Čísla  nazývame koeficientmi, čísla  absolútnymi členmi a  neznámymi sústavy ().

Sústavu () môžeme zapísať pomocou maticovej rovnice

, (2)

kde

 (3)

je maticou koeficientov sústavy (),

 (4)

je maticou absolútnych členov[[1]](#footnote-1) a

 (5)

je maticou neznámych[[2]](#footnote-2) sústavy ().

Ak v zápise v maticovom tvare chceme uviesť aj absolútne členy, sústavu () potom zapíšeme ako

 (6)

a nazývame ju rozšírenou maticou sústavy [2].

Ďalšou formou, s ktorou sa stretneme v nasledujúcich kapitolách, je tzv. *dolná trojuholníková matica* (v tomto prípade rozšírená o vektor pravých strán) a vyzerá nasledovne:

. (7)

Čiže  platí .

Matica, ktorej všetky prvky sú rovné nule, sa nazýva nulová matica a označuje sa **0[[3]](#footnote-3)**.

**Veta:** Hodnosť každej nenulovej matice sa rovná maximálnemu počtu nenulových riadkov s ňou ekvivalentnej trojuholníkovej matice.

Riešením sústavy () sa nazýva každá usporiadaná *n*-tica reálnych čísel , ktorá spĺňa všetky rovnice sústavy (). Vyriešiť sústavu znamená nájsť všetky jej riešenia. Počet riešení pre danú sústavu charakterizuje **Frobeniova veta**:

Sústava lineárnych rovníc () má riešenie práve vtedy, ak hodnosť matice sústavy () sa rovná hodnosti rozšírenej matice sústavy (), t.j. .

* Ak , kde  je počet neznámych sústavy (), potom sústava () má práve jedno riešenie.
* Ak , potom sústava () má nekonečne veľa riešení.
* Ak , potom sústava () nemá riešenie.
* My sa však zameriame na systémy *n* lineárnych rovníc o *n* neznámych, a to použitím rozličných metód, ktoré zaraďujeme do dvoch skupín: *priame* a *nepriame*.
  1. Priame metódy

Priame metódy nazývame aj *presné* alebo *finitné*. Skupina finitných metód ponúka riešenie sústavy pomocou konečného algoritmu, zahŕňa Gaussovu metódu, Cramerovo pravidlo, metódu vedúcich prvkov, metódu odmocnín a iné. Vo svojej podstate sú síce priame metódy presné, ale k určitým chybám sa pri výpočtoch nevyhneme, a to z dôvodu zaokrúhľovania výsledkov. Odhadnúť chybu výpočtu je relatívne zložité, no pri nepriamych metódach sa k týmto nepresnostiam pridáva aj vlastná chyba metódy [3].

* + 1. Inverzná matica a Cramerovo pravidlo

Nech je daná sústava (1), ktorá má tú vlastnosť, že :

 (8)

Označme

 (9)

maticu koeficientov sústavy (8),



vektor pravých strán a



vektor neznámych.

Ak je matica **A** regulárna, teda jej determinant je nenulový , potom platí, že sústava (8) má práve jedno riešenie[[4]](#footnote-4) [3].

Tým, že platí vzťah (2), môžeme vynásobením oboch strán rovnice (2) inverznou maticou dospieť k riešeniu:



 (10)

Ak je stupeň *n* matice **A** väčší ako 4, *n* > 4, určenie samotnej inverznej matice  je zdĺhavé, preto sa vzťah (10) používa v praxi len málokedy. Ak by sme zvolili riešenie pomocou Cramerových vzorcov  kde  sú determinanty získané z determinantu  matice **A** nahradením *i*-tého stĺpca matice **A** vektorom pravých strán, pre sústavy lineárnych rovníc o *n* neznámych sa riešenie uberá k výpočtu *n* + 1 determinantov stupňa *n*, čo je veľmi prácne, ak *n* nadobúda vysoké hodnoty. Preto boli vyvinuté iné alternatívne spôsoby.

* + 1. Gaussova eliminačná metóda

Najrozšírenejšou metódou na riešenie sústav lineárnych rovníc je bezpochyby Gaussova eliminačná metóda [3]. Možno ju uplatniť pri riešení nielen sústav *n* rovníc o *n* neznámych, ale aj pri maticiach ľubovoľných rozmerov. Napriek tomu sa v našom výklade obmedzíme len na jednoduchšie systémy rovníc.

Majme sústavu troch rovníc o troch neznámych

 (11)

Nech . Delením prvej rovnice sústavy (11) číslom , dostávame:

, (12)

pričom

 .

Prvok, ktorým sme delili každý člen rovnice, nazývame *vedúcim prvkom* (v tomto prípade prvok ).

Pomocou rovnice (12) môžeme zbaviť sústavu (11) premennej  nasledujúcim postupom:

1. Od druhej rovnice sústavy (11) odčítame rovnicu (12), ktorú predtým vynásobíme hodnotou .
2. Od tretej rovnice sústavy (11) odčítame rovnicu (12), ktorú predtým vynásobíme hodnotou .

Dostávame sústavu rovníc, ktorá je pozbavená prvkov, :



ktorej koeficienty  sú určené vzťahom .

Keď prvú rovnicu sústavy

 (11´)

predelíme vedúcim prvkom , dostávame

, (12´)

kde

.

Takým istým spôsobom, ako v predchádzajúcom prípade, môžeme aj zo sústavy (11´) vylúčiť neznámu .

Z pohľadu pôvodnej sústavy máme

 (13)

kde .

Z poslednej rovnice sústavy (13) už nie je problémom vyjadriť neznámu :

. (13´)

Doposiaľ dosiahnutú časť výpočtu označujeme ako priamy chod. Nasledujúcu časť- proces vyjadrovania a samotných výpočtov zvyšných neznámych, nazývame *spätným chodom*, resp. *spätnou substitúciou*.

Premennú  už vyjadrenú máme (13), zostávajú ešte neznáme  a . Tie určíme v opačnom poradí zo vzťahov (12´) a (12):

,

.

Ako vidíme, sústava (13) bola trojuholníkovou maticou, ktorú môžeme znázorniť nasledujúco:

.

Inak povedané, celý náš postup spočíval v aplikovaní ekvivalentných riadkových úprav na maticu sústavy rozšírenú o stĺpec pravých strán (rozšírená matica sústavy), čím sme dostali stupňovitý (horný trojuholníkovitý) tvar, na ktorý sme následne aplikovali spätný chod s účelom už samotného vyjadrenia neznámych [4].

Metódy popísané v kapitole 1.1.1 sa dajú aplikovať len na štvorcové systémy a baviac len na tie, ktorých matica systému je regulárna. GEM je metóda všeobecná, použiteľná pre riešenie ľubovoľnej lineárnej sústavy [5].

* + 1. Modifikovaná Gaussova eliminačná metóda s výberom hlavného prvku

Existujú však matice, ktoré nie sú vhodné pre riešenie pomocou GEM popísanej v kapitole 1.1.2. Preto boli vyvinuté jej rôzne modifikácie, ktoré tieto aspekty nevhodnosti odstraňujú. Okrem nevhodnej matice existuje aj ďalší dôvod, prečo nemožno Gaussovu eliminačnú metódu použiť, a to sú numerické aspekty. Ich podstata spočíva v nepresnosti aritmetických operácií realizovaných počítačom. Môžeme sa s nimi stretnúť, keď multiplikátory alebo prvky redukovanej matice sústavy sú v absolútnej hodnote veľké čísla, a preto rozdiel medzi presným a vypočítaným riešením je značný, najmä pri maticiach veľkých rozmerov (tzv. *numerická nestabilita metódy*) [6].

Jednoduchým príkladom môže byť matica

,

pri riešení ktorej dvomi cestami dochádzame k diametrálne rozličným výsledkom. Zaokrúhľujme na tri desatinné miesta. Pre realizáciu Gaussovej eliminačnej metódy bez výberu hlavného prvku volíme multiplikátor , po realizácii priameho a spätného chodu dostávame výsledok .

Ak však za vedúci prvok zvolíme prvý prvok druhého riadku  , multiplikátor nadobúda hodnotu  a úpravami sa dostávame k výsledku .

Znamená to, že výhodnejšie je vyberať za hlavné prvky tie, ktoré sú v absolútnej hodnote najväčšie, čím sa snažíme eliminovať chyby zaokrúhľovania. Ak vyberáme v danej fáze hlavný prvok zo všetkých prvkov, ktoré prichádzajú do úvahy, hovoríme o *úplnom* výbere hlavného prvku. Ak vyberáme hlavný prvok len z niektorých prvkov (napr. len z jedného riadku alebo stĺpca matice alebo submatice) hovoríme *čiastočnom* výbere hlavného prvku.

* 1. Nepriame metódy

Nepriame metódy sa zvyknú označovať aj ako približné alebo iteračné   
(tiež matrix-splitting). Pomocou nich sme schopní riešiť sústavy rovníc s požadovanou presnosťou na základe konvergencie nekonečných procesov. Ich využitie spočíva najmä pri riešení riedkych matíc, t.j. matíc, ktoré obsahujú relatívne málo nenulových prvkov[[5]](#footnote-5) [3].

* + 1. Jacobiho metóda

Deklarujme princíp Jacobiho metódy na systéme 3x3, čiže

 (14)

Premenné označené rámčekom vyjadríme, t.j.

 (15)

všeobecne



Dostali sme takto iteračnú postupnosť Jacobiho metódy, vyjadrenú funkciou  takto:

. (16)

V prvom kroku si podľa vlastného uváženia volíme počiatočnú aproximáciu [[6]](#footnote-6) a jej dosadením do vzťahu (15) získavame prvú aproximáciu . Takto postupne dosadzujeme vypočítané hodnoty, až pokiaľ nedosiahneme požadovanú presnosť (t.j.), ináč povedané, iteráciu ukončíme vtedy, keď sa na požadovanom desatinnom mieste čísla cifra nemení ďalším iterovaním alebo pokiaľ nevykonáme požadovaný počet krokov [7].

Jacobiho metódou nemusíme nájsť riešenie stále, čo sa prejaví divergenciou aproximácií. Ak je matica sústavy (14) ostro riadkovo alebo ostro stĺpcovo dominantná, Jacobiho metóda určite konverguje.

Definícia *ostrej riadkovej dominancie* je

, (17)

a *ostrej stĺpcovej dominancie* je

. (18)

Preto, ak si nie sme istý konvergenciou, je vhodné stanoviť maximálny počet krokov, po ktorých vieme ukončiť výpočet s tým, že metóda diverguje. V tom prípade je potrebné zvoliť inú metódu alebo sústavu nejako upraviť [7].

* + 1. Gauss-Seidelova metóda

Gauss-Seidelova metóda je akousi pozmenenou formou Jacobiho metódy. Podstata spočíva v modifikácii vyjadrenia závislostí neznámych oproti forme (15), a to nasledovne:



čo teda znamená, že pri *k*-tom výpočte neznámej  berieme do úvahy už vypočítané   
*k*-té aproximácie neznámych ,,..., z práve realizovanej iterácie a (*k-*1)-*vé* aproximácie neznámych ,, ..,  pochádzajúcich z predchádzajúcej iterácie.

Všeobecne (zložkovo):

.

Ak má matica  vlastnosť ostrej diagonálnej dominantnosti, potom   
Gauss-Seidelova metóda konverguje pre ľubovoľnú začiatočnú hodnotu vektora . Ak je matica  symetrická a kladne definitná (viď nižšie), potom Gauss-Seidelova metóda konverguje pre ľubovoľný začiatočný vektor .

Riešená matica  je *kladne definitná* vtedy a len vtedy, ak platí

 ( je teda symetrickou maticou)

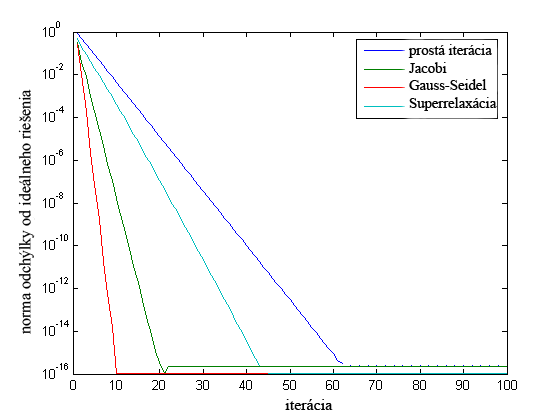
a pre všetky nenulové vektory



platí

[[7]](#footnote-7). (19)

* 1. Porovnanie priamych metód
* Pri Cramerovom pravidle jednotlivé neznáme počítame ako podiely determinantov, preto pravidlo je vhodné len pre veľmi malé sústavy rovníc [8].
* Vplyv chýb zaokrúhľovania pri GEM môže byť značný, preto sa radšej používa GEM s výberom hlavného prvku.
* GEM je náročná z časového aj pamäťového hľadiska, najlepšie sa hodí pre nie príliš rozsiahle sústavy reprezentované riedkou maticou.
  1. Porovnanie nepriamych metód
* Konvergencia Gauss-Seidelovej metódy je pre mnohé matice rýchlejšia než konvergencia Jacobiho metódy.
* Existujú matice, pre ktoré Gauss-Seidelova metóda konverguje a Jacobiho nie a naopak.
* Gauss-Seidelovej metóda má pri programovom riešení menšie nároky na pamäť, keďže nie je potrebné uchovávať hodnoty predchádzajúcej aproximácie.
* Jacobiho metóda umožňuje paralelný výpočet, zatiaľ čo Gauss-Seidelova metóda je zo svojej podstaty sekvenčná [7].



Obr. 1 Porovnanie konvergencie štyroch rôznych iteračných metód na riešenie sústav lineárnych rovníc, prostá iterácia, Jacobiho, Gauss-Seidelova a superrelaxačná metóda (pozri 1.5.1) pre náhodne generovanú maticu [9]

* 1. Ďalšie alternatívne postupy
     1. Relaxačná metóda

Nech je daná sústava *n* lineárnych rovníc o *n* neznámych:

 (20)

Prvým krokom relaxačnej metódy je úprava rovníc na tvar vhodný pre relaxáciu [3]. Tvar vhodný pre relaxáciu dostaneme tak, že pravé strany rovníc premiestnime na ľavé strany rovníc a každú rovnicu predelíme zápornou hodnotou diagonálneho prvku (pre *k*-tú rovnicu je to prvok - ).

Náš pôvodný systém (20) nadobudol formu

 (21)

kde

 a .

Nech  je počiatočnou aproximáciou sústavy (21). Dosadením týchto hodnôt do sústavy (21) dostávame tzv. *rezíduá*:

 (22)

Zväčšením jednej neznámej  o prírastok  sa zmenší príslušné rezíduum  o hodnotu , ostatné rezídua  sa zmenia o hodnoty . Ak teda chceme ďalšie rezíduum anulovať, musíme zväčšiť číslo  o hodnotu ; z toho vyplýva, že  a , pre .

Relaxačná metóda vo svojej najjednoduchšej forme záleží na tom, že pri každom kroku anulujeme zmenou hodnoty aproximácie príslušnej neznámej to rezíduum, ktoré je v absolútnej hodnote najväčšie. Proces ukončíme vtedy, keď sa všetky rezídua rovnajú nule s požadovanou presnosťou.

1. Návrh algoritmov

Z dôvodu neskoršej implementácie vybraných metód, skúsme pre ne navrhnúť jednotlivé algoritmy pomocou jazyka PL.

PL jazyk je expresívny transparentný jazyk slúžiaci na prezentáciu algoritmov, zložený z príkazov na zostavovanie programov. Podstatou je názorný popis funkcie programu, s tým, že sa upúšťa (abstrahuje) od implementačných detailov konkrétneho programovacieho jazyka.

Predpokladajme, že samotnému popisu konkrétneho algoritmu predchádza načítanie matice a uzatvára ho výpis nájdeného riešenia, čo môžeme názorne demonštrovať takto:

**comment** citaj maticu o rozmere *n* x *(n+1)*

**write** „Citam maticu o rozmere n x (n+1)“;

matica ← nacitajCeluMaticu();  
*pocetRiadkov* ← n;  
*pocetStlpcov* ← n+1;

***algoritmusPreKonkrétnuMetódu***

**comment** tlac vysledok  
**for** *i*←1 **step** 1**until** *i*<=*pocetRiadkov* **do**

**write** „xi = “ xi;

* 1. Algoritmus pre Gaussovu eliminačnú metódu

**comment** vykonaj priamy chod  
**for** *i*←1 **step** 1**until** *i*<*pocetRiadkov* **do**

**begin**

**comment** predel veduci riadok hodnotou na diagonale

**for** *j*←1 **step** 1**until** *j* <= *pocetStlpcov* **do**

*matica(i*,*j)*← *matica(i*,*j)*: *matica(i*,*i)*;

**comment** ku kazdemu riadku *k* pod veducim pripocitaj   
**comment** -1 x *matica(k,i)*nasobok veduceho riadku *i*

**for** *k*←*i+1* **step** 1**until** *k* <= *pocetRiadkov* **do**

**begin**

**comment** skopiruj povodny veduci riadok

*veduciRiadok* ← *matica(i);* **comment** vynasob ho hodnotou -1 x *matica(k,i)*

*veduciRiadok* ← *veduciRiadok* \* *(*-1 x *matica(k,i))*;

**comment** scitaj veduci riadok a *k*-ty riadok

*matica(k)* ← *veduciRiadok* + *matica(k)*;

**end**

**end**

**comment** vykonaj spatny chod

**comment** vypocitaj neznamu xn

xn ← *matica(pocetRiadkov*, *pocetStlpcov)* : *matica(pocetRiadkov*, *pocetStlpcov)*;

**comment** vypocitaj ostatne nezname

**for** *m*←(*pocetRiadkov*-1) **step** -1**until** *m*>=1 **do**

**begin**

**for** *prem*←*m+1* **step** 1**until** *prem* <= *pocetRiadkov* **do**

**begin**

*lavaStrana* ← *lavaStrana* + *matica(m*,*prem)* \* *x*prem;

**end**

xm = *matica(m*, *pocetStlpcov)* - *lavaStrana*;

*lavaStrana* ← 0;

**end**

* 1. Algoritmus pre modifikovanú Gaussovu eliminačnú metódu s čiastočným výberom hlavného prvku

**comment** vykonaj priamy chod vratane preusporiadania riadkov  
**for** *i*←1 **step** 1**until** *i*<*pocetRiadkov* **do**

**begin**

**comment** preusporiadaj riadky, ak prvok na diagonale nie je

**comment** maximalnym v absolutnej hodnote

*islargest* ← isDiagElemMax(*i*);

**if** *islargest* = FALSE **then** reorganizeMatrix(*i*);

**comment** predel veduci riadok hodnotou na diagonale

**for** *j*←1 **step** 1**until** *j* <= *pocetStlpcov* **do**

**matica(i,j)** ← matica(*i*,*j*): matica(*i*,*i*);

**comment** ku kazdemu riadku *k* pod veducim pripocitaj   
**comment** -1 x *matica(k,i)*nasobok veduceho riadku *i*

**for** *k*←*i+1* **step** 1**until** *k* <= *pocetRiadkov* **do**

**begin**

**comment** skopiruj povodny veduci riadok

*veduciRiadok* ← *matica(i);* **comment** vynasob ho hodnotou -1 x *matica(k,i)*

*veduciRiadok* ← *veduciRiadok* \* *(*-1 x *matica(k,i))*;

**comment** scitaj veduci riadok a *k*-ty riadok

*matica(k)* ← *veduciRiadok* + *matica(k)*;

**end**

**end**

**comment** vykonaj spatny chod

**comment** vypocitaj neznamu xn

xn ← *matica(pocetRiadkov*, *pocetStlpcov)* : *matica(pocetRiadkov*, *pocetStlpcov)*;

**comment** vypocitaj ostatne nezname

**for** *m*←(*pocetRiadkov*-1) **step** -1**until** *m*>=1 **do**

**begin**

**for** *prem*←*m+1* **step** 1**until** *prem* <= *pocetRiadkov* **do**

**begin**

*lavaStrana* ← *lavaStrana* + *matica(m*,*prem)* \* *x*prem;

**end**

xm = *matica(m*, *pocetStlpcov)* - *lavaStrana*;

*lavaStrana* ← 0;

**end**

**comment** zisti, ci prvok na diagonale je maximalny v absolutnej hodnote

**function** isDiagElemMax(*i*):

isMax ← *TRUE*;

**for** *p*←*i* **step** 1**until** *p* <= *pocetRiadkov* **do**

**if** *|matica(i*,*i)|>= |matica(p*,*i)|*

**then** isMax← *TRUE*;

**else**

**begin**

isMax←*FALSE*;

**return** isMax;

**end**

**return** isMax;

**comment** preusporiadaj maticu

**procedure** reorganizeMatrix(*i*) :

**comment** najdi index prvku, ktory predstavuje v danom stlpci maximum

*tempMax* ← *|matica(i*,*i)|* ;

*indexOfMax* ← *i*;

**for** *p*←*i* **step** 1**until** *p* <= *pocetRiadkov* **do**

**if** *tempMax < |matica(p*,*i)|* **then**

**begin**

*tempMax* ←*|matica(p*,*i)|*;

*indexOfMax* ← *p*;

**end**

**comment** ak diagonalny prvok nie je maximalny v abs.hod., potom ho **comment** vymen s tym, ktory je max.

**if** *indexOfMax* != *i* **then** vymenRiadky(i,indexOfMax);

**comment** vymen riadky s indexom *diagonal* a *indexVacsieho*

**procedure** vymenRiadky(*diagonal*,*indexVacsieho*) :

*novaMatica* ← *matica*;

**comment** na riadok diagonal umiestni riadok indexVacsieho a naopak

*novaMatica(diagonal)* ← *matica(indexVacsieho)*;

*novaMatica(indexVacsieho)* ← *matica(diagonal)*;

* 1. Algoritmus pre Jacobiho metódu

**comment** predel kazdy prvok hodnotou prvku na diagonale v prislusnom riadku

**for** i←1 **step** 1 **until** i<=pocetRiadkov **do**

**for** j←1 **step** 1 **until** i<=pocetStlpcov **do**

**begin**

matica(i,j) ← matica(i,j) : matica(i,i);

**end**

**comment** nastav pociatocnu aproximaciu, nech kazda premenne v 0.tom kroku =0

**for** *i*←1 **step** 1 **until** *i*<=*pocetRiadkov* **do**

*xi*(*0*) ← 0;

**comment** nacitaj pocet pozadovanych krokov

**write** „Kolko iteracii chces realizovat?“;

**read** *total*;

**comment** chce pouzivatel priebezne monitorovat aj presnost

**write** „Pocitat rozdiel hodnot prislusnych iteracii?“;

**read** *jePresnostPocitana*;

**comment** iteruj *total*-krat

*krok* ← 1;

**while** *krok*<=*total* do

**begin**

**for** *i*←1 **step** 1 **until** *i*<=*pocetRiadkov* **do**

**begin**

**for** *j*←1 **step** 1 **until** *j*<*pocetStlpcov* **do**

**if** *j*!=*i* **then**

*zvysok* ← *zvysok* + *matica(i*,*j)*\**xj*(*krok-1*);

**comment** vyrataj samotnu hodnotu aproximacie *Xi*(krok)

*xi*(*krok*) ← *matica(pocetRiadkov*,*pocetStlpcov) - zvysok;*

**if** *krok* = *total* AND *jePresnostPocitana* = *TRUE* **then**

*polePrePresnost(i)* ← *xi*(*krok*)- *xi*(*krok-1*); [[8]](#footnote-8)

**end**

*zvysok* ← 0;

*krok* ← *krok* + 1;

**end**

**comment** tlac presnost, ak ju chcel pouzivatel pocitat

**if** *jePresnostPocitana* = *TRUE* **then**  
**for** *i*←1 **step** 1**until** *i*<=*pocetRiadkov* **do**

**write** „Rozdiel medzi hodnotami xi(poslednyKrok) a xi(poslednyKrok-1) pre i =“ i „ je “ *polePrePresnost(i)*;

* 1. Algoritmus pre Gauss-Seidelovu metódu

**comment** predel kazdy prvok hodnotou prvku na diagonale v prislusnom riadku

**for** i←1 **step** 1 **until** i<=pocetRiadkov **do**

**for** j←1 **step** 1 **until** i<=pocetStlpcov **do**

**begin**

matica(i,j) ← matica(i,j) : matica(i,i);

**end**

**comment** nastav pociatocnu aproximaciu, nech kazda premenne v 0.tom kroku =0

**for** *i*←1 **step** 1 **until** *i*<=*pocetRiadkov* **do**

*xi*(*0*) ← 0;

**comment** nacitaj pocet pozadovanych krokov

**write** „Kolko iteracii chces realizovat?“;

**read** *total*;

**comment** chce pouzivatel priebezne monitorovat aj presnost

**write** „Pocitat rozdiel hodnot prislusnych iteracii?“;

**read** *jePresnostPocitana*;

**comment** iteruj *total*-krat

*krok* ← 1;

**while** *krok*<=*total* do

**begin**

**for** *i*←1 **step** 1 **until** *i*<=*pocetRiadkov* **do**

**begin**

**for** *j*←1 **step** 1 **until** *j*<i **do**

*zvysokPred* ← *zvysokPred* + *matica(i*,*j)*\**xj*(*krok*);

**for** *j*←i+1 **step** 1 **until** *j*<pocetStlpcov **do**

*zvysokPo* ← *zvysokPo* + *matica(i*,*j)*\**xj*(*krok-1*);

**comment** vyrataj samotnu hodnotu iteracnej neznamej *Xi*(krok)

*xi*(*krok*) ← *matica(pocetRiadkov*,*pocetStlpcov) – zvysokPred  
- zvysokPo;*

**if** *krok* = *total* AND *jePresnostPocitana* = *TRUE* **then**

*polePrePresnost(i)* ← *xi*(*krok*)- *xi*(*krok-1*); [[9]](#footnote-9)

**end**

*zvysokPred* ← 0;

*zvysokPo* ← 0;

*krok* ← *krok* + 1;

**end**

**comment** tlac presnost, ak ju chcel pouzivatel pocitat

**if** *jePresnostPocitana* = *TRUE* **then**  
**for** *i*←1 **step** 1**until** *i*<=*pocetRiadkov* **do**

**write** „Rozdiel medzi hodnotami xi(poslednyKrok) a xi(poslednyKrok-1) pre i =“ i „ je “ *polePrePresnost(i)*;

1. Programová realizácia
   1. Funkcionalita programu

Hlavnou úlohou programu LinearSystemsSolver je, ako už názov napovedá, riešiť sústavy *n* lineárnych rovníc o *n* neznámych. Metódu riešenia si volí používateľ sám, na základe prítomných „pluginov“ , ktorými sa dá program ľubovoľne dopĺňať[[10]](#footnote-10).

LinearSystemsSolver, v základnom vydaní, obsahuje hneď niekoľko vybraných metód, ktoré boli popísané v predchádzajúcich kapitolách.

Sú to:

* Gaussova eliminačná metóda,
* Gaussova eliminačná metóda s čiastočným výberom hlavného prvku,
* Jacobiho metóda,
* Gauss-Seidelova metóda,
* metóda Choleského.

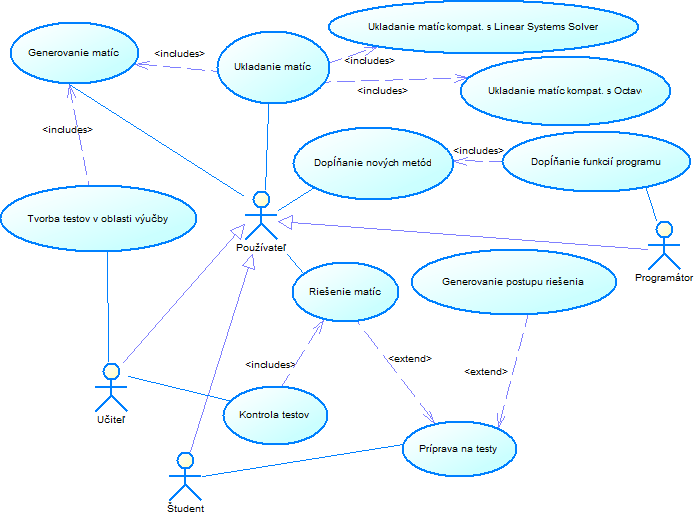
Matica vo forme dát je načítavaná zo súboru lokalizovaného dynamicky, výsledok je zobrazený na štandardný výstup.

Ďalšou požiadavkou na program bolo zakomponovať do aplikácie aj generátor matíc, ktorý by bol schopný vyprodukovať maticu o ľubovoľnom rozmere, vrátane okamžitého zobrazenia príslušného riešenia.

Generátor zároveň umožňuje aj určiť rozsah, z ktorého koeficienty  a neznáme  pochádzajú.

Úroveň programu dvíha aj možnosť generovania PDF súboru obsahujúceho popis postupu riešenia zadanej sústavy rovníc s ohľadom na zvolenú metódu. To znamená, že program nie je výhradne exaktným prostriedkom na výpočet riešenia systémov rovníc, ale zároveň aj edukačným spôsobom podáva opis postupu získania výsledku práve riešenej metódy.

Spomínané prípady použitia vrátane bližšej špecifikácie funkcií popisuje diagram prípadov použitia na obrázku nižšie.



Obr. 2 Diagram prípadov použitia aplikácie Linear Systems Solver

* 1. Výber programovacieho prístupu a jazyka, implementácia

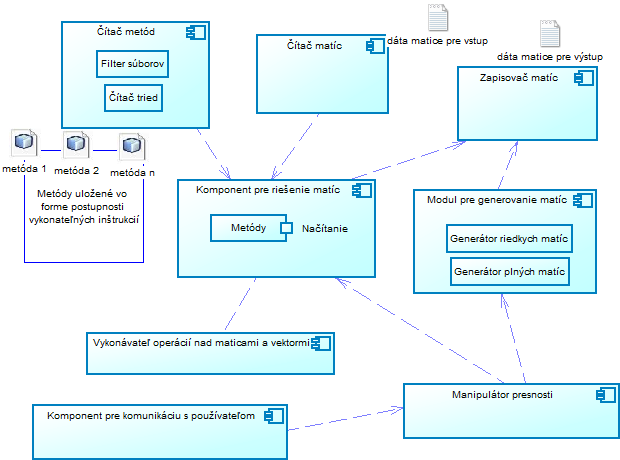
Pre implementáciu programu bol zvolený objektovo-orientovaný prístup v kombinácii s aspektovo-orientovaným prístupom. Dôvodov bolo hneď niekoľko. Použitie OOP je v týchto dňoch v praxi značne uprednostňované, čo súvisí najmä s intuitívnym chápaním sveta ako množiny objektov, ktoré spolu interagujú. AOP poslúžil na oddelenie záležitostí, dodržanie princípov modularity a na prehľadnejšiu implementáciu pre dosiahnutie sekundárnych zámerov programu. Za jazyk programovania sme vybrali Javu s rozšírením AspectJ[[11]](#footnote-11), čo spolu s technológiou FOP vytvorilo uspokojivú kombináciu na dosiahnutie vytýčených cieľov. Programovacím prostredím bolo Eclipse[[12]](#footnote-12), čo umožnilo efektívne monitorovať použitie aspektov v zdrojovom kóde.

* 1. Programové moduly

Programová realizáciu predstavuje spojenie viacerých modulov do jedného funkčného celku. Tie môžeme rozdeliť do základných dvoch skupín:

* vlastné,
* knižničné.

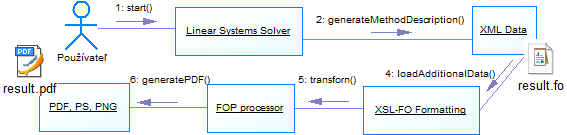
K vlastným modulom patrí *core*, *exceptions*, *examples*, *img*, *linearSystems*, *rounder* a s*wingUI*.



Obr. 3 Základné komponenty programu Linear Systems Solver (abstrakcia)

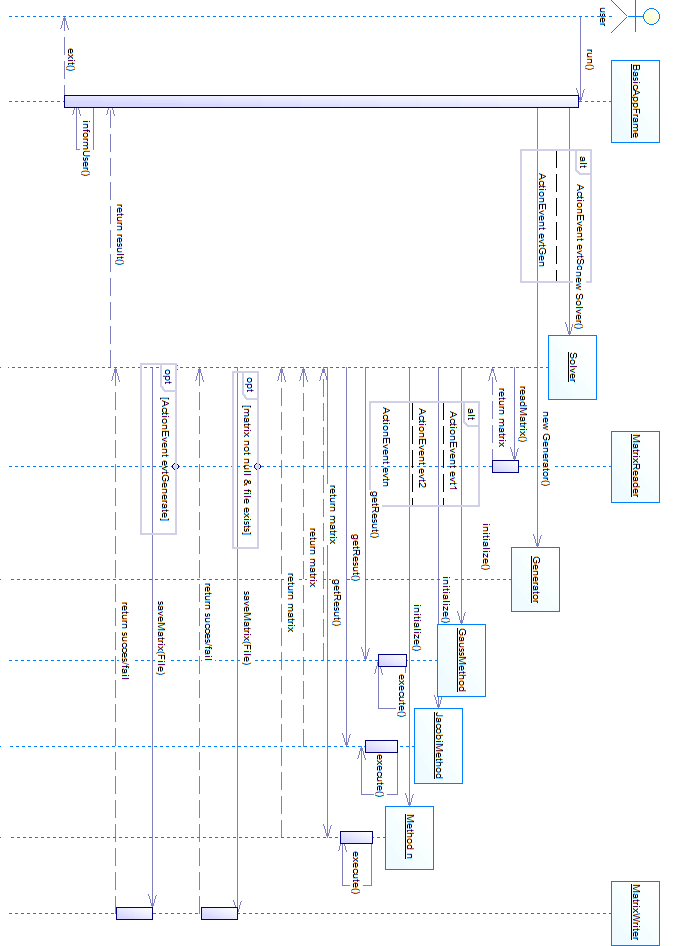
Modul *core* je samotným jadrom logiky programu, definuje procesy na základnej úrovni, komunikáciu, predávanie informácií a pod. Modul *exceptions* je súborom tzv. výnimiek, ktoré charakterizujú výnimočné stavy programu. Modul *examples* obsahuje súbory uchovávajúce matice pre demonštráciu práce jednotlivých metód, *img* je zložkou používaných obrázkov programu, *linearSystems* so svojimi ďalšími zložkami (*data* – programová reprezentácia vektorov a matíc, *dataHandlers* – logické jednotky zabezpečujúce načítavanie a zápis matíc do súborov ako aj metód dynamicky naťahovaných do programu, *exceptions* – výnimky potenciálne vzniknuté pri práci s vektormi a maticami, *interfaces* – rozhrania pre implementáciu vlastných metód a napokon *methods* spolu s *printAspects*, čo sú metódy vrátane programového vybavenia podporujúceho generovanie popisu príslušných metód do FO súborov) predstavuje matematické jadro aplikácie, *rounder* je jednoduchým podprogramom na zaokrúhľovanie čísel a *swingUI* časťou starajúcou sa o zobrazenie aplikácie vo forme okien.

Ku knižničným modulom patrí *fop* a*aspectjrt*. Modul *fop* predstavuje programové rozšírenie o schopnosť dynamického generovania textov do FO súborov a ich následnej konverzie na súbory formátu PDF a *aspectjrt* umožňuje využívať AOP pri implementácii.



Obr. 4 Diagram reprezentujúci komunikáciu medzi systémom  
Linear Systems Solver a FOP technológiou

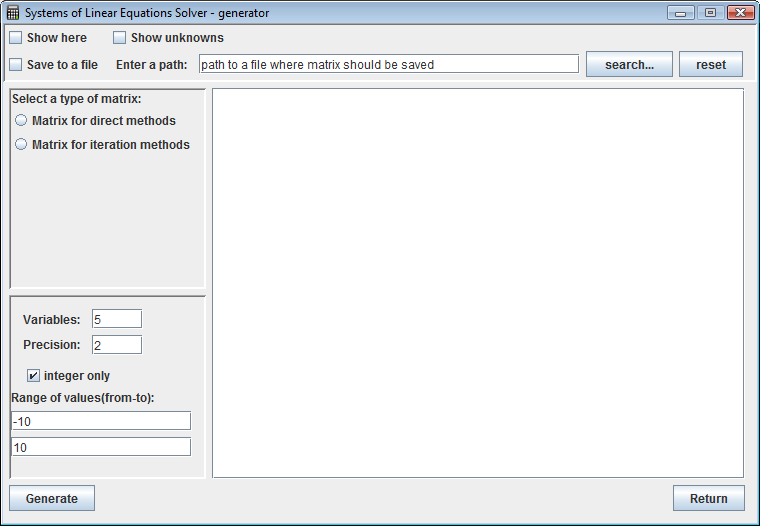
Sekvenčný diagram (Obr. 5) zobrazuje sčasti postupnosť vykonávania akcií navrhovaného programu.



Obr. 5 Sekvenčný diagram programu Linear Systems Solver

1. Experimentálne overenie činnosti programu
   1. Vstupy
      1. Vstupy pre generátor matíc

Vstupom pre generátor matíc sú požiadavky používateľa na požadovanú maticu, ktoré zadáva priamo do aplikačného okna.



*Zobraziť aj neznáme*

*Návrat*

*Spustiť generovanie*

*Použiť čísla v rozsahu od ... do ...*

*Celé alebo desatinné čísla*

*Počet použitých desatinných miest*

*Počet neznámych*

*Zmazať cestu*

*Uložiť do súboru, ku ktorému cesta je zadaná priamo alebo prostredníctvom prieskumníka súborov*

*Zobraziť v tomto okne*

Obr. 6 Aplikačné okno generátora matíc

* + 1. Vstupy časti programu, ktorá rieši matice

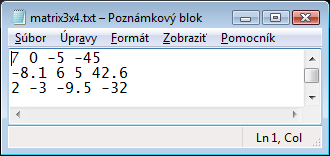
Vstupom pre modul riešiaci systémy lineárnych rovníc je systém samotný. Ten musí byť lokalizovaný v súbore, ku ktorému používateľ pristúpi pred spustením výpočtu prostredníctvom zadania cesty v programe[[13]](#footnote-13).

Štruktúra súboru je nasledovná:

* jednému riadku matice (jednej rovnici systému) zodpovedá jeden riadok súboru,
* do riadku sa zapisujú len koeficienty príslušných premenných, ak má koeficient nulovú hodnotu, nevynechávame ho, ale zapisujeme na zodpovedajúce miesto hodnotu 0,
* prvky vektora pravých strán, čiže hodnoty na pravo od znaku „=“ pri sústavách rovníc, sa zapisujú bez toho, aby im tento znak predchádzal,
* ak sú súčasťou prvého riadka napríklad 4 hodnôt, teda 4 čísla, program bude očakávať maticu rozmeru 3x4 (čiže rozšírená matica sústavy o vektor pravých strán).

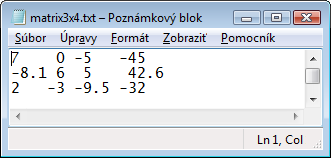
Demonštrácia spôsobu reprezentácie sústavy matematicky a v súbore (1):

 (1)



Obr. 7 Sústava (1) zapísaná v súbore podľa uvedených požiadaviek (bez zarovnávania)

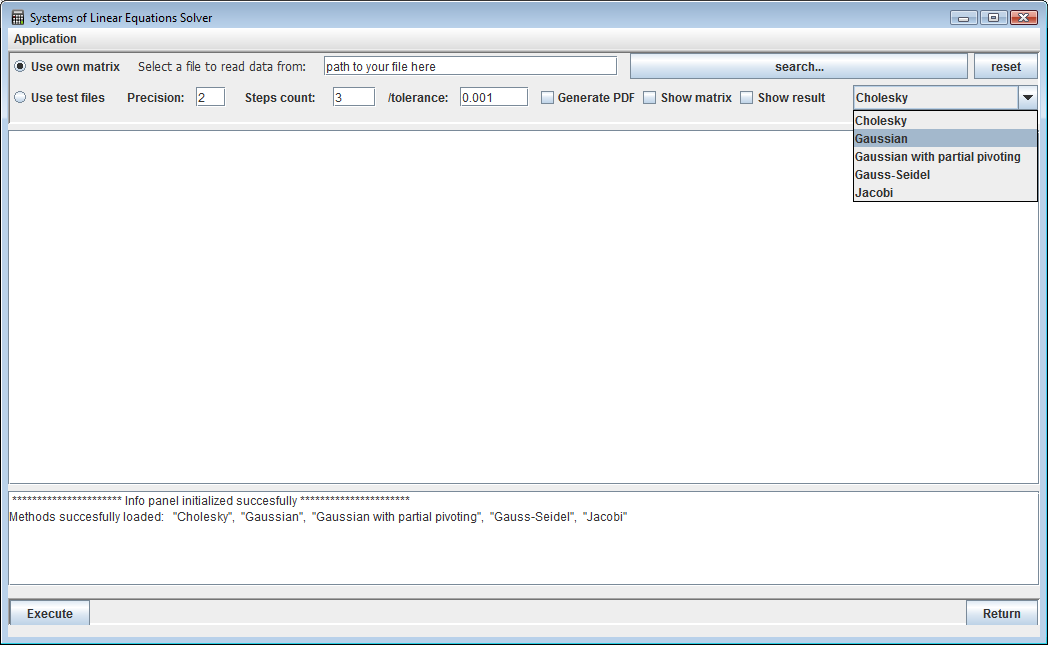
Kvôli estetickým dôvodom je možné zapísať sústavu (1) do súboru aj s dodaním viacerých bielych znakov, sémantika dát sa pri tom nemení.



Obr. 8 Sústava (1) zapísaná v súbore podľa uvedených požiadaviek (so zarovnaním)

Samotné okno vyzerá nasledovne:

*Cesta k súboru uchovávajúcemu maticu*



*Použiť vlastnú maticu alebo ukážkovú*

*Spustiť výpočet*

*Počet požadovaných krokov*

*alebo požadovaná presnosť*

*Počet desatinných miest*

*braných do úvahy*

*Návrat*

*Zobraziť výsledok?*

*Zobraziť maticu?*

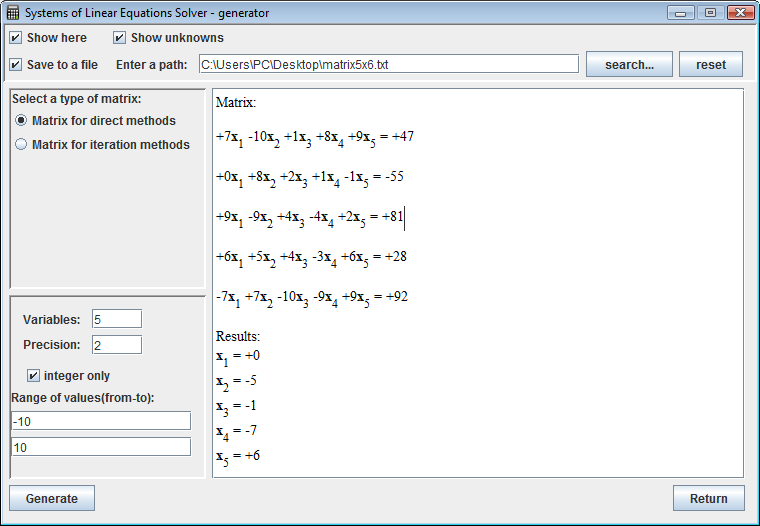
*Generovať PDF?*

*Načítané metódy*

Obr. 9 Aplikačné okno modulu pre riešenie matíc

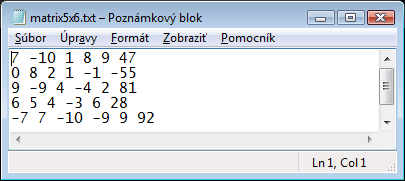
* 1. Výstupy
     1. Výstup generátora matíc

Výstupom generátora matíc podľa očakávania je matica resp. systém lineárnych rovníc vo forme textu. Ten môže byť zobrazený v aplikačnom okne.



Obr. 10 Výstup generátora matíc pre maticu 5x6 s využitím 2 desatinných miest

Poprípade, môže byť uložený do súboru definovaného súborovou cestou.



Obr. 11 Zobrazenie výstupu generátora matíc

Zobrazenie ekvivalentné obsahu okna ako na obrázku (Obr. 10) je možné získať bežným internetovým prehliadačom otvorením súboru „GeneratedMatrix.html“ v adresári LinearSystemsSolver, ktorý je automaticky vytváraný v domovskom adresári používateľa pri prvom spustení výpočtu s voľbou zobrazenia v okne aplikácie[[14]](#footnote-14).

* + 1. Výstupy časti programu, ktorá rieši matice

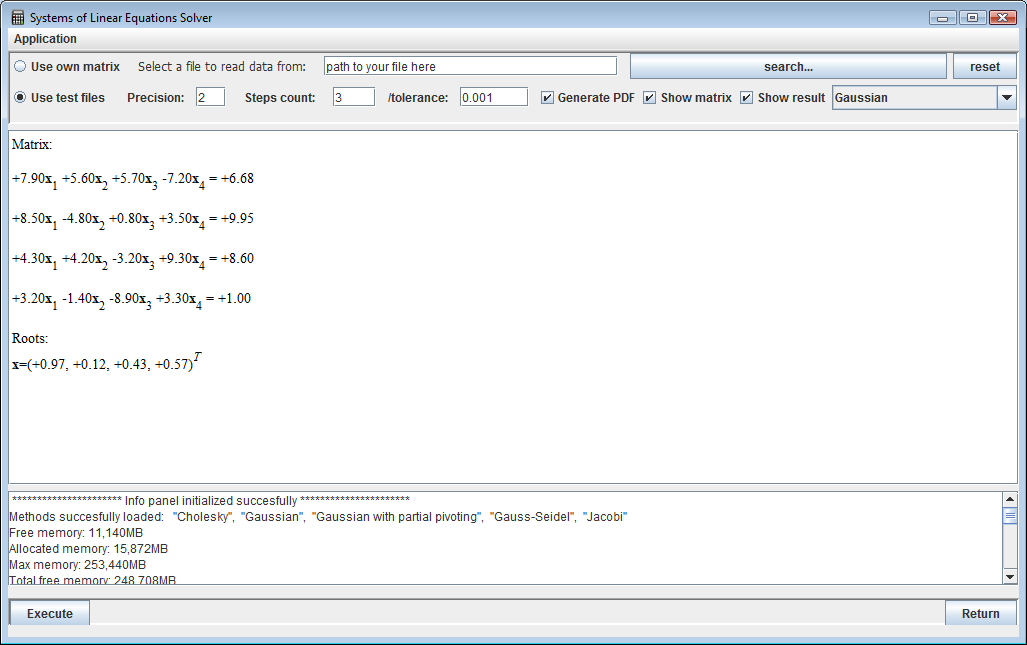
Výstupom časti programu, ktorá má na svedomí spracovanie sústavy, je samotné riešenie tejto rozšírenej matice, čo nie je nič iné, než zobrazenie nájdeného vektora riešenia. Pri realizácii výpočtu má používateľ možnosť v informačnom paneli sledovať riadiace akcie vykonávané programom.

Napríklad:

* načítané metódy,
* voľný a využitý pamäťový priestor sekundárnej pamäťovej jednotky pred a po uskutočnení výpočtu,
* použité cesty k súboru s riešenou maticou,
* monitorovanie začiatku a konca výpočtu,
* celkový čas výpočtu,
* stav generovania a konverzie FO a PDF súboru vrátanie ich lokalizácie.

Ak si používateľ zvolí možnosť generovania výstupu do PDF súboru, potom po úspešnom výpočte vrátane procesu generovania, je používateľ schopný vytvorený PDF súbor obsahujúci celý popis algoritmu lokalizovať v súborovom systéme na mieste definovanom ako

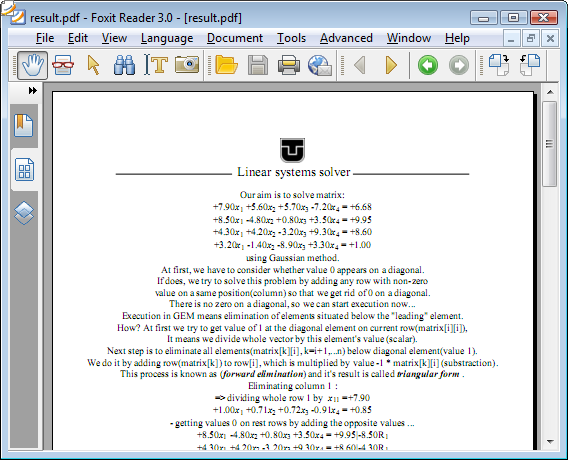
domovskýAdresárPoužívateľa\LinearSystemsSolver\result.pdf .



*Panel zobrazujúci riešenú maticu a výsledok riešenia*

*Informačný panel*

Obr. 12 Aplikačné okno programu po ukončení výpočtu



Obr. 13 Dynamicky vytváraný popis postupu výpočtu pre používateľom zadávanú maticu

* 1. Porovnanie rýchlosti programu pre konkrétne metódy

V nasledujúcej tabuľke sú uvedené priemerné časy, za ktoré sa podarilo vygenerovať matice príslušnej veľkosti a požadovanej formy[[15]](#footnote-15). Experiment bol uskutočnený opakovane, s tým, že získané údaje sa použili na výpočet aritmetického priemeru[[16]](#footnote-16).

Tab. 1 Časy trvania generovania matíc vhodných pre príslušný druh metódy

|  |  |  |
| --- | --- | --- |
| **Počet neznámych** | **Matica pre metódy** | |
| **Priame** | **Nepriame** |
| **10** | **0-1** | **3** |
| **100** | **47** | **21** |
| **1 000** | **2945** | **1346** |
| **5 000** | **162 491 (02:42,491)** | **5246** |
|  | **Čas [ms]** | |

Tab. 2 zobrazuje časy trvania riešenia matíc uvedených rozmerov. Ako aj v predchádzajúcom prípade, meranie bolo opakované viackrát, s následným výpočtom aritmetického priemeru[[17]](#footnote-17).

Tab. 2 Časy trvania výpočtu riešenia rôznych metód pre matice rôzneho typu a rôznych rozmerov

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Počet neznámych** | **Typ matice** | | | | | | |
| **Matica pre priame metódy** | | | | **Matica pre nepriame metódy** | | |
| **Gaussova eliminačná** | **Gaussova eliminačná s výberom hlavného prvku** | **Metóda Choleského** | **Octave** | **Jacobiho metóda** | **Gauss-Seidelova metóda** | **Octave** |
| **10** | **1** | **1** | **1** | **1** | **1** | **1** | **0-1** |
| **100** | **9** | **8** | **5** | **2** | **2-3** | **1-2** | **2** |
| **1 000** | **6519** | **6305** | **5186** | **335** | **72** | **45** | **343** |
|  | **Čas [ms]** | | | | | | |

* 1. Ukážkové príklady

**Príklad 1:** Ak vedenie stavby odošle 1 pracovníka zo staveniska M na stavenisko N, potom bude na oboch staveniskách rovnaký počet pracovníkov. Ak odošle 1 pracovníka zo stanoviska N na stavenisko M, potom bude na stavenisku M dvojnásobný počet pracovníkov oproti stavenisku N. Koľko pracovníkov bolo pôvodne na staveniskách M a N?

**Riešenie:** Prvá veta zadania vraví, že rovnaký počet pracovníkov bude na stavbe, ak zo staveniska M jeden pracovník odíde, a to na stavenisko N. Na stavenisku N bude teda o jedného pracovníka viac, ako bolo pôvodne. Tento vzťah zapíšeme vo forme rovnice



kde *x* je pôvodný počet pracovníkov na stavbe M a *y* počet pracovníkov na stavbe N.

Taktiež vieme, že na stavenisku M bude dvojnásobný počet pracovníkov, ak 1 pracovník zo staveniska N prejde na stavenisko M, a teda platí

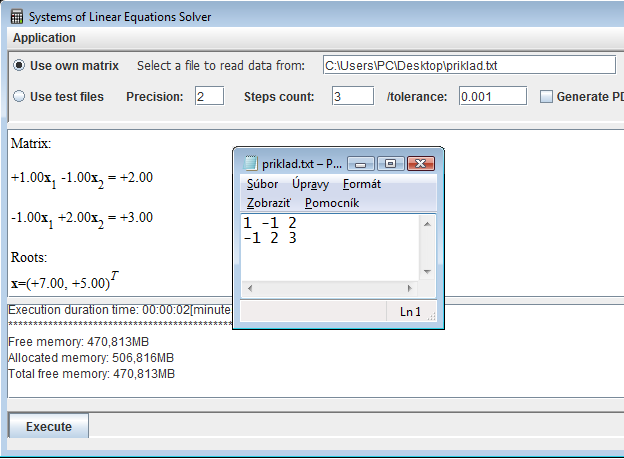


Dostávame tak sústavu 2 lineárnych rovníc o 2 neznámych



zapísanú pomocou rozšírenej matice a riešenú pomocou Linear Systems Solver takto:





Obr. 14 Demonštrácia riešenia [príkladu 1](#PRIKLAD_MATICA_1) pomocou Linear System Solver

**Odpoveď:** Na stavenisku M bolo pôvodne 7 pracovníkov a na stavenisku N 5 pracovníkov.

**Príklad 2:** Kvapalné hnojivo *A* obsahuje 25 % dusíka, 15 % fosforu a 10 % draslíka, kvapalné hnojivo *B* obsahuje 15 % dusíka, 15 % fosforu a 5 % draslíka, kvapalné hnojivo *C* obsahuje 20 % dusíka, 10 % fosforu a 10 % draslíka. Koľko litrov každého hnojiva je treba na výrobu 100** hnojiva *D*, ktoré má obsahovať 20 % dusíka, 13 % fosforu a 8,5 % draslíka?

**Riešenie:** Nech vektor  vyjadruje percentuálny obsah dusíka, fosforu a draslíka v jednotlivých hnojivách. Musí byť splnená vektorová rovnica

,

kde  sú hľadané množstvá hnojív *A*,*B*,*C* v litroch. Rozpíšeme rovnicu po zložkách a dostávame sústavu troch lineárnych rovníc o 3 neznámych :

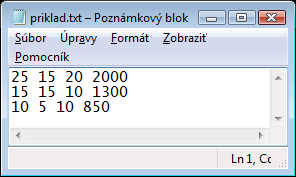


čo môžeme zapísať v tvare rozšírenej matice sústavy.



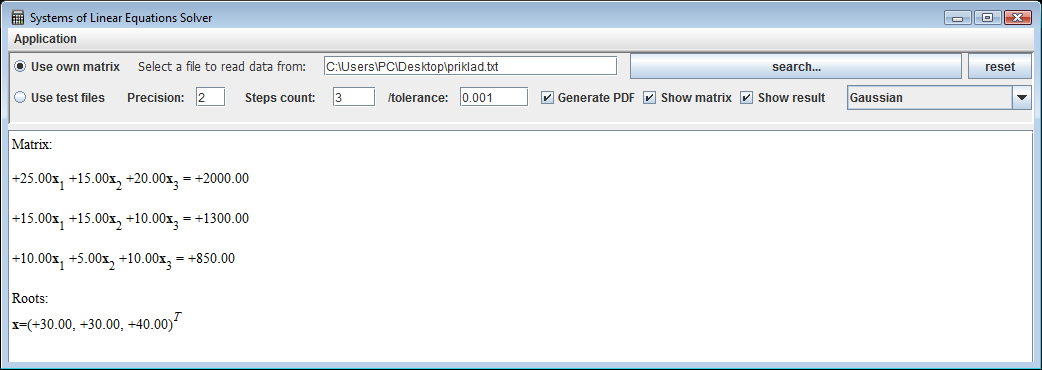
Teraz už len ostáva nájsť riešenie matice pomocou programu Linear Systems Solver.

Zapíšeme maticu do súboru:



Obr. 15 Zápis matice z [príkladu 2](#PRIKLAD_MATICA_2) do súboru

Spustíme program, lokalizujeme súbor, inicializujeme výpočet a získame riešenie.



Obr. 16 Nájdenie riešenia k príkladu pomocou programu Linear Systems Solver

**Odpoveď:** Na výrobu 100 hnojiva D potrebujeme 30 hnojiva A, 30 hnojiva B a 40 hnojiva C.

**Príklad 3:** Pán Novák investoval peniaze do dvoch bankových účtov. Prvá banka mu poskytla úrok 4,5 %, druhá banka 7 %. Na účet so 7 %-ným úrokom pán Novák vložil 2-krát taký objem peňazí ako na účet so 4,5 %-ným úrokom. Koľko peňazí vložil pán Novák na jednotlivé účty, keď na konci roka bol zisk z oboch účtov dokopy vyčíslený na sumu 1017,5 €?

**Riešenie:** Nech *x* reprezentuje množstvo peňazí investovaných na účet   
so 4,5 %-ným úrokom. Nech *y* reprezentuje množstvo peňazí investovaných na účet so 7 %-ným úrokom. Ak k tomu pridáme poznatok o stave oboch účtov na konci obdobia zúčtovania, dostávame tabuľku

Tab. Tabuľka známych informácií vyplývajúcich zo slovnej úlohy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **4,5 %-ný účet** | **7 %-ný účet** | **Zisk dokopy [€]** |
| **Základ** | ***x*** | ***y*** |  |
| **Úroková sadzba** | **0,045*x*** | **0,07*y*** | **1017,5** |

Prepísaním druhého riadku tabuľky do formy

+=

dostávame matematické vyjadrenie



Taktiež vieme, že 7 %-ný účet disponoval 2-násobným objemom peňazí oproti účtu 4,5 %-nému, a teda platí

=,

z toho vyplýva



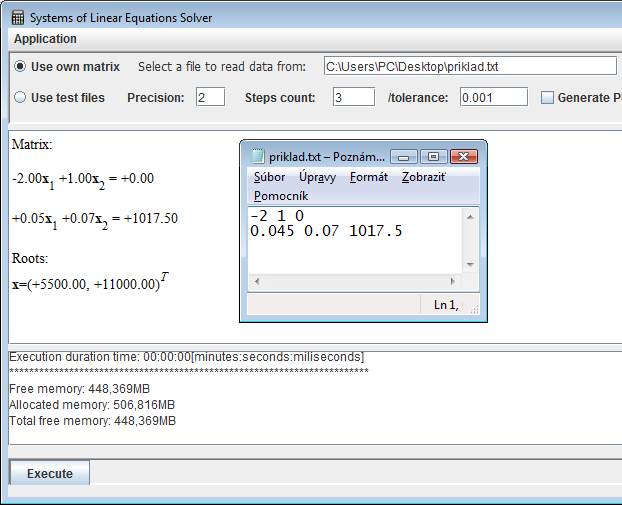
Zapísaním týchto vzťahov do sústavy rovníc vzniká systém



zapísaný vo forme matice



a vypočítaný programom tak, ako popisuje obrázok nižšie.



Obr. Riešenie príkladu získané programom Linear Systems Solver

**Odpoveď:** Na účet so 4,5 %-ným úrokom pán Novák vložil 5500 € a na účet so 7 %-ným úrokom 11000 €.

Tieto príklady sú, samozrejme, iba zjednodušené, čo sa týka pochopenia a rozmeru počítaných systémov. V praxi sa stretávame s ďaleko rozsiahlejšími systémami, čo však nemení nič na správnosti riešenia sústav tohto typu vytvoreným programom.

1. Záver

Cieľom predkladanej bakalárskej práce bolo naštudovať metódy aplikované pri riešení sústav lineárnych rovníc a možnosti ich pretavenia do softvérovej realizácie, potenciálne, s podporou popisu priebehu jednotlivých metód využiteľného v oblasti výučby na bakalárskom stupni štúdia.

Sme presvedčení, že tento cieľ sa podarilo naplniť. Program nielen dokáže riešiť matice nemalých rozmerov, ale dáva používateľovi aj možnosť pre vlastný systém lineárnych rovníc sledovať postup výpočtu spolu s príslušnými komentármi. To ho robí odlišným od väčšiny dostupných programov. V porovnaní s programom Octave sa v prípade matíc vhodných pre riešenie priamymi metódami dosiahli väčšie časy trvania výpočtov, čo však na druhej strane kompenzujú merania riešení systémov v prípade iteračných metód, kde náš program získaval výsledky riešení rýchlejšie. Podporuje formátovaný výstup systému rovníc a jeho riešenia priamo do okna aplikácie použitím HTML technológie, ktorý sa dá zobraziť aj v čase, keď aplikácia nie je aktívna. Načítavanie a samotný výstup matíc do súborov zadefinovaných používateľom ponúka akési rozhranie pre dosiahnutie kompatibility s programom Octave.

Existujú však rôzne obmedzenia programu, ktoré by mohli byť predmetom ďalšieho zdokonaľovania. Spomeňme napríklad reprezentáciu čísel s pohyblivou rádovou čiarkou v číslicových počítačoch a s tým spojené chyby výpočtu, neschopnosť programu generovať popis do PDF súboru pre matice od určitých rozmerov kvôli obmedzenému priestoru formátu A4, potenciálne rozšíriteľného na ďalšie formáty (treba sa však zamyslieť nad otázkou, či väčšie formáty majú reálne použitie), zaujímavou problematikou môže byť príprava samotných matíc pred začatím výpočtu z pohľadu zefektívnenia algoritmov, alebo otázka transformácie úplných matíc na riedke, či prepojenie vytvoreného programu pomocou open source aplikácií s cieľom rozšíriť jeho možnosti a pod.

Akokoľvek, dúfame, že výsledok našej práce nájde svoje uplatnenie v oblastiach, pre ktoré bol vytváraný, a že sa stane používanou pomôckou vyučujúcich pri výučbe (vytváranie príkladov pomocou generátora matíc) ako aj pri prípravách študentov na zápočty či skúšky (kontrola vypočítaných príkladov, overenie postupu konfrontáciou s vytvoreným PDF súborom).

Zoznam použitej literatúry

1. ORSZÁGHOVÁ, Dana – PECHOČIAK, Tomáš – TRENČIANSKA, Anna – KMEŤ, František: Matematika I. 1.vyd. Nitra : SPU, 2002. 337 s. ISBN 80-7137-985-9
2. KVASNIČKA, Vladimír – POSPÍCHAL, Jiří: Algebra a diskrétna matematika.  
    1. vyd. Bratislava: STU Press, 2008. 494 s. ISBN 978-80-227-2934-5
3. DĚMIDOVIČ, Boris Pavlovič – MARON, Isaak Abramovič: Základy numerickej matematiky. 1. vyd. Praha : SNTL, 1966. 721 s.
4. MOLNÁROVÁ, Monika – MYŠKOVÁ, Helena: Úvod do lineárnej algebry [online]. 2005 [cit. 2012-5-13], s. 33. Dostupné na internete <<http://web.tuke.sk/fei-km/old/MAT1/ULAzbierka.pdf>>. ISBN 80-8073-361-9
5. Matematika on-line [online]. [cit. 2012-5-9]. Dostupné na internete <<http://www.mathematica.sk/Algebra/Theory/07Theory.xml>>.
6. Riešené úlohy z matematiky I [online]. 2009. Aktualizované 2011-4-13 [cit. 2012-5-13]. Dostupné na internete <<http://www.math.sk/skripta>>.
7. Študijné materiály pre absolventov gymnázia [online]. Gymnázium Brno-Řečkovice. [cit. 2012-5-11]. Dostupné na internete <<http://inforec.wz.cz/Stud_mat/oktava_files/soustavy.pdf>>.
8. KRISTEK, Jozef: Numerické metódy matematiky I – Riešenie sústav lineárnych rovníc [online]. [cit. 2012-5-13]. Dostupné na internete <<http://www.earthphysics.sk/mainpage/stud_mat/menm/prednaska4.pdf>>.
9. Numerické metódy – Riešenie sústav lineárnych rovníc [online]. Aktualizované 2008-3-12 [cit. 2012-5-14]. Dostupné na internete <<http://xuv.kfe.fjfi.cvut.cz/mediawiki/index.php?title=Cvi%C4%8Den%C3%AD_Numerick%C3%A9_metody_-_%C5%98e%C5%A1en%C3%AD_soustav_line%C3%A1rn%C3%ADch_rovnic&printable=yes>>.
10. MIGLIORE, Ed: Systems of Linear Equations [online].  
     Aktualizované 2009-10-14 [cit. 2012-5-22]. Dostupné na internete: <<http://people.ucsc.edu/~miglior/chapter%20pdf/Ch03_SE.pdf>>.

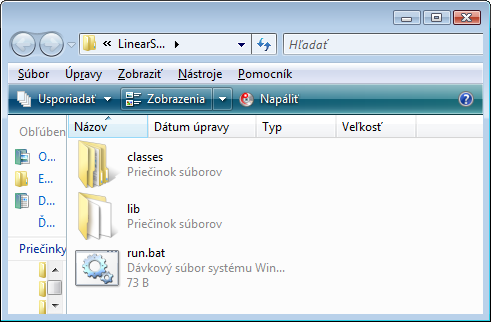
Prílohy

1. Používateľská príručka
2. Systémová príručka
3. CD médium – bakalárska práca v elektronickej podobe vrátane programu s príslušnými súbormi

Používateľská príručka

* 1. Začíname

Program môžete nájsť v priečinku LinearSystemsSolver na príslušnom pamäťovom médiu. Základnú štruktúru dodávaného adresára popisuje nasledujúci obrázok.



Obr. Obsah adresára LinearSystemsSolver

Krátky popis jednotlivých súborov:

* **classes** – obsahuje súbory so strojovo vykonateľným kódom, ako aj súbory s maticami pre ukážkové demonštrácie a obrázky používané v aplikácii vo forme ikon.
* **lib** – jeho obsahom sú knižnice potrebné pre generovanie výstupu do PDF súborov a knižnice umožňujúce použiť aspektovo-orientovaný prístup.
* **run.bat** – spustiteľný súbor v prostredí operačného systému MS Windows.
  + 1. Spustenie aplikácie

Aplikáciu je možné spustiť viacerými spôsobmi. Niektoré sú závislé od základného programového prostredia, v ktorom používateľ pracuje, iné sú univerzálne.

* Súbor „run.bat“ je spustiteľným súborom aplikácie pod OS Windows, a to buď
* dvojklikom ľavého tlačidla myši
* pomocou príkazového riadku zadaním:   
  CestaKPriecinkuLinearSystemsSolver>run.bat.
* Aplikáciu je možné spustiť taktiež pomocou príkazového riadku v prostredí OS Windows, ako aj OS Unix zadaním

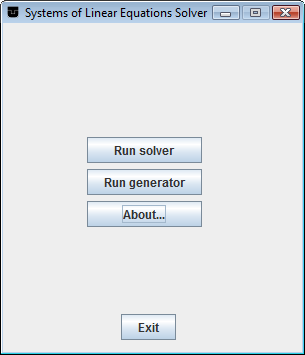
„java -classpath classes/.;lib/aspectjrt.jar;lib/fop.jar core.App“

v jednom riadku z priečinka LinearSystemsSolver, t.j.

CestaKPriecinkuLinearSystemsSolver>java -classpath classes/.;lib/aspectjrt.jar;lib/fop.jar core.App[[18]](#footnote-18).

* + 1. Hlavné menu

Ak ste boli v predchádzajúcom kroku úspešní, malo by sa vám objaviť nasledovné okno:

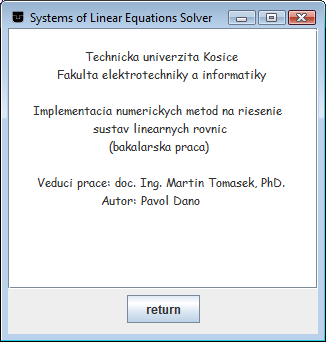


Obr. Hlavné menu programu Linear Systems Solver

Použitie jednotlivých tlačidiel je intuitívne jasné:

* **Run solver -** kliknutím spustí program na riešenie matíc,
* **Run generator -** kliknutím spustí generátor matíc,
* **About ... -** kliknutím otvorí okno s informáciami o programe,
* **Exit -** korektne ukončí bežiaci program.
  + 1. Pár informácií nezaškodí

Kliknutím na tlačidlo „About ...“ otvoríte okno s krátkymi informáciami o projekte, názve bakalárskej práce, vedúcom práce a autorovi.



Obr. Okno so stručnými informáciami o projekte Linear Systems Solver

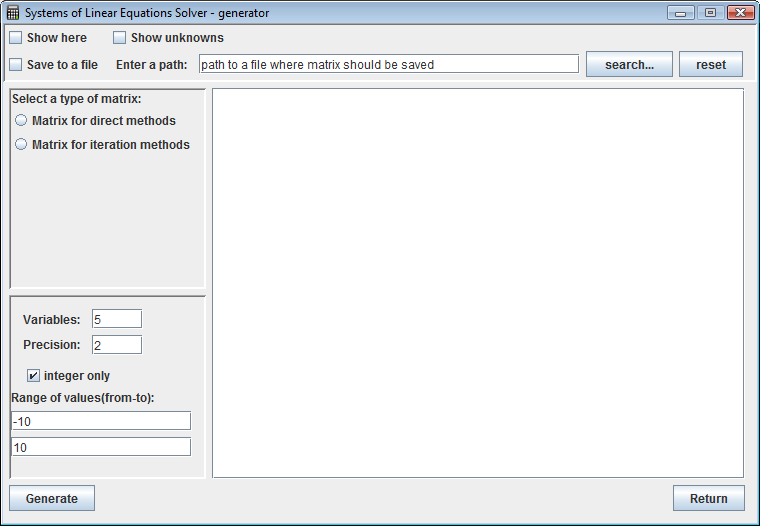
* 1. Pracujeme s maticami

V nasledujúcich podkapitolách budeme popisovať prácu s podprogramami určenými na prácu s maticami, a to

* s generátorom matíc,
* s podprogramom pre riešenie sústav lineárnych rovníc.
  + 1. Generátor matíc

Základné komponenty okna generátora matíc sú zobrazené na tomto obrázku:

*Zobraziť aj neznáme*



*Zmazať cestu*

*Zobraziť v tomto okne*

*Uložiť do súboru, ku ktorému cesta je zadaná priamo alebo prostredníctvom prieskumníka súborov*

*Spustiť generovanie*

*Použiť čísla v rozsahu od ... do ...*

*Celé alebo desatinné čísla*

*Počet použitých desatinných miest*

*Počet neznámych*

Obr. Aplikačné okno generátora matíc

#### Nastavenie vlastností matice

Generátor matíc umožňuje vygenerovať riešiteľné matice. Tieto matice majú vlastnosti, ktoré zadá používateľ v uvedenom okne. Týmito vlastnosťami sú:

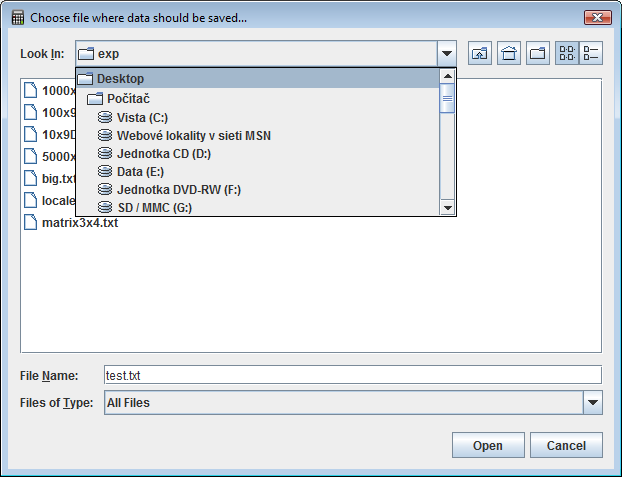
* počet premenných (*n*),
* počet uvažovaných desatinných miest,
* použitie desatinných čísel alebo len celých čísel[[19]](#footnote-19),
* rozsah použitých hodnôt pre koeficienty  nastaviteľný od dolnej hranice po hornú hranicu[[20]](#footnote-20),
* typ generovanej matice (pre priame alebo nepriame metódy).

#### Zobrazenie generovanej matice

Generovanú maticu je možné zobraziť buď v aplikácii, alebo nepriamo, v súbore, jej uložením[[21]](#footnote-21).

Generovanú maticu je možné zobraziť priamo v aplikačnom okne na pravej strane od zadávaných vlastností, a to vrátane zobrazenia hodnôt riešenia alebo bez neho. Vhodné v prípade náhodného generovania matice počas výučby, čo môžu oceniť vyučujúci, lebo študentom je v reálnom čase vytvárané zadanie (príklad), bez zobrazenia samotného riešenia[[22]](#footnote-22).

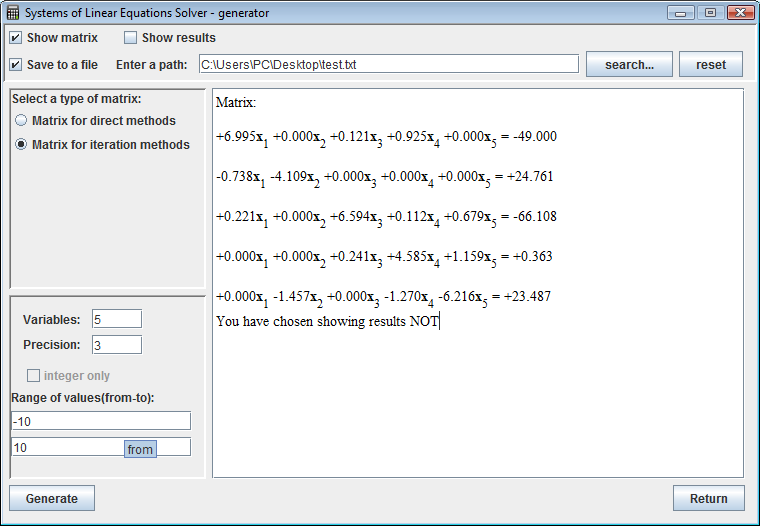
Maticu je taktiež možné uložiť do súboru na neskoršie použitie. Na tento účel slúži textové pole „Enter a path: “ alebo prehľadávač súborového systému (*file browser*) inicializovaný kliknutím tlačidla „search ...“. Názov súboru sa zadáva do poľa „File Name“ ako ukazuje obrázok. Výber sa potvrdí kliknutím na tlačidlo „Open“.



Obr. Prehľadávač súborov

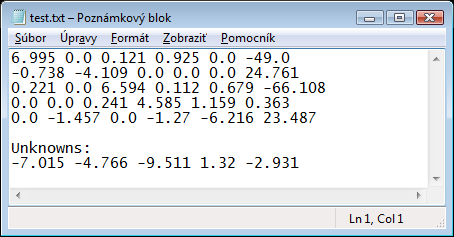
Povedzme, že chceme generovať maticu o rozmere *n* = 5, nech koeficienty majú hodnoty pohybujúce sa v rozmedzí od =-10 do 10, nech berieme do úvahy 3 desatinné miesta a nech je to matica vhodná na aplikáciu iteračných metód.

Vyprodukovaná matica nech je uložená do súboru „*test.txt*“, ktorý sa bude nachádzať na pracovnej ploche. Keďže chceme riešenie hľadať sami (alebo v prípade testovania študentov to nechať na nich ☺), hodnoty neznámych vektora **x** nezobrazíme[[23]](#footnote-23).



Obr. Ukážkový príklad pre generovanie matice

Obr. 16 popisuje výsledný súbor uložený na pracovnej ploche.



Obr. Generovaná matica uložená v súbore

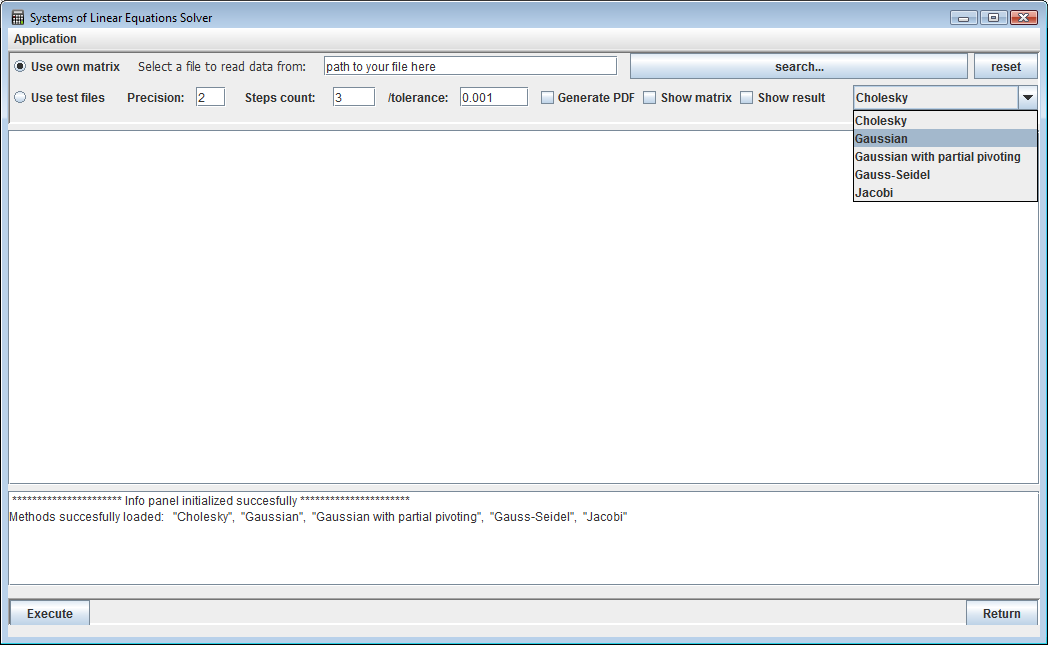
#### Návrat do hlavného menu

Späť do hlavného menu sa používateľ dostane kliknutím ľavého tlačidla myši na tlačidlo „Return“.

* + 1. Podprogram pre riešenie sústav lineárnych rovníc

Okno podprogramu pre riešenie matíc vyzerá nasledovne:

*Cesta k súboru uchovávajúcemu maticu*



*Návrat*

*Zobraziť výsledok?*

*Zobraziť maticu?*

*Generovať PDF?*

*Načítané metódy*

*Počet požadovaných krokov*

*alebo požadovaná presnosť*

*Spustiť výpočet*

*Použiť vlastnú maticu alebo ukážkovú*

*Počet desatinných miest*

*braných do úvahy*

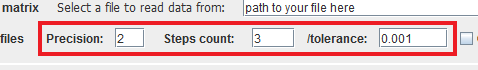
Obr. Aplikačné okno podprogramu pre riešenie matíc

#### Lokalizácia matice ako vstupných dát

Pri potrebe riešiť sústavy lineárnych rovníc programom Linear Systems Solver je v prvom rade potrebné lokalizovať požadovanú maticu. Tá, musí byť uložená v súbore tak, ako to program očakáva. Pre lokalizáciu súboru buď používateľ zadá úplnú cestu k súboru priamo, alebo na to využije vstavaný prehľadávač súborového systému, a lokalizuje súbor prechádzaním súborového systému (bližší popis v podkapitole 0). Implicitne je ponúknutá možnosť skúšobného otestovania práce programu pomocou matíc, ktoré sú uložené v súboroch dodávaných s programom. Každej metóde prislúcha práve jeden súbor s maticou (vyplýva zo zdrojového kódu), ale hlbší význam v tom hľadať netreba. Matica uložená v súbore je univerzálnym zdrojom dát pre ktorúkoľvek metódu toho istého typu. Ak sa používateľ rozhodne spustiť program len ukážkovo, s implicitne dodávanými maticami, volí tlačidlo „Use test files“, ináč, „Use own matrix“.

#### Nastavenie charakteristík výpočtu

Ďalšie možnosti programu predstavuje panel s položkami „Precision“, „Step count“ a „Tolerance“. Položkou „Precision“ zadáva počet desatinných miest, ktoré sa budú pri výpočte (zobrazení) zohľadňovať. Položky „Step count“ a taktiež aj „Tolerance“ sú použiteľné len v prípade iteračných metód, pre ktoré je charakteristický iteračný proces založený na určitom počte vykonaných krokov(„Step count“), resp. dosiahnutej precíznosti/presnosti/odchýlke („Precision“)[[24]](#footnote-24). Pri zadávaní týchto hodnôt treba byť opatrný, pretože sa môže stať, že výpočet nikdy nedosiahne zadanú presnosť (10-20, ale záleží od kovergencie metódy, reprezentácii čísla v počítači a pod.), a teda neskončí.



Obr. Položky pre nastavenie charakteristík výpočtu

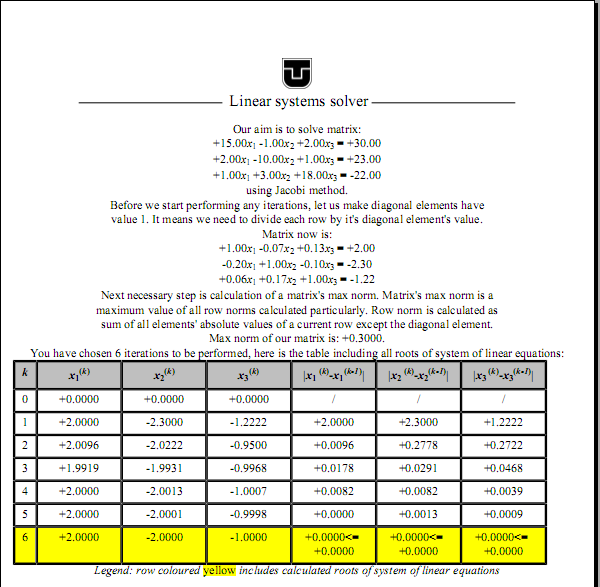
#### Nastavenie zobrazenia výsledku

Výsledok spracovania matice môže byť zobrazený rôznymi spôsobmi. Ak si používateľ nepraje zobraziť v okne aplikácie riešenú maticu, ponechá políčko „Show matrix“ nezaškrtnuté. Ak si používateľ nepraje zobraziť v okne aplikácie ani výsledok, teda hľadané riešenie, ponechá políčko „Show result“ nezaškrtnuté. Osobitnou možnosťou je zobraziť realizovaný výpočet vrátane slovného popisu vytvorením súboru formátu PDF[[25]](#footnote-25). Pre splnenie tejto požiadavky používateľ zaškrtne políčko „Generate PDF“.



Obr. Nezaškrtnuté položky pre rôzne zobrazenia výsledku výpočtu

Generovaný PDF súbor je ukladaný do používateľského domovského adresára do priečinka LinearSystemsSolver, ktorý je vytváraný počas behu programu[[26]](#footnote-26). V niektorých prípadoch je potrebné uvážiť voľbu tejto možnosti. Dôvodom môže byť neschopnosť generovať celú maticu do priestorovo obmedzeného formátu A4, poprípade iných časti algoritmu (napríklad tabuľka pre zobrazenie priebežných hodnôt aproximácií).



Obr. Ukážka PDF výstupu pre Jacobiho metódu

#### Návrat do hlavného menu

Späť do hlavného menu sa používateľ dostane kliknutím ľavého tlačidla myši na tlačidlo „Return“.

* 1. Končíme

Z hlavného menu sa aplikácia korektne ukončuje stlačením tlačidla „Exit“ ľavým tlačidlom myši.



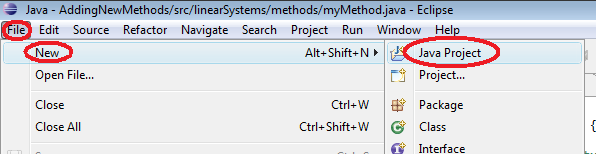
Obr. Korektné ukončenie aplikácie

* 1. Programujeme vlastné metódy

V tejto časti používateľskej príručky si priblížime postup dotvárania vlastných metód. Využijeme pri tom známe vývojové prostredie Eclipse[[27]](#footnote-27).

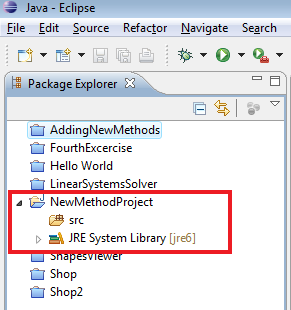
* + 1. Vytvorenie projektu

Projekt vytvoríme pomocou klasického postupu. Otvoríme prostredie Eclipse a zvolíme File -> New -> Project a z ponuky vyberieme Java project[[28]](#footnote-28).



Obr. Vytvorenie nového Java projektu

Následne zvolíme meno (napr. „NewMethodProject“) a miesto uloženia. Package Explorer by nám mal ponúknuť takýto pohľad:



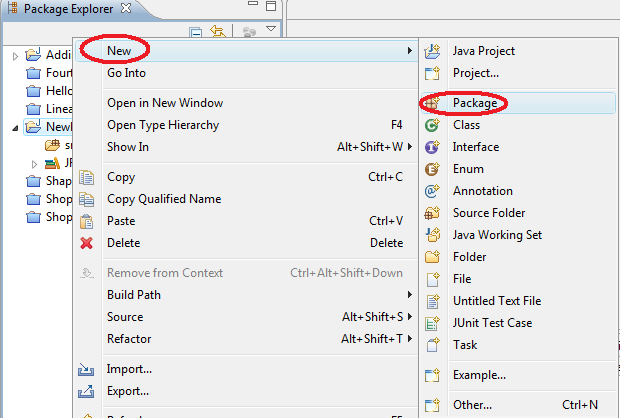
Obr. Vytvorený projekt v Package Exploreri

Predtým, než zahájime samotnú implementáciu, musia byť splnené ďalšie podmienky a vykonané ďalšie kroky na dosiahnutie kompatibility s frameworkom Linear Systems Solver.

* + 1. Príprava

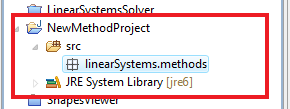
V štádiu prípravy vytvoríme triedu reprezentujúcu našu vlastnú matematickú metódu. Tá musí byť umiestnená v balíku „methods“, ktorý je umiestnený v otcovskom balíku „linearSystems“.

Vložme teda do priečinka „src“ novovytvorený balík, ktorý priamo pomenujeme ako „linearSystems.methods“[[29]](#footnote-29).



Obr. Vloženie nového balíka linearSystems.methods do adresára src

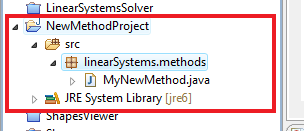
Náš projekt teda vyzerá nasledovne:



Obr. Náš projekt po vytvorení nového balíka linearSystems.methods

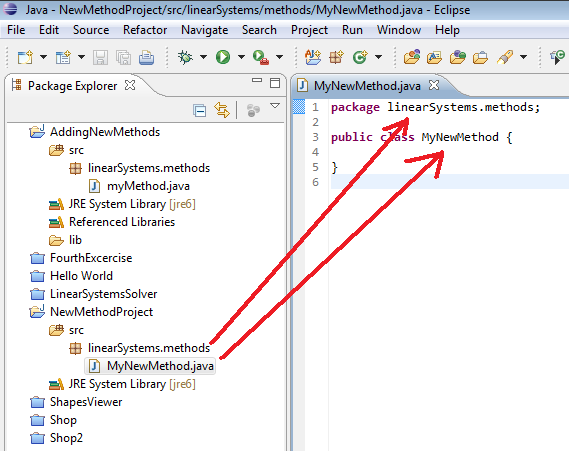
Teraz už môžeme vytvoriť triedu reprezentujúcu našu novú metódu. Postup je podobný ako pri vytvorení balíka, ale namiesto balíka (Package) volíme triedu (Class). Umiestnime ju do v predchádzajúcom kroku vytvoreného balíka „linearSystems.methods“ a pomenujeme podľa potreby (napríklad „MyNewMethod“).

Mal by sa nám naskytnúť takýto pohľad:



Obr. Náš projekt po vytvorení triedy reprezentujúcej novú metódu

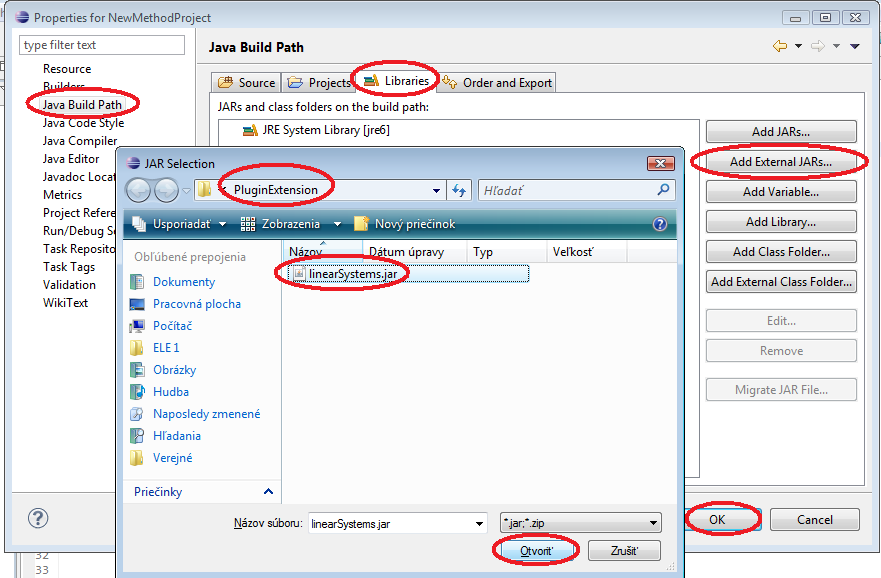
Obsah našej triedy by po otvorení mal vyzerať nasledovne:



Obr. Obsah triedy MyNewMethod

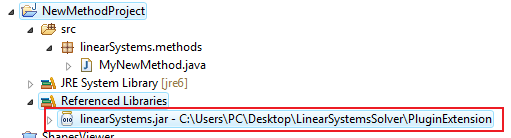
Podmienkou, ktorú treba splniť, aby bola dodržaná kompatibilita medzi vytváranou triedou (metódou) a Linear Systems Solver frameworkom, je nutnosť implementovania rozhrania „LinearSystemsSolvable“ triedou „MyNewMethod“.   
Jeho umiestnenie je: linearSystems.interfaces.LinearSystemsSolvable. Knižnica, kde sa rozhranie nachádza, nesie názov „linearSystems.jar“ a je dodávaná spolu s programom. Môžete ju nájsť medzi zdrojovými kódmi v priečinku „PluginExtension“.

Vložíme knižnicu „linearSystems.jar“ do nášho projektu klinutím pravého tlačidla myši na ikone projektu v Package Exploreri, zvolíme poslednú možnosť („Properties“), na ľavej strane pod položkou „Java Build Path“ v záložke „Libraries“ cez tlačidlo „Add external JARs...“ lokalizujeme spomínanú knižnicu v súborovom systéme a výber potvrdíme.



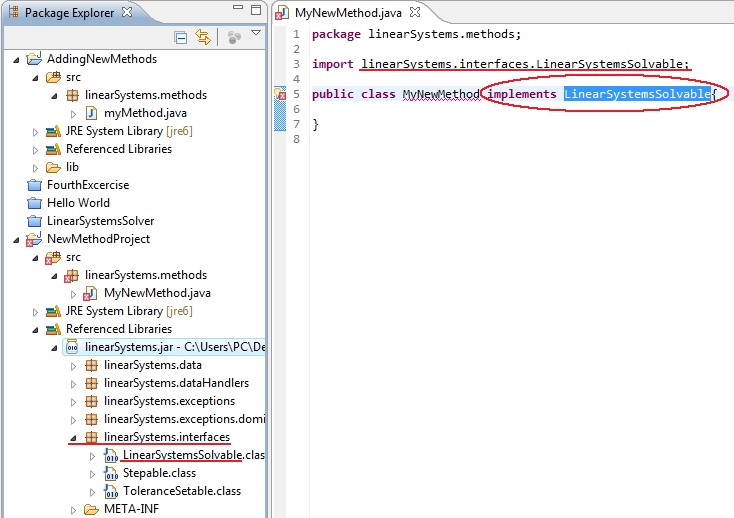
Obr. Vloženie knižnice linearSystems.jar

V Package Exploreri sa to prejaví pridaním ikony .jar súboru do položky „Referenced Libraries“.



Obr. Náhľad na projekt v Eclipse po pridaní knižnice linearSystems.jar

Teraz už môže vytváraná metóda implementovať rozhranie „LinearSystemsSolvable“.

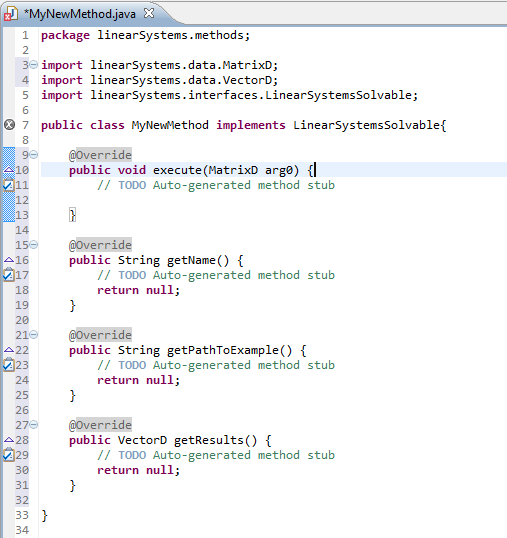


Zvýraznené tlačidlo

Obr. Rozšírenie triedy MyNewMethod o implementovanie rozhrania LinearSystemsSolvable

Kliknutím na zvýraznené tlačidlo Eclipse automaticky pridá do zdrojového kódu metódy, ktoré je nutné implementovať pre správnu prácu programu.   
Implicitne sa importovali aj ďalšie triedy, ktoré sú potrebné pri odovzdávaní informácií. Pre bližšie informácie je potrebné nahliadnuť do Java API dokumentácie uvedenej v prílohách.

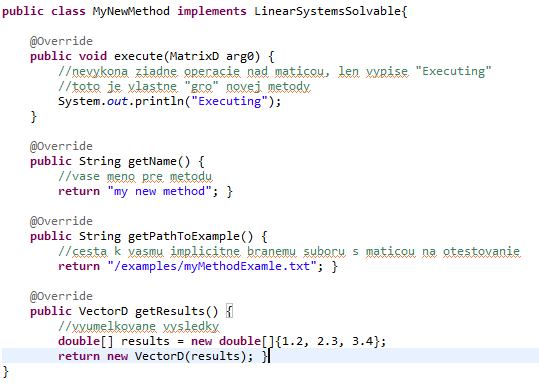
Obr. 31 demonštruje vygenerovaný kód, potrebný na implementáciu.



Obr. Automatické vloženie metód potrebných na správnu funkciu rozširovaného programu

* + 1. Implementácia

V tomto kroku nasleduje samotná implementácia automaticky vložených metód objektu (matematickej metódy) závislá na používateľovi - programátorovi. Jednoduchá ukážka je zobrazená na obrázku uvedenom nižšie.

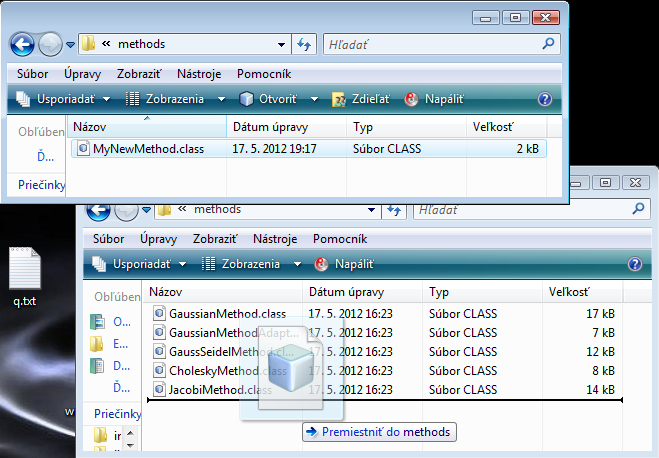


Obr. Demonštračný príklad implementácie

* + 1. Zavedenie vytvorenej metódy do systému Linear Systems Solver

Posledným nevyhnutným krokom pre zavŕšenie procesu dodania novej metódy do programu je umiestnenie vytvoreného[[30]](#footnote-30) „.class“ súboru do adekvátneho adresára, z ktorého sú metódy načítavané.

V súborovom systéme nájdeme umiestnenie nami naprogramovanej metódy „MyNewMethod.class“. Túto metódu premiestnime na miesto uloženia programu Linear Systems Solver. Akciu popisuje nasledovný obrázok:

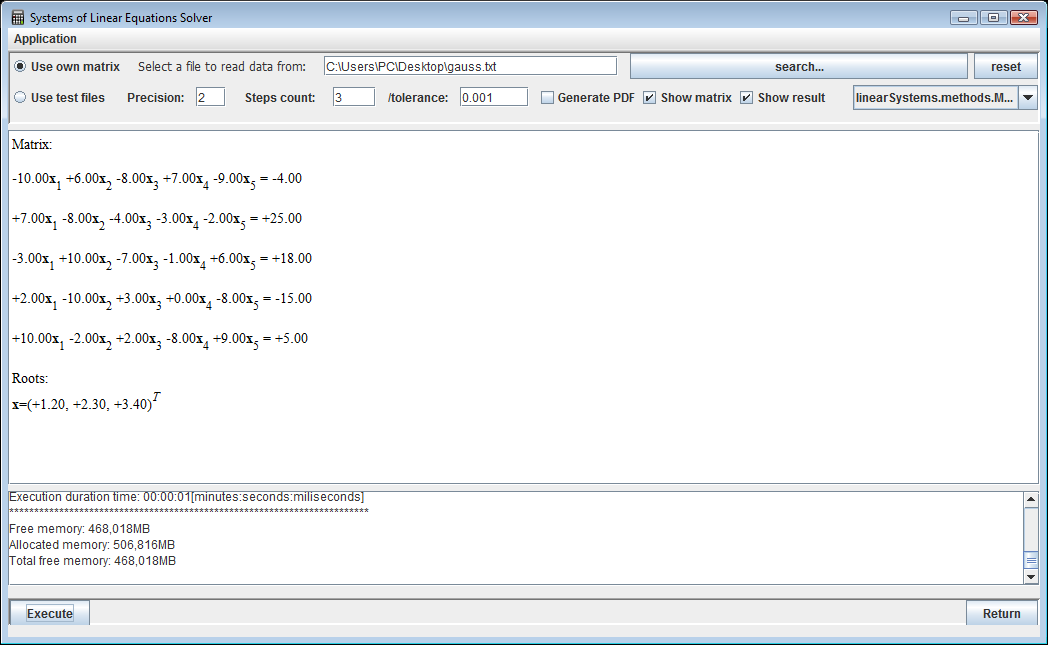


Adresár classes/linearSystems/methods programu Linear Systems Solver

Adresár umiestnenia naprogramovaných metód (projekty Eclipse)

Obr. Premiestnenie vytvorenej metódy do adekvátneho adresára  
 programu Linear Systems Solver

Teraz už len ostáva spustiť aplikáciu a otestovať, či sme postupovali správne.



Riešenie

Meno našej metódy

Obr. Aplikácie vytvorenej metódy v bežiacom programe

Dôkazom úspešného naprogramovania vlastnej metódy je jej názov (reprezentácia návratovej hodnoty toString() pre metódu[[31]](#footnote-31)) uvedený v ComboBoxe a výsledok, ktorý samozrejme nie je správny z matematického hľadiska, ale je správny z funkčného hľadiska, pretože keď nahliadneme do implementácie, zistíme, že vrátený výsledok je stále ten istý, a to vektor [1,2 ; 2,3 ; 3,4] [[32]](#footnote-32).

Systémová príručka

* 1. Základné informácie

V tejto časti sa dočítate o základných podmienkach, ktoré treba splniť, aby ste správne nakonfigurovali prostredie pre ďalší vývoj aplikácie Linear Systems Solver.

* + 1. Výber programovacieho jazyka

Aplikácia Linear Systems Solver je naprogramovaná v implementačnom jazyku Java[[33]](#footnote-33).

* + 1. Výber vývojového prostredia

Aplikácia Linear Systems Solver je vyvíjaná vo vývojovom prostredí Eclipse[[34]](#footnote-34) (verzia Indigo). Dôvodom je použitie princípov Aspektovo-orientovaného prístupu, v našom prípade prekladača AspectJ, ktorý je najlepšie integrovaný práve vo vývojom prostredí Eclipse.

* + 1. Príprava prostredia

Pred začatím samotnej implementácie nových funkcií, je potrebné pridať dosiaľ vytvorené zdrojové kódy, ktoré nájdete v adresári *src* a *lib* na pribalenom pamäťovom médiu v časti pre prílohy.

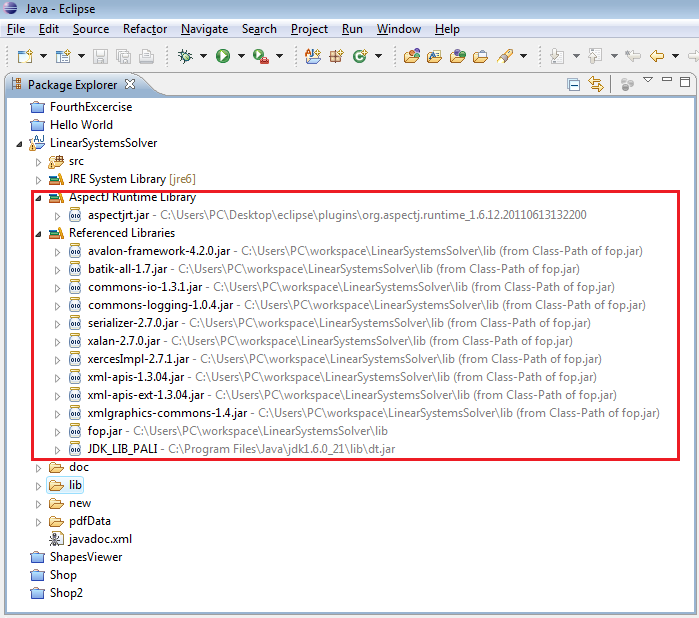
* + - 1. Vlastné zdrojové kódy

Vlastné zdrojové kódy sú umiestnené v adresári *src* rodičovského adresára LinearSystemsSolver.

* + - 1. Knižničné moduly

Ku korektnej práci programu je potrebné pridať aj knižničné moduly podporujúce dodatočné (neprimárne) funkcie programu. Pre použitie aspektovo-orientovaného prístupu nezabudnite k projektu pridať knižnicu *aspectjrt.jar* (import alebo pridaním do adresára *lib*). Pre podporu konverzie FO súborov na PDF je nevyhnutné pridať knižnicu *fop.jar* vrátane podporných. Všetky sú umiestnené v adresári *lib* priečinka LinearSystemsSolver.

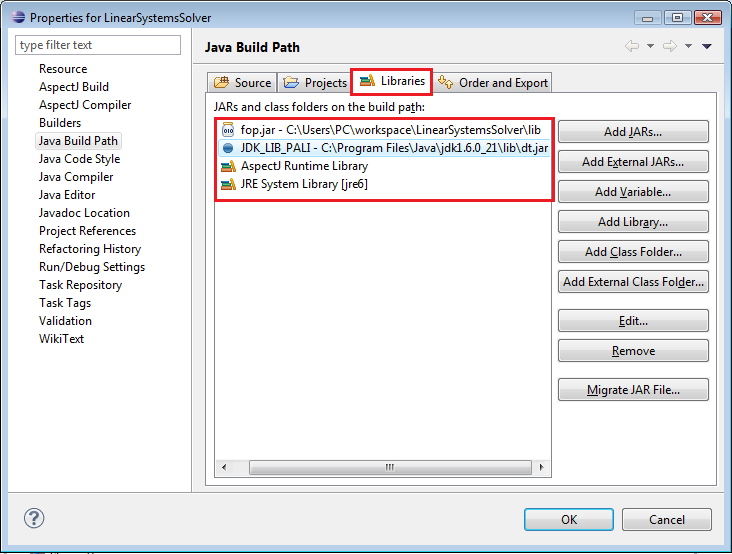
Package Explorer by vám mal poskytnúť takýto náhľad:



Obr. Náhľad na pripravený projekt v záložke Package Explorer

* + - 1. Classpath

Skontrolujte záložku „Libraries“ v možnostiach (pravým tlačidlom na meno projektu v Package Exploreri) v časti Java Build Path. Ak chcete generovať Javadoc neskôr, mali by ste lokalizovať aj dodatočný *.jar* súbor JDK. Nastavenie popisuje nasledovný obrázok:



Obr. Nastavenie classpath (Java build path) priamo v Eclipse

Pre iné prípady sa *classpath* nastavuje tradične pod OS Windows v rámci systémovej premennej prostredia (C:\aspectj1.6\bin; C:\FOP\fop-1.0;).

* + - 1. Kompilácia

Kompiláciu, ak ste postupovali správne, vykonáva vývojové prostredie Eclipse po každom uložení zdrojového kódu (*Ctrl+S*). Pre kompiláciu z príkazového riadku zadajte príkaz vo forme

cestaKProjektu>ajc -*verzia* -sourceroots "src\core;src\exceptions;src\linearS

ystems\data;src\linearSystems\dataHandlers;src\linearSystems\exceptions;src\line

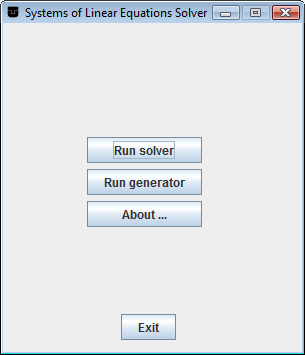
arSystems\interfaces;src\linearSystems\methods;src\linearSystems\printAspects;sr

c\rounder;src\swingUI" -cp src.;lib\aspectjrt.jar;lib\fop.jar -d bin

,kde *verzia* je verziou aspectJ kompilátora (voliteľné), -sourceroots zahŕňa všetky zdrojové súbory, a prepínač –cp definuje classpath, teda cestu k potrebným knižniciam používaných v programe (tu aspectjrt.jar a fop.jar), -d vytvára nový priečinok, kde umiestňuje preložené .class súbory.

* + 1. Spúšťanie

Spúšťanie aplikácie sa vykonáva, taktiež tradične, stlačením zeleného tlačidla v hlavnom paneli alebo v záložke Run. Pre spúšťanie pomocou príkazového riadku (alternatívne cez *.bat* súbor v OS Windows) z adresára vášho projektu je kľúčový príkaz v takomto tvare:java -Xms512m -classpath bin/.;lib/aspectjrt.jar;lib/fop.jar core.App

Prepínač -Xms512m je voliteľný, no nevyhnutný na alokáciu dodatočného pamäťového miesta v prípade manipulácie s maticami väčších rozmerov. Samozrejme, voľbu veľkosti (v tomto prípade 512 MB) voľte tak, ako uznáte za vhodné. Ak ste postupovali správne, mal by sa vám naskytnúť takýto pohľad:

Obr. Hlavné menu aplikácie   
Linear Systems Solver

* 1. Dokumentácia

V tejto časti sú v skratke popísané triedy programového systému Linear Systems Solver. Názvy kapitol predstavujú úplné mena tried a sú zoradené abecedne vo vzostupnom poradí.

* + 1. core.App

Public Member Functions

* App ()
* void execute (String path, LinearSystemsSolvable method) throws FileNotFoundException, IOException, WrongDoubleFormatException, BaseDirectoryCreationException

Static Public Member Functions

* static void main (String[] args)

Static Public Attributes

* static final String baseDataDirPath = "LinearSystemsSolver"
* static File baseDataDirFile = null
* static MatrixReader matrixReader = MatrixReader.getInstance()
* static DynamicPDFGenerator pdfGenerator
* static DynamicHTMLGenerator htmlGenerator
* static MethodsClassLoader classLoader

Static Private Member Functions

* static File createBaseDirectoryFile () throws BaseDirectoryCreationException

Detailed Description

Funkcia: jadro aplikácie, obsahuje main().

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 36 súboru App.java.

Constructor & Destructor Documentation

#### core.App.App ()

konštruktor

Definícia na riadku 83 súboru App.java.

Member Function Documentation

#### static File core.App.createBaseDirectoryFile () throws BaseDirectoryCreationException [static, private]

vytvára základný udájový adresár pre ukladanie .html, .fo, .pdf súborov v používateľskom domovskom adresári

##### Returns:

file reprezentácia súboru k základnému dátovému adresáru aplikácie

##### Exceptions:

|  |  |
| --- | --- |
| *BaseDirectoryCreationException* | výnimka hádzaná v prípade neúspešného vytvorenia dátového adresára |

Definícia na riadku 116 súboru App.java.

#### void core.App.execute (String *path*, LinearSystemsSolvable *method*) throws FileNotFoundException, IOException, WrongDoubleFormatException, BaseDirectoryCreationException

vykonáva celý algoritmus

##### Parameters:

|  |  |
| --- | --- |
| *path* | cesta k zdrojovému súboru s maticou |
| *method* | metóda, ktorá sa má vykonať |

##### Exceptions:

|  |  |
| --- | --- |
| *FileNotFoundException* | výnimka hádzaná v prípade nenájdenia súboru |
| *IOException* | výnimka hádzaná v prípade I/O chyby |
| *WrongDoubleFormatException* | výnimka hádzaná v prípade nesprávneho formátu čísla v súbore definovanom premennou path |
| *BaseDirectoryCreationException* | výnimka hádzaná v prípade neúspešého vytvorenia základného dátového adresára |

Definícia na riadku 97 súboru App.java.

#### static void core.App.main (String[] *args*) [static]

inicializácia aplikácie

##### Parameters:

|  |  |
| --- | --- |
| *args* | nepoužitý argument |

Definícia na riadku 136 súboru App.java.

Member Data Documentation

#### File core.App.baseDataDirFile = null [static]

objekt súboru reprezentujúci základný dátový adresár aplikácie

Definícia na riadku 48 súboru App.java.

#### final String core.App.baseDataDirPath = "LinearSystemsSolver" [static]

cesta k základnému dátovému adresáru aplikácie

Definícia na riadku 41 súboru App.java.

#### MethodsClassLoader core.App.classLoader [static]

Initial value: MethodsClassLoader

.getInstance()

inštancia objektu MethodsClassLoader

Definícia na riadku 70 súboru App.java.

#### DynamicHTMLGenerator core.App.htmlGenerator [static]

Initial value: DynamicHTMLGenerator

.getInstance()

inštancia objektu DynamicHTMLGenerator

Definícia na riadku 64 súboru App.java.

#### MatrixReader core.App.matrixReader = MatrixReader.getInstance() [static]

inštancia objektu MatrixReader

Definícia na riadku 53 súboru App.java.

#### DynamicPDFGenerator core.App.pdfGenerator [static]

Initial value: DynamicPDFGenerator

.getInstance()

inštancia objektu DynamicPDFGenerator

Definícia na riadku 58 súboru App.java.

* + 1. core.basicAspects.Informer

Private Attributes

* WeakReference
* < javax.swing.JTextArea > infoPanel

Static Private Attributes

* static Informer instance

Detailed Description

Funkcia: informuje používateľa o akciách vykonaných alebo práve vykonávaných.

##### Author:

Pavol Dano

Definícia na riadku 24 súboru Informer.java.

Member Data Documentation

#### WeakReference<javax.swing.JTextArea> core.basicAspects.Informer.infoPanel [private]

slabá referencia na infopanel basicSolverFrame-u

Definícia na riadku 34 súboru Informer.java.

#### Informer core.basicAspects.Informer.instance [static, private]

inštancia triedy Informer

Definícia na riadku 29 súboru Informer.java.

* + 1. core.DynamicHTMLGenerator

Static Private Attributes

* static String pathToHtml
* static StringBuilder text = new StringBuilder()
* static String ln = System.getProperty("line.separator")
* static DynamicHTMLGenerator instance

Detailed Description

Funkcia: generuje HTML súbor pre zobrazenie matice po vykonaní výpočtu.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 37 súboru DynamicHTMLGenerator.java.

Member Data Documentation

#### DynamicHTMLGenerator core.DynamicHTMLGenerator.instance [static, private]

inštancia triedy

Definícia na riadku 57 súboru DynamicHTMLGenerator.java.

#### String core.DynamicHTMLGenerator.ln = System.getProperty("line.separator") [static, private]

oddeľovač riadkov

Definícia na riadku 52 súboru DynamicHTMLGenerator.java.

#### String core.DynamicHTMLGenerator.pathToHtml [static, private]

cesta k .html súboru

Definícia na riadku 42 súboru DynamicHTMLGenerator.java.

#### StringBuilder core.DynamicHTMLGenerator.text = new StringBuilder() [static, private]

StringBuilder objekt pre udržiavanie a manipuláciu s obsahu .html súboru

Definícia na riadku 47 súboru DynamicHTMLGenerator.java.

* + 1. core.DynamicPDFGenerator

Public Member Functions

* void addMatrixAsTextBoldElement (MatrixD matrixD, int row, int column)

Static Public Member Functions

* static DynamicPDFGenerator getInstance ()
* static String getText ()
* static void addText (String str)
* static void addTextBrLn (String str)
* static void addTextlnBrKeepWithNext (String str)
* static void addlnBr ()
* static void addlnBrBr ()
* static void addBr ()
* static void addLn ()
* static void addTextln (String str)
* static void addHead ()
* static void addTail ()
* static void addMatrixElement (double element)
* static void addVectorAsText (VectorD vectorD)
* static void addVectorAsTextRSubstraction (VectorD vectorD, int dividerRow)
* static void addMatrixAsText (MatrixD matrixD)
* static void addMatrixAsTextTwoBoldElementsOnAColumn (MatrixD matrixD, int column, int row1, int row2)
* static void addMatrixAsTextDiagonalElementsExpressedForm (MatrixD matrixD)
* static void addMatrixAsTextBoldUnderDiagonal (MatrixD matrixD)
* static void addTextInitialApproxValuesEqZero (MatrixD matrixD)
* static void addIterationElement (String variable, String subIndex, String supIndex)
* static void addPicture (File pictFile, String align, String height, String width)
* static void printAlgorithmIntoAFile (File file) throws FileNotFoundException, IOException

Private Member Functions

* DynamicPDFGenerator ()

Static Private Member Functions

* static void addLogoToHead ()

Static Private Attributes

* static StringBuilder text = new StringBuilder()
* static String ln = System.getProperty("line.separator")
* static String pageHeight = String.valueOf(297)
* static String pageWidth = String.valueOf(210)
* static String headerHeight = String.valueOf(20)
* static String footerHeight = String.valueOf(20)
* static DynamicPDFGenerator instance

Detailed Description

Funkcia: Trieda pre popis aktuálnej matematickej metódy pre generovanie XSL-FO dokumentu pomocou konverzie do PDF.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 37 súboru DynamicPDFGenerator.java.

Constructor & Destructor Documentation

#### core.DynamicPDFGenerator.DynamicPDFGenerator () [private]

konštruktor

Definícia na riadku 94 súboru DynamicPDFGenerator.java.

Member Function Documentation

#### static void core.DynamicPDFGenerator.addBr () [static]

pridanie FO nového riadku vo forme reťazca ako oddelného bloku

Definícia na riadku 156 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addHead () [static]

pridanie hlavičky FO súboru

Definícia na riadku 178 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addIterationElement (String *variable*, String *subIndex*, String *supIndex*) [static]

pridanie prvkov iterácie s horným a dolným indexom

##### Parameters:

|  |  |
| --- | --- |
| *variable* | Pridávaná premenná |
| *subIndex* | Dolný index |
| *supIndex* | Horný index |

Definícia na riadku 457 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addLn () [static]

pridanie nových riadkov ako ´\n´ znakov

Definícia na riadku 163 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addlnBr () [static]

priadnie znaku nového riadka a FO formy nového riadka

Definícia na riadku 138 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addlnBrBr () [static]

priadnie znaku nového riadka a FO formy nového riadka dvakrát

Definícia na riadku 147 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addLogoToHead () [static, private]

pridanie loga tuke, ak existuje

Definícia na riadku 211 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addMatrixAsText (MatrixD *matrixD*) [static]

pridanie matrix objektu vo forme textu – základná forma

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | pridávaná matica |

Definícia na riadku 284 súboru DynamicPDFGenerator.java.

#### void core.DynamicPDFGenerator.addMatrixAsTextBoldElement (MatrixD *matrixD*, int *row*, int *column*)

pridanie matrix objektu vo forme textu – 1 prvok tlačený **hrubým** štýlom písma

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | pridávaná matica |
| *riadok* | riadok element tlačeného hruým písmom |
| *stĺpec* | stĺpec element tlačeného hruým písmom |

Definícia na riadku 313 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addMatrixAsTextBoldUnderDiagonal (MatrixD *matrixD*) [static]

tlačí maticu tak, že dolná časť pod diagonálou je tlačená použitím hrubého písma

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | tlačená matica |

Definícia na riadku 412 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addMatrixAsTextDiagonalElementsExpressedForm (MatrixD *matrixD*) [static]

tlačí maticu tak, že ľavá časť matice zahŕňa len prvky na diagonal a pravá strana obsahuje všetky ostatné

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | tlačená matica |

Definícia na riadku 376 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addMatrixAsTextTwoBoldElementsOnAColumn (MatrixD *matrixD*, int *column*, int *row1*, int *row2*) [static]

tlačí maticu tak, že dva prvky na tom istom stĺpci tlačí **hrubým** písmom

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | pridávaná matica |
| *stĺpec* | stĺpec , v ktorom sú umiestnené oba prvky tlačené nahrubo |
| *row1* | riadok prvého prvku tlačeného nahrubo |
| *row2* | riadok druhého prvku tlačeného nahrubo |

Definícia na riadku 345 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addMatrixElement (double *element*) [static]

pridanie formátoveného reťatca prvku matice

##### Parameters:

|  |  |
| --- | --- |
| *element* | prvok, ktorý je pridávaný vo forme formatováneho reťazca |

Definícia na riadku 228 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addPicture (File *pictFile*, String *align*, String *height*, String *width*) [static]

pridáva obrázok do FO súboru

##### Parameters:

|  |  |
| --- | --- |
| *pictFile* | objekt File reprezentujúci obrázok |
| *align* | zarovnanie obrázka (center, left, right) |
| *height* | mierka výšky obrázka |
| *width* | mierka šírky obrázka |

Definícia na riadku 470 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addTail () [static]

pridáva pätu FO súboru

Definícia na riadku 218 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addText (String *str*) [static]

pridanie textu neformátovane

##### Parameters:

|  |  |
| --- | --- |
| *str* | pridávaný reťazec |

Definícia na riadku 109 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addTextBrLn (String *str*) [static]

pridanie textu nasledovaného znakom ´\n´ a FO reprezentáciou nového riadka

##### Parameters:

|  |  |
| --- | --- |
| *str* | pridávaný reťazec |

Definícia na riadku 119 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addTextInitialApproxValuesEqZero (MatrixD *matrixD*) [static]

tlačí nulovú aproximáciu pre všetky neznáme

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorej neznáme sú tlačené vo forme prvej aproximácie |

Definícia na riadku 440 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addTextln (String *str*) [static]

pridáva text vrátane znaku '\n'

##### Parameters:

|  |  |
| --- | --- |
| *str* | pridávaný reťazec |

Definícia na riadku 171 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addTextlnBrKeepWithNext (String *str*) [static]

pridáva text tak, že je ako blok stale zobrazený na nasledovnej strane s nasledovným blokom

##### Parameters:

|  |  |
| --- | --- |
| *str* | pridávaný reťazec |

Definícia na riadku 129 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addVectorAsText (VectorD *vectorD*) [static]

pridáva vector vo forme textu

##### Parameters:

|  |  |
| --- | --- |
| *vectorD* | pridávaný vektor |

Definícia na riadku 236 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.addVectorAsTextRSubstraction (VectorD *vectorD*, int *dividerRow*) [static]

pridáva vector vo forme textu vrátane reprezentácie |R substrakcie

##### Parameters:

|  |  |
| --- | --- |
| *vectorD* | pridávaný vektor |
| *dividerRow* | index riadka použitého pre R subtrakciu |

Definícia na riadku 260 súboru DynamicPDFGenerator.java.

#### static DynamicPDFGenerator core.DynamicPDFGenerator.getInstance () [static]

getter pre inštanciu

##### Returns:

inštanciu

Definícia na riadku 87 súboru DynamicPDFGenerator.java.

#### static String core.DynamicPDFGenerator.getText () [static]

getter String-ovej reprezentácie obsahu FO súboru

##### Returns:

String-ová reprezentácia obsahu FO súboru

Definícia na riadku 101 súboru DynamicPDFGenerator.java.

#### static void core.DynamicPDFGenerator.printAlgorithmIntoAFile (File *file*) throws FileNotFoundException, IOException [static]

vytvára XSL-FO (.fo) súbor vrátane vyberanej metódy a riešenej matice

##### Parameters:

|  |  |
| --- | --- |
| *file* | cesta k .fo súboru |

##### Exceptions:

|  |  |
| --- | --- |
| *FileNotFoundException* | výnimka hádzaná v prípade nenájdenia súboru |
| *IOException* | výnimka hádzaná v prípade neexistencie súboru (zlá cesta a pod.) |

Definícia na riadku 494 súboru DynamicPDFGenerator.java.

Member Data Documentation

#### String core.DynamicPDFGenerator.footerHeight = String.valueOf(20) [static, private]

výška päty dokumentu

Definícia na riadku 69 súboru DynamicPDFGenerator.java.

#### String core.DynamicPDFGenerator.headerHeight = String.valueOf(20) [static, private]

výška hlavičky dokumentu

Definícia na riadku 64 súboru DynamicPDFGenerator.java.

#### DynamicPDFGenerator core.DynamicPDFGenerator.instance [static, private]

inštancia triedy

Definícia na riadku 74 súboru DynamicPDFGenerator.java.

#### String core.DynamicPDFGenerator.ln = System.getProperty("line.separator") [static, private]

oddeľovač riadkov

Definícia na riadku 47 súboru DynamicPDFGenerator.java.

#### String core.DynamicPDFGenerator.pageHeight = String.valueOf(297) [static, private]

veľkosť strany - výška (297 mm )

Definícia na riadku 53 súboru DynamicPDFGenerator.java.

#### String core.DynamicPDFGenerator.pageWidth = String.valueOf(210) [static, private]

veľkosť strany - šírka (210 mm )

Definícia na riadku 59 súboru DynamicPDFGenerator.java.

#### StringBuilder core.DynamicPDFGenerator.text = new StringBuilder() [static, private]

StringBuilder objekt pre reprezentáciu obsahu dynamicky generovaného Fo súboru

Definícia na riadku 42 súboru DynamicPDFGenerator.java.

* + 1. core.FOtoPDFconverter

Static Public Member Functions

* static FOtoPDFconverter getInstance ()
* static void convertFO2PDF (File fo, File pdf) throws IOException, FOPException, TransformerException

Static Private Attributes

* static FopFactory fopFactory
* static FOtoPDFconverter instance

Detailed Description

Funkcia: Realizuje konverziu FO súboru na PDF použitím FOP technológií.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 45 súboru FOtoPDFconverter.java.

Member Function Documentation

#### static void core.FOtoPDFconverter.convertFO2PDF (File *fo*, File *pdf*) throws IOException, FOPException, TransformerException [static]

konvertuje FO súbor na PDF použitím FOP

##### Parameters:

|  |  |
| --- | --- |
| *fo* | FO súbor |
| *pdf* | cieľový PDF súbor |

##### Exceptions:

|  |  |
| --- | --- |
| *IOException* | výnimka hádzaná v prípade I/O chyby |
| *FOPException* | výnimka hádzaná pri použití FOP technológií (všeobecná) |
| *TransformerException* | výnimka hádzaná v prípade vzniku problému s transformáciou FO na PDF |

Definícia na riadku 81 súboru FOtoPDFconverter.java.

#### static FOtoPDFconverter core.FOtoPDFconverter.getInstance () [static]

getter

##### Returns:

Inštancia objektu konvertora

Definícia na riadku 69 súboru FOtoPDFconverter.java.

Member Data Documentation

#### FopFactory core.FOtoPDFconverter.fopFactory [static, private]

inštancia objektu fopFactory

Definícia na riadku 50 súboru FOtoPDFconverter.java.

#### FOtoPDFconverter core.FOtoPDFconverter.instance [static, private]

inštancia objektu konvertora

Definícia na riadku 55 súboru FOtoPDFconverter.java.

* + 1. core.Generator

Public Member Functions

* Generator ()
* void generateIntegerMatrixDirect (int size, int rangeFrom, int rangeTo) throws NegativeValueException, ReverseRangeValuesException
* void generateNonIntegerMatrixDirect (int size, int precision, double rangeFrom, double rangeTo) throws NegativeValueException, ReverseRangeValuesException
* MatrixD getDMatrix ()
* VectorD getDUnknowns ()
* MatrixI getIMatrix ()
* VectorI getIUnknowns ()
* void generateNonIntegerMatrixIndirect (int size, int precision, double rangeFrom, double rangeTo) throws NegativeValueException, ReverseRangeValuesException

Private Member Functions

* void generateVariablesValuesForInt ()
* void generateMatrixValuesAijForInt ()
* void generateMatrixValuesBiForInt ()
* void generateVariablesValuesForDouble ()
* void generateMatrixValuesAijForDouble ()
* void generateMatrixValuesBiForDouble ()
* double getRange (double rangeFrom, double rangeTo)
* int getRange (int rangeFrom, int rangeTo)
* void generateMatrixValuesAijForDoubleIndirect ()
* void generateMatrixValuesAiiForDoubleIndirect ()
* double getMaxNormForRow (int row)

Private Attributes

* int precision
* int absoluteLimit
* int centerValue
* double absoluteRange
* double movement
* Random random
* int[] iVariablesX
* int[][] iMatrix
* double[] dVariablesX
* double[][] dMatrix
* int size

Detailed Description

Funkcia: generuje náhodné matice.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 35 súboru Generator.java.

Constructor & Destructor Documentation

#### core.Generator.Generator ()

konštruktor, inicializuje Random objekt

Definícia na riadku 96 súboru Generator.java.

Member Function Documentation

#### void core.Generator.generateIntegerMatrixDirect (int *size*, int *rangeFrom*, int *rangeTo*) throws NegativeValueException, ReverseRangeValuesException

alokuje dostatok pamäte pre generovanie MatrixD, ktorá pozostáva z celočíselných hodnôt pre riešenia priamymi metódami

##### Parameters:

|  |  |
| --- | --- |
| *size* | veľkosť generovanej matice |
| *rangeFrom* | počatočná hodnota |
| *rangeTo* | limitná hodnota |

##### Exceptions:

|  |  |
| --- | --- |
| *NegativeValueException* | hádzaná v prípade objavenia sa zápornej hodnoty |
| *ReverseRangeValuesException* | hádzaná v prípade opačného poradia hodnôt rozsahu od-do |

Definícia na riadku 110 súboru Generator.java.

#### void core.Generator.generateMatrixValuesAiiForDoubleIndirect () [private]

generuje Aii maticu s číslami v desatinnom zobrazení pre nepriame metódy

Definícia na riadku 368 súboru Generator.java.

#### void core.Generator.generateMatrixValuesAijForDouble () [private]

generuje Aij hodnoty pre matice s desatinnými číslami

Definícia na riadku 215 súboru Generator.java.

#### void core.Generator.generateMatrixValuesAijForDoubleIndirect () [private]

generuje Aij matice s desatinnými číslami pre nepriame metódy

Definícia na riadku 346 súboru Generator.java.

#### void core.Generator.generateMatrixValuesAijForInt () [private]

generuje Aij hodnoty pre celočíselnú maticu

Definícia na riadku 180 súboru Generator.java.

#### void core.Generator.generateMatrixValuesBiForDouble () [private]

generuje Bi hodnoty pre maticu s desatinnými číslami

Definícia na riadku 227 súboru Generator.java.

#### void core.Generator.generateMatrixValuesBiForInt () [private]

generuje Bi hodnoty pre celočíselnú maticu

Definícia na riadku 192 súboru Generator.java.

#### void core.Generator.generateNonIntegerMatrixDirect (int *size*, int *precision*, double *rangeFrom*, double *rangeTo*) throws NegativeValueException, ReverseRangeValuesException

alokuje dostatok pamäte pre generovanú maticu neceločíselnú pre priame metódy

##### Parameters:

|  |  |
| --- | --- |
| *size* | veľkosť generovanej matice |
| *precision* | definuje koľko cifier po desatinnej čiarke sú brané do úvahy |
| *rangeFrom* | počatočná hodnota |
| *rangeTo* | limitná hodnota |

##### Exceptions:

|  |  |
| --- | --- |
| *NegativeValueException* | hádzaná v prípade objavenia sa zápornej hodnoty |
| *ReverseRangeValuesException* | hádzaná v prípade opačného poradia hodnôt rozsahu od-do |

Definícia na riadku 144 súboru Generator.java.

#### void core.Generator.generateNonIntegerMatrixIndirect (int *size*, int *precision*, double *rangeFrom*, double *rangeTo*) throws NegativeValueException, ReverseRangeValuesException

alokuje dostatok pamäte pre generovanú maticu neceločíselnú pre nepriame metódy

##### Parameters:

|  |  |
| --- | --- |
| *size* | veľkosť generovanej matice |
| *precision* | definuje koľko cifier po desatinnej čiarke sú brané do úvahy |
| *rangeFrom* | počatočná hodnota |
| *rangeTo* | limitná hodnota |

##### Exceptions:

|  |  |
| --- | --- |
| *NegativeValueException* | hádzaná v prípade objavenia sa zápornej hodnoty |
| *ReverseRangeValuesException* | hádzaná v prípade opačného poradia hodnôt rozsahu od-do |

Definícia na riadku 317 súboru Generator.java.

#### void core.Generator.generateVariablesValuesForDouble () [private]

generuje hodnoty neznámych pre neceločíselnú maticu

Definícia na riadku 205 súboru Generator.java.

#### void core.Generator.generateVariablesValuesForInt () [private]

generuje hodnoty neznámych pre celočíselnu maticu

Definícia na riadku 170 súboru Generator.java.

#### MatrixD core.Generator.getDMatrix ()

getter

##### Returns:

Matica s neceločíselnými prvkami

Definícia na riadku 241 súboru Generator.java.

#### VectorD core.Generator.getDUnknowns ()

getter

##### Returns:

Vector hľadaných hodnôt pre neceločíselné hodnoty

Definícia na riadku 249 súboru Generator.java.

#### MatrixI core.Generator.getIMatrix ()

getter

##### Returns:

Matica obsahujúca výlučne celé čísla

Definícia na riadku 257 súboru Generator.java.

#### VectorI core.Generator.getIUnknowns ()

getter

##### Returns:

Vector riešenia pozostávajúci z celých čísel

Definícia na riadku 265 súboru Generator.java.

#### double core.Generator.getMaxNormForRow (int *row*) [private]

počíta normu pre špecifický riadok generovanej matice

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | riadok, ktorého norma sa počíta |

##### Returns:

Maximálna norma matice

Definícia na riadku 387 súboru Generator.java.

#### double core.Generator.getRange (double *rangeFrom*, double *rangeTo*) [private]

počíta rozsah na základe požadovaných vlastností

##### Parameters:

|  |  |
| --- | --- |
| *rangeFrom* | nižšia hranice pre generovanie |
| *rangeTo* | vyššia hranice pre generovanie |

##### Returns:

computed range

Definícia na riadku 275 súboru Generator.java.

#### int core.Generator.getRange (int *rangeFrom*, int *rangeTo*) [private]

počíta rozsah na základe požadovaných vlastností, verzia pre celočíselný rozsah

##### Parameters:

|  |  |
| --- | --- |
| *rangeFrom* | nižšia hranice pre generovanie |
| *rangeTo* | vyššia hranice pre generovanie |

##### Returns:

Vypočítaný rozsah

Definícia na riadku 294 súboru Generator.java.

Member Data Documentation

#### int core.Generator.absoluteLimit [private]

absolútny limit

Definícia na riadku 45 súboru Generator.java.

#### double core.Generator.absoluteRange [private]

absolútny rozsah

Definícia na riadku 55 súboru Generator.java.

#### int core.Generator.centerValue [private]

hodnota pre centrovanie náhodne generovaných celočíselnych hodnôt celočíselnej matice

Definícia na riadku 50 súboru Generator.java.

#### double [][] core.Generator.dMatrix [private]

neceločíselná matica

Definícia na riadku 85 súboru Generator.java.

#### double [] core.Generator.dVariablesX [private]

neceločíselný vector neznámych

Definícia na riadku 80 súboru Generator.java.

#### int [][] core.Generator.iMatrix [private]

Celočíselná matica

Definícia na riadku 75 súboru Generator.java.

#### int [] core.Generator.iVariablesX [private]

celočíselný vektor neznámych

Definícia na riadku 70 súboru Generator.java.

#### double core.Generator.movement [private]

posun

Definícia na riadku 60 súboru Generator.java.

#### int core.Generator.precision [private]

precíznosť generovania

Definícia na riadku 40 súboru Generator.java.

#### Random core.Generator.random [private]

Random objekt použitý pre generovanie hodnôt

Definícia na riadku 65 súboru Generator.java.

#### int core.Generator.size [private]

veľkosť matice

Definícia na riadku 90 súboru Generator.java.

* + 1. exceptions.BaseDirectoryCreationException

Public Member Functions

* BaseDirectoryCreationException ()
* BaseDirectoryCreationException (String msg)

Detailed Description

Funkcia: hádzaná v prípade neúspešného vytvorenia základného dátového adresára obsahujúceho .fo, .pdf, .html súbory.

##### Author:

Pavol Dano

Definícia na riadku 22 súboru BaseDirectoryCreationException.java.

Constructor & Destructor Documentation

#### exceptions.BaseDirectoryCreationException.BaseDirectoryCreationException ()

konštruktor

Definícia na riadku 27 súboru BaseDirectoryCreationException.java.

#### exceptions.BaseDirectoryCreationException.BaseDirectoryCreationException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 34 súboru BaseDirectoryCreationException.java.

* + 1. exceptions.MatrixSizeException

Public Member Functions

* MatrixSizeException ()
* MatrixSizeException (String msg)

Detailed Description

Funkcia: reprezentuje chybový stav signalizujúci neočakávaný počet prvkov matice.

##### Author:

Pavol Dano

Definícia na riadku 22 súboru MatrixSizeException.java.

Constructor & Destructor Documentation

#### exceptions.MatrixSizeException.MatrixSizeException ()

konštruktor

Definícia na riadku 27 súboru MatrixSizeException.java.

#### exceptions.MatrixSizeException.MatrixSizeException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 34 súboru MatrixSizeException.java.

* + 1. exceptions.NegativeValueException

Public Member Functions

* NegativeValueException ()
* NegativeValueException (String message)

Detailed Description

Funkcia: hádzaná, keď sa záporná hodnota objaví v kontexte, kde takáto hodnota nie je očakávaná.

##### Author:

Pavol Dano

Definícia na riadku 22 súboru NegativeValueException.java.

Constructor & Destructor Documentation

#### exceptions.NegativeValueException.NegativeValueException ()

konštruktor

Definícia na riadku 27 súboru NegativeValueException.java.

#### exceptions.NegativeValueException.NegativeValueException (String *message*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *message* | správa zobrazená, ak je požadovaná |

Definícia na riadku 35 súboru NegativeValueException.java.

* + 1. exceptions.ReverseRangeValuesException

Public Member Functions

* ReverseRangeValuesException ()
* ReverseRangeValuesException (String msg)

Detailed Description

Funkcia: hádzaná, ak dolný a horný limit pre generovanie matíc majú prevrátené poradie, číže, ak dolná hranica > horná hranica.

##### Author:

Pavol Dano

Definícia na riadku 23 súboru ReverseRangeValuesException.java.

Constructor & Destructor Documentation

#### exceptions.ReverseRangeValuesException.ReverseRangeValuesException ()

konštruktor

Definícia na riadku 28 súboru ReverseRangeValuesException.java.

#### exceptions.ReverseRangeValuesException.ReverseRangeValuesException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 36 súboru ReverseRangeValuesException.java.

* + 1. linearSystems.data.MatrixD

Public Member Functions

* MatrixD ()
* MatrixD (double[][] matrix)
* int isZeroOnDiagonal ()
* double[] getVectorOnPosition (int row)
* void swapVectors (int row1, int row2)
* String toString ()
* String toString (int precisionBeforeComa, int precisionAfterComa)
* String toPrintedForm ()
* MatrixD clone ()
* void zeroMatrix ()

Public Attributes

* int rows
* int riadky
* double[][] matrix

Detailed Description

Funkcia: trieda reprezentujúca maticu pozostávajúcu z čísel s desatinnou čiarkou.

##### Author:

Pavol Dano

Definícia na riadku 20 súboru MatrixD.java.

Constructor & Destructor Documentation

#### linearSystems.data.MatrixD.MatrixD ()

konštruktor

Definícia na riadku 40 súboru MatrixD.java.

#### linearSystems.data.MatrixD.MatrixD (double *matrix*[][])

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *matrix* | inicializovaný poľom dát |

Definícia na riadku 47 súboru MatrixD.java.

Member Function Documentation

#### MatrixD linearSystems.data.MatrixD.clone ()

"konštruktor" pre kopírovanie objektu

##### Returns:

Matica s hodnotami rovnakými ako hodnoty objektu, na ktorom je metóda clone() volaná

Definícia na riadku 160 súboru MatrixD.java.

#### double [] linearSystems.data.MatrixD.getVectorOnPosition (int *row*)

vracia vektor reprezentujúci konkrétny riadok matice

##### Parameters:

|  |  |
| --- | --- |
| *row* | riadok matice |

##### Returns:

vracia vector matice

Definícia na riadku 72 súboru MatrixD.java.

#### int linearSystems.data.MatrixD.isZeroOnDiagonal ()

vracia -1, ak 0 sa objaví na diagonal matice

##### Returns:

-1 alebo i ako pozícia nuly na diagonále

Definícia na riadku 57 súboru MatrixD.java.

#### void linearSystems.data.MatrixD.swapVectors (int *row1*, int *row2*)

Vymieňa vektory row1 a row2

##### Parameters:

|  |  |
| --- | --- |
| *row1* | vymieňaný riadok |
| *row2* | vymieňaný riadok |

Definícia na riadku 83 súboru MatrixD.java.

#### String linearSystems.data.MatrixD.toPrintedForm ()

reprezentácia matice pre tlač do súboru vo forme reťazca znakov

##### Returns:

reprezentáciu matice vo forme reťazca

Definícia na riadku 142 súboru MatrixD.java.

#### String linearSystems.data.MatrixD.toString ()

reťazcová reprezentácia matice tlačenej na štandardný výstup implicitne použitím precíznosti o hodnote 2 cifier za desatinnou čiarkou

##### Returns:

reprezentácia matice vo forme reťazca znakov

Definícia na riadku 100 súboru MatrixD.java.

#### String linearSystems.data.MatrixD.toString (int *precisionBeforeComa*, int *precisionAfterComa*)

reprezentácia matice pre tlač na štandardný výstup vo forme reťazca znakov, tentokrát s definovanou precíznosťou

##### Parameters:

|  |  |
| --- | --- |
| *precisionBeforeComa* | počet cifier pred desatinnou čiarkou |
| *precisionAfterComa* | počet cifier za desatinnou čiarkou |

##### Returns:

reprezentácia matice vo forme reťazca znakov

Definícia na riadku 121 súboru MatrixD.java.

#### void linearSystems.data.MatrixD.zeroMatrix ()

nastaví hodnoty všetkých prvkov poľa matrix na hodnotu 0

Definícia na riadku 173 súboru MatrixD.java.

Member Data Documentation

#### int linearSystems.data.MatrixD.riadky

počet riadkov

Definícia na riadku 30 súboru MatrixD.java.

#### double [][] linearSystems.data.MatrixD.matrix

pole neceločíselných hodnôt – dáta matice

Definícia na riadku 35 súboru MatrixD.java.

#### int linearSystems.data.MatrixD.rows

počet riadkov

Definícia na riadku 25 súboru MatrixD.java.

* + 1. linearSystems.data.MatrixExecutionsHelper

Static Public Member Functions

* static void getRidOfZeroOnDiagonal (MatrixD matrixD, int position) throws MatrixEditableNotException
* static void sumTwoVectors (double[] destination, double[] source)
* static void divideVectorByNumber (double[] vector, double divider)
* static void multipleVectorByNumber (double[] vector, double multiplier)
* static boolean checkRowDominance (MatrixD matrixD) throws RowlyIndominantMatrixException
* static boolean checkColumnDominance (MatrixD matrixD) throws ColumnlyIndominantMatrixException
* static void divideElementsByDiagonalValues (MatrixD matrixD)
* static double getMaxNorm (MatrixD matrixD)
* static void copyVectors (VectorD source, VectorD dest)

Detailed Description

Funkcia: trieda MatrixExecutionsHelper je zodpovedná za vykonávanie dodatočných operácií nad maticami.

##### Author:

Pavol Dano

Definícia na riadku 27 súboru MatrixExecutionsHelper.java.

Member Function Documentation

#### static boolean linearSystems.data.MatrixExecutionsHelper.checkColumnDominance (MatrixD *matrixD*) throws ColumnlyIndominantMatrixException [static]

kontroluje stĺpcovú dominanciu matice

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorej dominancia je testovaná |

##### Returns:

true, ak matica je stĺpcovo dominantná

##### Exceptions:

|  |  |
| --- | --- |
| *ColumnlyIndominantMatrixException* | hádzaná, ak matica nie je stĺpcovo dominantná |

Definícia na riadku 122 súboru MatrixExecutionsHelper.java.

#### static boolean linearSystems.data.MatrixExecutionsHelper.checkRowDominance (MatrixD *matrixD*) throws RowlyIndominantMatrixException [static]

kontroluje riadkovú dominanciu matice

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorej dominancia je testovaná |

##### Returns:

true, ak matica je riadkovo dominantná

##### Exceptions:

|  |  |
| --- | --- |
| *ColumnlyIndominantMatrixException* | hádzaná, ak matica nie je stĺpcovo dominantná |

Definícia na riadku 98 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.copyVectors (VectorD *source*, VectorD *dest*) [static]

kopíruje vektory zo zdroja do nové miesto

##### Parameters:

|  |  |
| --- | --- |
| *source* | zdrojový vektor |
| *dest* | miesto uloženia kopírovaného vektora |

Definícia na riadku 179 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.divideElementsByDiagonalValues (MatrixD *matrixD*) [static]

delí všetky členy riadku hodnotou prvku na diagonále

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorej členy sú delené |

Definícia na riadku 142 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.divideVectorByNumber (double[] *vector*, double *divider*) [static]

delí celý vektor skalárom

##### Parameters:

|  |  |
| --- | --- |
| *vector* | vector, ktorého prvky sú predelené |
| *divider* | hodnota, ktorou je daný vector predelený |

Definícia na riadku 74 súboru MatrixExecutionsHelper.java.

#### static double linearSystems.data.MatrixExecutionsHelper.getMaxNorm (MatrixD *matrixD*) [static]

vracia maximálnu hodnotu všetkých prvkov na diagonále okrem diagonálneho pre konkrétnu maticu

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | objekt matice |

##### Returns:

Maximálna hodnota riadku matice matrixD

Definícia na riadku 157 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.getRidOfZeroOnDiagonal (MatrixD *matrixD*, int *position*) throws MatrixEditableNotException [static]

odstraňuje hodnotu 0 z konkrétneho riadku matice matrixD

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica s nulou na diagonále |
| *position* | pozícia na diagonále, ktorej prvok má hodnotu nula |

##### Exceptions:

|  |  |
| --- | --- |
| *MatrixEditableNotException* | hádzaná, ak nie je možné upraviť maticu požadovaným spôsobom |

Definícia na riadku 36 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.multipleVectorByNumber (double[] *vector*, double *multiplier*) [static]

násobí vector skalárom

##### Parameters:

|  |  |
| --- | --- |
| *vector* | násobený vektor |
| *multiplier* | násobiteľ |

Definícia na riadku 85 súboru MatrixExecutionsHelper.java.

#### static void linearSystems.data.MatrixExecutionsHelper.sumTwoVectors (double[] *destination*, double[] *source*) [static]

sčítava 2 vektory

##### Parameters:

|  |  |
| --- | --- |
| *destination* | konečný vector |
| *source* | zdrojový vektor |

Definícia na riadku 63 súboru MatrixExecutionsHelper.java.

* + 1. linearSystems.data.MatrixI

Public Member Functions

* MatrixI ()
* MatrixI (int[][] matrix)
* String toString ()
* String toString (int precisionBeforeComa)
* String toPrintedForm ()

Public Attributes

* int rows
* int riadky
* int[][] matrix

Detailed Description

Funkcia: trieda reprezentujúca maticu pre uchovanie celočíselných hodnôt.

##### Author:

Pavol Dano

Definícia na riadku 21 súboru MatrixI.java.

Constructor & Destructor Documentation

#### linearSystems.data.MatrixI.MatrixI ()

konštruktor

Definícia na riadku 41 súboru MatrixI.java.

#### linearSystems.data.MatrixI.MatrixI (int *matrix*[][])

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *matrix* | pole uchovávajúce hodnoty |

Definícia na riadku 48 súboru MatrixI.java.

Member Function Documentation

#### String linearSystems.data.MatrixI.toPrintedForm ()

reprezentácia matice pre tlač do súboru vo forme reťazca znakov

##### Returns:

Reprezentácie matice ako reťazca

Definícia na riadku 97 súboru MatrixI.java.

#### String linearSystems.data.MatrixI.toString ()

reprezentácia matice vo forme reťazca pre tlač do okna aplikácie

##### Returns:

reprezentácie matice ako reťazca

Definícia na riadku 59 súboru MatrixI.java.

#### String linearSystems.data.MatrixI.toString (int *precisionBeforeComa*)

reprezentácia matice vo forme reťazca pre tlač do okna aplikácie

##### Parameters:

|  |  |
| --- | --- |
| *precisionBeforeComa* | počet cifier pred desatinnou čiarkou |

##### Returns:

reprezentácie matice ako reťazca

Definícia na riadku 78 súboru MatrixI.java.

Member Data Documentation

#### int linearSystems.data.MatrixI.riadky

počet riadkov

Definícia na riadku 31 súboru MatrixI.java.

#### int [][] linearSystems.data.MatrixI.matrix

pole celých čísel reprezentujúcich skutočné matematické data matice

Definícia na riadku 36 súboru MatrixI.java.

#### int linearSystems.data.MatrixI.rows

počet riadkov matice

Definícia na riadku 26 súboru MatrixI.java.

* + 1. linearSystems.data.VectorD

Public Member Functions

* VectorD ()
* VectorD (double[] vector)
* double[] getVector ()
* void zeroVector ()
* String toString ()
* String toString (int precision)
* String toPrintedForm ()

Public Attributes

* double[] vector

Detailed Description

Funkcia: reprezentuje matematický vektor pre úschovu čísel v pohyblivej rádovej čiarke.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 24 súboru VectorD.java.

Constructor & Destructor Documentation

#### linearSystems.data.VectorD.VectorD ()

konštruktor

Definícia na riadku 34 súboru VectorD.java.

#### linearSystems.data.VectorD.VectorD (double[] *vector*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *vector* | pole reálnych hodnôt vektora. |

Definícia na riadku 41 súboru VectorD.java.

Member Function Documentation

#### double [] linearSystems.data.VectorD.getVector ()

getter

##### Returns:

vektor hodnôt

Definícia na riadku 49 súboru VectorD.java.

#### String linearSystems.data.VectorD.toPrintedForm ()

reprezentácia vektora vo forme reťazca pri ukladaní do súboru

##### Returns:

Reprezentácia vektora vo forme objektu typu String

Definícia na riadku 92 súboru VectorD.java.

#### String linearSystems.data.VectorD.toString ()

reprezentácia vektora vo forme reťazca pri zobrazení napríklad na štandardný výstup

##### Returns:

Reprezentácia vektora vo forme objektu typu String

Definícia na riadku 64 súboru VectorD.java.

#### String linearSystems.data.VectorD.toString (int *precision*)

reprezentácia vektora vo forme reťazca pri zobrazení napríklad na štandardný výstup vrátane počtu cifier (za desatinnou čiarkou), ktoré sa majú brať do úvahy

##### Parameters:

|  |  |
| --- | --- |
| *precision* | počet cifier braných do úvahy |

##### Returns:

reprezentácia vektora vo forme objektu typu String

Definícia na riadku 77 súboru VectorD.java.

#### void linearSystems.data.VectorD.zeroVector ()

Definícia na riadku 53 súboru VectorD.java.

Member Data Documentation

#### double [] linearSystems.data.VectorD.vector

pole skutočných matematických hodnôt vektora

Definícia na riadku 29 súboru VectorD.java.

* + 1. linearSystems.data.VectorI

Public Member Functions

* VectorI ()
* VectorI (int[] vector)
* int[] getVector ()
* String toString ()
* String toString (int precision)
* String toPrintedForm ()

Public Attributes

* int[] vector

Detailed Description

Funkcia: reprezentuje matematický vektor pre úschovu čísel v pevnej rádovej čiarke.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 24 súboru VectorI.java.

Constructor & Destructor Documentation

#### linearSystems.data.VectorI.VectorI ()

konštruktor

Definícia na riadku 34 súboru VectorI.java.

#### linearSystems.data.VectorI.VectorI (int[] *vector*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *vector* | pole matematických hodnôt vektora |

Definícia na riadku 41 súboru VectorI.java.

Member Function Documentation

#### int [] linearSystems.data.VectorI.getVector ()

getter

##### Returns:

vector int[]

Definícia na riadku 49 súboru VectorI.java.

#### String linearSystems.data.VectorI.toPrintedForm ()

reprezentácia vektora vo forme reťazca pri ukladaní do súboru

##### Returns:

reprezentácia vektora vo forme objektu typu String

Definícia na riadku 85 súboru VectorI.java.

#### String linearSystems.data.VectorI.toString ()

reprezentácia vektora vo forme reťazca pri zobrazení napríklad na štandardný výstup

##### Returns:

reprezentácia vektora vo forme objektu typu String

Definícia na riadku 58 súboru VectorI.java.

Member Data Documentation

#### int [] linearSystems.data.VectorI.vector

vector consisting of integer numbers

Definícia na riadku 29 súboru VectorI.java.

* + 1. linearSystems.dataHandlers.MatrixReader

Private Attributes

* double[][] matrix
* double[][] originalMatrix

Static Private Attributes

* static MatrixReader instance

Detailed Description

Funkcia: číta maticu zo špecifikovaného súboru.

##### Author:

Pavol Dano

Definícia na riadku 32 súboru MatrixReader.java.

Member Data Documentation

#### MatrixReader linearSystems.dataHandlers.MatrixReader.instance [static, private]

inštancia triedy

Definícia na riadku 37 súboru MatrixReader.java.

#### double [][] linearSystems.dataHandlers.MatrixReader.matrix [private]

matica pre uloženie načítaných hodnôť (neskôr upravovaná)

Definícia na riadku 42 súboru MatrixReader.java.

#### double [][] linearSystems.dataHandlers.MatrixReader.originalMatrix [private]

matica pre uloženie načítaných hodnôť (nepodlieha modifikácii)

Definícia na riadku 48 súboru MatrixReader.java.

* + 1. linearSystems.dataHandlers.MatrixWriter

Public Member Functions

* void setMatrix (MatrixD matrixD)
* void setMatrix (MatrixI matrixI)
* void setResult (VectorI vectorI)
* void setResult (VectorD vectorD)
* MatrixWriter ()
* void writeMatrixToFile (String path) throws FileNotFoundException, IOException

Private Attributes

* MatrixD matrixD
* MatrixI matrixI
* VectorD vectorD
* VectorI vectorI

Detailed Description

Funkcia: zapisuje maticu do požadovaného súboru.

##### Author:

Pavol Dano

Definícia na riadku 35 súboru MatrixWriter.java.

Constructor & Destructor Documentation

#### linearSystems.dataHandlers.MatrixWriter.MatrixWriter ()

konštruktor

Definícia na riadku 96 súboru MatrixWriter.java.

Member Function Documentation

#### void linearSystems.dataHandlers.MatrixWriter.setMatrix (MatrixD *matrixD*)

setter pre matrixD

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | nová matica |

Definícia na riadku 61 súboru MatrixWriter.java.

#### void linearSystems.dataHandlers.MatrixWriter.setMatrix (MatrixI *matrixI*)

setter pre matrixI

##### Parameters:

|  |  |
| --- | --- |
| *matrixI* | nová matica |

Definícia na riadku 70 súboru MatrixWriter.java.

#### void linearSystems.dataHandlers.MatrixWriter.setResult (VectorI *vectorI*)

setter pre vectorI (celočíselné premenné)

##### Parameters:

|  |  |
| --- | --- |
| *vectorI* | nový vektor riešenia |

Definícia na riadku 79 súboru MatrixWriter.java.

#### void linearSystems.dataHandlers.MatrixWriter.setResult (VectorD *vectorD*)

setter pre vectorD (neceločíselné premenné)

##### Parameters:

|  |  |
| --- | --- |
| *vectorD* | nový vektor riešenia |

Definícia na riadku 88 súboru MatrixWriter.java.

#### void linearSystems.dataHandlers.MatrixWriter.writeMatrixToFile (String *path*) throws FileNotFoundException, IOException

zapisuje matrixI alebo matrixD maticu do súboru špecifikovaného cestou

##### Parameters:

|  |  |
| --- | --- |
| *path* | cesta k novému súboru |

##### Exceptions:

|  |  |
| --- | --- |
| *FileNotFoundException* | hádzaná, ak súbor nemôže byť nájdený |
| *IOException* | hádzaná, ak I/O operácie sa nedajú uskutočniť |

Definícia na riadku 105 súboru MatrixWriter.java.

Member Data Documentation

#### MatrixD linearSystems.dataHandlers.MatrixWriter.matrixD [private]

matice s hodnotami v pohyblivej rádovej čiarke

Definícia na riadku 40 súboru MatrixWriter.java.

#### MatrixI linearSystems.dataHandlers.MatrixWriter.matrixI [private]

matice s hodnotami celočíselnými (nepohyblivá rádova čiarka)

Definícia na riadku 44 súboru MatrixWriter.java.

#### VectorD linearSystems.dataHandlers.MatrixWriter.vectorD [private]

vektor hodnôt v pohyblivej rádovej čiarke

Definícia na riadku 49 súboru MatrixWriter.java.

#### VectorI linearSystems.dataHandlers.MatrixWriter.vectorI [private]

vektor hodnôt v nepohyblivej rádovej čiarke

Definícia na riadku 53 súboru MatrixWriter.java.

* + 1. linearSystems.dataHandlers.MethodsClassLoader

Classes

* class JavaFileFilter

Static Public Member Functions

* static MethodsClassLoader getInstance ()
* static LinearSystemsSolvable[] loadMethodsClasses () throws ClassNotFoundException, InstantiationException, IllegalAccessException, NoMethodsAsJavaClassesException

Private Member Functions

* MethodsClassLoader (ClassLoader parent)

Static Private Attributes

* static MethodsClassLoader instance
* static LinearSystemsSolvable[] methods
* static JavaFileFilter fileFilter

Detailed Description

Funkcia: MethodsClassLoader dynamicky načítavá triedy reprezentujúce metódy pre vykonávanie operácií nad maticami s cieľom hľadať ich riešenie z implicitne definovaného adresára použitím reflexie.

##### Author:

Pavol Dano

Definícia na riadku 32 súboru MethodsClassLoader.java.

Constructor & Destructor Documentation

#### linearSystems.dataHandlers.MethodsClassLoader.MethodsClassLoader (ClassLoader *parent*) [private]

konštruktor inicializovaný objektov typu JavaFileFilter(hľadanie súborov so sufixom .class)

##### Parameters:

|  |  |
| --- | --- |
| *parent* | ClassLoader |

Definícia na riadku 71 súboru MethodsClassLoader.java.

Member Function Documentation

#### static MethodsClassLoader linearSystems.dataHandlers.MethodsClassLoader.getInstance () [static]

getter

##### Returns:

vracia inštanciu triedy

Definícia na riadku 62 súboru MethodsClassLoader.java.

#### static LinearSystemsSolvable [] linearSystems.dataHandlers.MethodsClassLoader.loadMethodsClasses () throws ClassNotFoundException, InstantiationException, IllegalAccessException, NoMethodsAsJavaClassesException [static]

dynamicky lokalizuje “.class” súbory a skúša ich načítať pomocou reflexive do bežiacej aplikácie ako objekty typu LinearSystemsSolver (rozhranie)

##### Returns:

Pole načítaných metód

##### Exceptions:

|  |  |
| --- | --- |
| *ClassNotFoundException* | hádzaná, ak žiadna definícia pre triedu špecifikovanú týmto menom neexistuje |
| *InstantiationException* | hádzaná, ak object nemože byť inicializovaný |
| *IllegalAccessException* | hádzaná, ak aplikácia nemá žiaden prístup k definícii tejto triedy |
| *NoMethodsAsJavaClassesException* | hádzaná, ak žiadne metódy neboli nájdené/načítané |

Definícia na riadku 85 súboru MethodsClassLoader.java.

Member Data Documentation

#### JavaFileFilter linearSystems.dataHandlers.MethodsClassLoader.fileFilter [static, private]

filter súborov pre “.class” súbory

Definícia na riadku 47 súboru MethodsClassLoader.java.

#### MethodsClassLoader linearSystems.dataHandlers.MethodsClassLoader.instance [static, private]

inštancia triedy

Definícia na riadku 37 súboru MethodsClassLoader.java.

#### LinearSystemsSolvable [] linearSystems.dataHandlers.MethodsClassLoader.methods [static, private]

pole načítaných metód

Definícia na riadku 42 súboru MethodsClassLoader.java.

* + 1. linearSystems.dataHandlers.MethodsClassLoader.JavaFileFilter

Dedí od FilenameFilter.

Public Member Functions

* JavaFileFilter (String mask)
* boolean accept (File dir, String name)
* void setMask (String mask)

Detailed Description

Funkcia: vnútorná trieda javaFileFilter pre načítavanie súborov definovaných pomocou masky.

##### Author:

Pavol Dano

Definícia na riadku 123 súboru MethodsClassLoader.java.

Constructor & Destructor Documentation

#### linearSystems.dataHandlers.MethodsClassLoader.JavaFileFilter.JavaFileFilter (String *mask*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *mask* | maska použitá pri hľadaní špecifických súborov |

Definícia na riadku 136 súboru MethodsClassLoader.java.

Member Function Documentation

#### boolean linearSystems.dataHandlers.MethodsClassLoader.JavaFileFilter.accept (File *dir*, String *name*)

kontroluje, či názov súboru sa zhoduje s hľadanou maskou

##### Parameters:

|  |  |
| --- | --- |
| *dir* | adresár, v ktorom je súbor |
| *name* | meno súboru |

##### Returns:

true, ak sa zhodujú, inak false

Definícia na riadku 146 súboru MethodsClassLoader.java.

#### void linearSystems.dataHandlers.MethodsClassLoader.JavaFileFilter.setMask (String *mask*)

setter pre masku

##### Parameters:

|  |  |
| --- | --- |
| *mask* | nová hodnota masky |

Definícia na riadku 158 súboru MethodsClassLoader.java.

* + 1. linearSystems.exceptions.dominanceExceptions.ColumnlyIndominantMatrixException

Public Member Functions

* ColumnlyIndominantMatrixException ()
* ColumnlyIndominantMatrixException (String msg)

Detailed Description

Funkcia: reprezentuje maticovú indominanciu na základe výpočtu pre stĺpec.

##### Author:

Pavol Dano

Definícia na riadku 22 súboru ColumnlyIndominantMatrixException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.dominanceExceptions.ColumnlyIndominantMatrixException.ColumnlyIndominantMatrixException ()

konštruktor

Definícia na riadku 27 súboru ColumnlyIndominantMatrixException.java.

#### linearSystems.exceptions.dominanceExceptions.ColumnlyIndominantMatrixException.ColumnlyIndominantMatrixException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 35 súboru ColumnlyIndominantMatrixException.java.

* + 1. linearSystems.exceptions.dominanceExceptions.DiagonalyIndominantMatrixException

Public Member Functions

* DiagonalyIndominantMatrixException ()
* DiagonalyIndominantMatrixException (String msg)

Detailed Description

Funkcia: reprezentuje maticovú indominanciu (vo všeobecnosti pre maticu).

##### Author:

Pavol Dano

Definícia na riadku 22 súboru DiagonalyIndominantMatrixException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.dominanceExceptions.DiagonalyIndominantMatrixException.DiagonalyIndominantMatrixException ()

konštruktor

Definícia na riadku 27 súboru DiagonalyIndominantMatrixException.java.

#### linearSystems.exceptions.dominanceExceptions.DiagonalyIndominantMatrixException.DiagonalyIndominantMatrixException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 35 súboru DiagonalyIndominantMatrixException.java.

* + 1. linearSystems.exceptions.dominanceExceptions.RowlyIndominantMatrixException

Public Member Functions

* RowlyIndominantMatrixException ()
* RowlyIndominantMatrixException (String msg)

Detailed Description

Funkcia: reprezentuje stav riadkovej nedominantnosti matice.

##### Author:

Pavol Dano

Definícia na riadku 21 súboru RowlyIndominantMatrixException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.dominanceExceptions.RowlyIndominantMatrixException.RowlyIndominantMatrixException ()

konštruktor

Definícia na riadku 26 súboru RowlyIndominantMatrixException.java.

#### linearSystems.exceptions.dominanceExceptions.RowlyIndominantMatrixException.RowlyIndominantMatrixException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 34 súboru RowlyIndominantMatrixException.java.

* + 1. linearSystems.exceptions.MatrixEditableNotException

Public Member Functions

* MatrixEditableNotException ()
* MatrixEditableNotException (String msg)

Detailed Description

Funkcia: hádzaná, ak matica nie je schopná úpravy do takej formy, aby na nej bol konkrétny výpočet realizovateľný.

##### Author:

Pavol Dano

Definícia na riadku 21 súboru MatrixEditableNotException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.MatrixEditableNotException.MatrixEditableNotException ()

konštruktor

Definícia na riadku 26 súboru MatrixEditableNotException.java.

#### linearSystems.exceptions.MatrixEditableNotException.MatrixEditableNotException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 33 súboru MatrixEditableNotException.java.

* + 1. linearSystems.exceptions.NoMethodsAsJavaClassesException

Public Member Functions

* NoMethodsAsJavaClassesException ()
* NoMethodsAsJavaClassesException (String msg)

Detailed Description

Funkcia: hádzaná, ak žiadne metódy neboli nájdené/načítané.

##### Author:

Pavol Dano

Definícia na riadku 23 súboru NoMethodsAsJavaClassesException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.NoMethodsAsJavaClassesException.NoMethodsAsJavaClassesException ()

konštruktor

Definícia na riadku 28 súboru NoMethodsAsJavaClassesException.java.

#### linearSystems.exceptions.NoMethodsAsJavaClassesException.NoMethodsAsJavaClassesException (String *msg*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *msg* | správa zobrazená, ak je požadovaná |

Definícia na riadku 36 súboru NoMethodsAsJavaClassesException.java.

* + 1. linearSystems.exceptions.WrongDoubleFormatException

Public Member Functions

* WrongDoubleFormatException ()
* WrongDoubleFormatException (String message)

Detailed Description

Funkcia: hádzaná, ak súbor dát pre reprezentáciu matice obsahuje tak člen, ktorý nie je možné parsovať do objektu typu double

##### Author:

Pavol Dano

Definícia na riadku 23 súboru WrongDoubleFormatException.java.

Constructor & Destructor Documentation

#### linearSystems.exceptions.WrongDoubleFormatException.WrongDoubleFormatException ()

konštruktor

Definícia na riadku 28 súboru WrongDoubleFormatException.java.

#### linearSystems.exceptions.WrongDoubleFormatException.WrongDoubleFormatException (String *message*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *message* | správa zobrazená, ak je požadovaná |

Definícia na riadku 36 súboru WrongDoubleFormatException.java.

* + 1. linearSystems.interfaces.LinearSystemsSolvable Interface Reference

Obr. Diagram dedičnosti pre linearSystems.interfaces.LinearSystemsSolvable

Public Member Functions

* abstract String getName ()
* abstract String getPathToExample ()
* abstract void execute (MatrixD matrixD)
* abstract VectorD getResults ()

Detailed Description

Funkcia: Signaluzuje schopnosť metódy riešiť sústavy lineárnych rovníc.

##### Author:

Pavol Dano

Definícia na riadku 24 súboru LinearSystemsSolvable.java.

Member Function Documentation

#### abstract void linearSystems.interfaces.LinearSystemsSolvable.execute (MatrixD *matrixD*) [pure virtual]

vykonáva danú metódu

Implemented in linearSystems.methods.JacobiMethod, linearSystems.methods.GaussSeidelMethod, linearSystems.methods.GaussianMethod, linearSystems.methods.CholeskyMethod, and linearSystems.methods.GaussianMethodAdapted.

#### abstract String linearSystems.interfaces.LinearSystemsSolvable.getName () [pure virtual]

vracia meno metódy

Implemented in linearSystems.methods.CholeskyMethod, linearSystems.methods.JacobiMethod, linearSystems.methods.GaussSeidelMethod, linearSystems.methods.GaussianMethod, and linearSystems.methods.GaussianMethodAdapted.

#### abstract String linearSystems.interfaces.LinearSystemsSolvable.getPathToExample () [pure virtual]

vracia cestu k implicitnému súboru obsahujúcemu ukážkový príklad metódy

Implemented in linearSystems.methods.CholeskyMethod, linearSystems.methods.JacobiMethod, linearSystems.methods.GaussSeidelMethod, linearSystems.methods.GaussianMethod, and linearSystems.methods.GaussianMethodAdapted.

#### abstract VectorD linearSystems.interfaces.LinearSystemsSolvable.getResults () [pure virtual]

Vracia výsledok výpočtu vo forme VectorD objektu

Implemented in linearSystems.methods.JacobiMethod, linearSystems.methods.GaussSeidelMethod, linearSystems.methods.CholeskyMethod, linearSystems.methods.GaussianMethod, and linearSystems.methods.GaussianMethodAdapted.

* + 1. linearSystems.interfaces.Stepable Interface Reference

Obr. Diagram dedičnosti pre linearSystems.interfaces.Stepable

Public Member Functions

* void setStepsLeft (int stepsLeft)
* int getStepsLeft ()

Detailed Description

Funkcia: signalizuje, že metóda je krokovateľná. To jest, v procese iterácie sa výpočet odráža od počtu vykonaných krokov.

##### Author:

Pavol Dano

Definícia na riadku 23 súboru Stepable.java.

Member Function Documentation

#### int linearSystems.interfaces.Stepable.getStepsLeft ()

getter pre počet krokov

##### Returns:

Počet krokov

Implemented in linearSystems.methods.JacobiMethod, and linearSystems.methods.GaussSeidelMethod.

#### void linearSystems.interfaces.Stepable.setStepsLeft (int *stepsLeft*)

setter pre počet krokov

##### Parameters:

|  |  |
| --- | --- |
| *stepsLeft* | počet krokov |

Implemented in linearSystems.methods.JacobiMethod, and linearSystems.methods.GaussSeidelMethod.

* + 1. linearSystems.interfaces.ToleranceSetable Interface Reference

Obr. Diagram dedičnosti pre linearSystems.interfaces.ToleranceSetable

Public Member Functions

* void setTolerance (double tolerance)
* double getTolerance ()

Detailed Description

Funkcia: signalizuje, že metóda je realizovateľná s ohľadom na dosiahnutú presnosť. To jest, v procese iterácie sa výpočet odráža od dosiahnutej presnosti.

##### Author:

Pavol Dano

Definícia na riadku 23 súboru ToleranceSetable.java.

Member Function Documentation

#### double linearSystems.interfaces.ToleranceSetable.getTolerance ()

getter pre toleranciu.

##### Returns:

Aktuálna tolerancia

Implemented in linearSystems.methods.JacobiMethod, and linearSystems.methods.GaussSeidelMethod.

#### void linearSystems.interfaces.ToleranceSetable.setTolerance (double *tolerance*)

setter pre toleranciu

##### Parameters:

|  |  |
| --- | --- |
| *tolerance* | nová tolerancia (presnosť) |

Implemented in linearSystems.methods.JacobiMethod, and linearSystems.methods.GaussSeidelMethod.

* + 1. linearSystems.methods.CholeskyMethod

Obr. Diagram dedičnosti pre linearSystems.methods.CholeskyMethod

Public Member Functions

* void execute (MatrixD matrixD)
* VectorD getResults ()
* String getName ()
* String toString ()
* String getPathToExample ()

Private Member Functions

* void fullfillMatrixOnDiagonal (MatrixD matrixD)
* void splitMatrixAIntoMatrixBAndC ()
* void eliminateColumn (int col\_row)
* void eliminateRow (int col\_row)
* void bij (int i, int j)
* double Ebij (int i, int j)
* void cij (int i, int j)
* double Ecij (int i, int j)
* void calculateXiYi ()
* void yij (int row)
* void xij (int row)
* double Exij (int row)
* double Eyij (int row)

Private Attributes

* String name = "Cholesky"
* String pathToExample = "/examples/cholesky.txt"
* MatrixD matrixD\_A
* MatrixD matrixD\_B
* MatrixD matrixD\_C
* VectorD vectorD\_y
* VectorD vectorD\_x

Detailed Description

Funkcia: objekt reprezentujúci Choleského priamej metódy pre riešenie sústav lineárnych rovníc.

##### Author:

Pavol Dano

Definícia na riadku 24 súboru CholeskyMethod.java.

Member Function Documentation

#### void linearSystems.methods.CholeskyMethod.bij (int *i*, int *j*) [private]

výpočet bij

##### Parameters:

|  |  |
| --- | --- |
| *i* | riadok |
| *j* | stĺpec |

Definícia na riadku 158 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.calculateXiYi () [private]

výpočet xi a yi

Definícia na riadku 202 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.cij (int *i*, int *j*) [private]

výpočet cij

##### Parameters:

|  |  |
| --- | --- |
| *i* | riadok |
| *j* | stĺpec |

Definícia na riadku 181 súboru CholeskyMethod.java.

#### double linearSystems.methods.CholeskyMethod.Ebij (int *i*, int *j*) [private]

výpočet sumácie reprezentovanej ako E(bij)

##### Parameters:

|  |  |
| --- | --- |
| *i* | riadok |
| *j* | riadok |

##### Returns:

sumácia príslušných hodnôt

Definícia na riadku 168 súboru CholeskyMethod.java.

#### double linearSystems.methods.CholeskyMethod.Ecij (int *i*, int *j*) [private]

výpočet sumácie reprezentovanej ako E(cij)

##### Parameters:

|  |  |
| --- | --- |
| *i* | riadok |
| *j* | riadok |

##### Returns:

sumácia príslušných hodnôt

Definícia na riadku 191 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.eliminateColumn (int *col\_row*) [private]

eliminácia stĺpca

##### Parameters:

|  |  |
| --- | --- |
| *col\_riadok* | stĺpec-riadok eliminated |

Definícia na riadku 137 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.eliminateRow (int *col\_row*) [private]

eliminácia stĺpca

##### Parameters:

|  |  |
| --- | --- |
| *col\_riadok* | stĺpec- riadok eliminovaný |

Definícia na riadku 147 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.execute (MatrixD *matrixD*) [virtual]

pre bližšie informácie nahliadnite do teoretickej časti bakalárskej práce, prosím

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | riešená matica |

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 69 súboru CholeskyMethod.java.

#### double linearSystems.methods.CholeskyMethod.Exij (int *row*) [private]

výpočd Eij pre špecifický riadok a premennú x

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | riadok |

##### Returns:

sumácia Exij

Definícia na riadku 241 súboru CholeskyMethod.java.

#### double linearSystems.methods.CholeskyMethod.Eyij (int *row*) [private]

výpočet Eij pre špecifický riadok and premennú y

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | riadok |

##### Returns:

sumácia Eyij

Definícia na riadku 255 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.fullfillMatrixOnDiagonal (MatrixD *matrixD*) [private]

napĺňa maticu hodnotami 0 a 1 nasledovne: diagonálne prvky majú hodnotu 1 a prvky pod a nad diagonálou hodnotu 0

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | napĺňaná matica |

Definícia na riadku 99 súboru CholeskyMethod.java.

#### String linearSystems.methods.CholeskyMethod.getName () [virtual]

getter pre meno metódy

##### Returns:

meno metódy

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 276 súboru CholeskyMethod.java.

#### String linearSystems.methods.CholeskyMethod.getPathToExample () [virtual]

vracia cestu k súboru s príkladom

##### Returns:

vracia cestu k súboru s príkladom

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 289 súboru CholeskyMethod.java.

#### VectorD linearSystems.methods.CholeskyMethod.getResults () [virtual]

vracia výsledok výpočtu

##### Returns:

Výsledok výpočtu

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 268 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.splitMatrixAIntoMatrixBAndC () [private]

inicializuje maticu A a B tak, že v ďalšom kroku je prvý riadok matice A kopírovaný do prvého riadka matice B predeleného diagonálnym prvkom a skopírovaným do matice C napĺňaním zvyšným jej prvkokv

Definícia na riadku 117 súboru CholeskyMethod.java.

#### String linearSystems.methods.CholeskyMethod.toString ()

Definícia na riadku 281 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.xij (int *row*) [private]

výpočet xij pre konkrétny riadok

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | riadok |

Definícia na riadku 230 súboru CholeskyMethod.java.

#### void linearSystems.methods.CholeskyMethod.yij (int *row*) [private]

výpočet yij pre konkrétny riadok

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | riadok |

Definícia na riadku 220 súboru CholeskyMethod.java.

Member Data Documentation

#### MatrixD linearSystems.methods.CholeskyMethod.matrixD\_A [private]

matica A

Definícia na riadku 39 súboru CholeskyMethod.java.

#### MatrixD linearSystems.methods.CholeskyMethod.matrixD\_B [private]

matica B

Definícia na riadku 44 súboru CholeskyMethod.java.

#### MatrixD linearSystems.methods.CholeskyMethod.matrixD\_C [private]

matica C

Definícia na riadku 49 súboru CholeskyMethod.java.

#### String linearSystems.methods.CholeskyMethod.name = "Cholesky" [private]

meno metódy

Definícia na riadku 29 súboru CholeskyMethod.java.

#### String linearSystems.methods.CholeskyMethod.pathToExample = "/examples/cholesky.txt" [private]

cesta k súboru

Definícia na riadku 34 súboru CholeskyMethod.java.

#### VectorD linearSystems.methods.CholeskyMethod.vectorD\_x [private]

vektor x

Definícia na riadku 59 súboru CholeskyMethod.java.

#### VectorD linearSystems.methods.CholeskyMethod.vectorD\_y [private]

vektor y

Definícia na riadku 54 súboru CholeskyMethod.java.

* + 1. linearSystems.methods.GaussianMethod

Obr. Diagram dedičnosti pre linearSystems.methods.GaussianMethod

Public Member Functions

* void execute (MatrixD matrixD)
* VectorD getResults ()
* String getName ()
* String toString ()
* String getPathToExample ()

Private Member Functions

* void handleZerosOnDiagonal () throws MatrixEditableNotException
* void doExecution ()
* void provideForwardElimination ()
* void eliminateColumn (int column, int startAt)
* void expressXn ()
* void expressXi (int row)
* void computeResult (int row, double rest)

Private Attributes

* String name = "Gaussian"
* String pathToExample = "/examples/gauss.txt"
* MatrixD matrixD
* double[][] matrix
* double[] result
* int rows
* int riadky
* double[] vector

Detailed Description

Funkcia: objekt reprezentujúci Gaussovu priamu metódu pre riešenie sústav lineárnych rovníc.

##### Author:

Pavol Dano

Definícia na riadku 26 súboru GaussianMethod.java.

Member Function Documentation

#### void linearSystems.methods.GaussianMethod.computeResult (int *row*, double *rest*) [private]

vypočíta výsledok pre konkrétny riadok s poznaním hodnoty zvyšku rovnice

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | pre ktorý sa hodnota počíta |
| *rest* | zvyšná časť riadku (hodnotovo) |

Definícia na riadku 184 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.doExecution () [private]

vykoná Gaussovu eliminačnú metódu

Definícia na riadku 114 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.eliminateColumn (int *column*, int *startAt*) [private]

eliminácia stĺpca na hodnotu 1 na pozícii diagonálneho prvku vrátane výpočtu ostatných prvkov pod diagonálov na hodnotu 0

##### Parameters:

|  |  |
| --- | --- |
| *stĺpec* | eliminovaný stĺpec |
| *startAt* | pozícia, na ktorej eliminácia začína |

Definícia na riadku 144 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.execute (MatrixD *matrixD*) [virtual]

vykonanie samotnej metódy

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorá je objektom vykonávania operácií |

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 73 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.expressXi (int *row*) [private]

vyjadrenie riadku xi (spätná substitúcia)

##### Parameters:

|  |  |
| --- | --- |
| *riadok* | Index riadku, kde zvyšná hodnota sa počíta v závislosti od hodnoty tohto parametre |

Definícia na riadku 167 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.expressXn () [private]

vyjadrenie poslednej neznámej(posledný riadok)

Definícia na riadku 158 súboru GaussianMethod.java.

#### String linearSystems.methods.GaussianMethod.getName () [virtual]

vracia meno metódy

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 190 súboru GaussianMethod.java.

#### String linearSystems.methods.GaussianMethod.getPathToExample () [virtual]

getter

##### Returns:

cesta k súboru s implicitným príkladom

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 203 súboru GaussianMethod.java.

#### VectorD linearSystems.methods.GaussianMethod.getResults () [virtual]

Získanie výsledku vo forme objektu VectorD

##### Returns:

Výsledok riešenia

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 107 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.handleZerosOnDiagonal () throws MatrixEditableNotException [private]

zbavuje sa núl na diagonal, ak sa objavia

##### Exceptions:

|  |  |
| --- | --- |
| *MatrixEditableNotException* | hádzaná pri nemožnosti pozmeniť maticu do vhodnej formy |

Definícia na riadku 95 súboru GaussianMethod.java.

#### void linearSystems.methods.GaussianMethod.provideForwardElimination () [private]

eliminácia všetkých riadkov jeden po druhom

Definícia na riadku 131 súboru GaussianMethod.java.

#### String linearSystems.methods.GaussianMethod.toString ()

Definícia na riadku 195 súboru GaussianMethod.java.

Member Data Documentation

#### int linearSystems.methods.GaussianMethod.riadky [private]

počet stĺpcov matice

Definícia na riadku 61 súboru GaussianMethod.java.

#### double [][] linearSystems.methods.GaussianMethod.matrix [private]

pole reprezentujúce hodnoty matice

Definícia na riadku 46 súboru GaussianMethod.java.

#### MatrixD linearSystems.methods.GaussianMethod.matrixD [private]

riešená matica

Definícia na riadku 41 súboru GaussianMethod.java.

#### String linearSystems.methods.GaussianMethod.name = "Gaussian" [private]

meno metódy

Definícia na riadku 31 súboru GaussianMethod.java.

#### String linearSystems.methods.GaussianMethod.pathToExample = "/examples/gauss.txt" [private]

cesta k súboru s implicitným príkladom

Definícia na riadku 36 súboru GaussianMethod.java.

#### double [] linearSystems.methods.GaussianMethod.result [private]

vypočítaný výsledok

Definícia na riadku 51 súboru GaussianMethod.java.

#### int linearSystems.methods.GaussianMethod.rows [private]

počet riadkov matcie

Definícia na riadku 56 súboru GaussianMethod.java.

#### double [] linearSystems.methods.GaussianMethod.vector [private]

dočasný vector pre manipuláciu s maticou

Definícia na riadku 66 súboru GaussianMethod.java.

* + 1. linearSystems.methods.GaussianMethodAdapted

Obr. Diagram dedičnosti pre linearSystems.methods.GaussianMethodAdapted

Public Member Functions

* GaussianMethodAdapted ()
* void execute (MatrixD matrixD)
* VectorD getResults ()
* String getName ()
* String toString ()
* String getPathToExample ()

Private Member Functions

* void matrixToAdaptedForm (MatrixD matrixD)
* int findRowWithMaxInColumn (MatrixD matrixD, int column, int startFrom)

Private Attributes

* String name = "Gaussian with partial pivoting"
* GaussianMethod gaussMethod
* String pathToExample = "/examples/gaussAdapted.txt"

Detailed Description

Funkcia: Reprezentuje Gaussovu eliminačnú metódu s výberom hlavného prvku

##### Author:

Pavol Dano

Definícia na riadku 25 súboru GaussianMethodAdapted.java.

Constructor & Destructor Documentation

#### linearSystems.methods.GaussianMethodAdapted.GaussianMethodAdapted ()

konštruktor

Definícia na riadku 45 súboru GaussianMethodAdapted.java.

Member Function Documentation

#### void linearSystems.methods.GaussianMethodAdapted.execute (MatrixD *matrixD*) [virtual]

riešenie matice

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | riešená matica |

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 53 súboru GaussianMethodAdapted.java.

#### int linearSystems.methods.GaussianMethodAdapted.findRowWithMaxInColumn (MatrixD *matrixD*, int *column*, int *startFrom*) [private]

hľadá riadok s maximálnou hodnotou v špecifikovanom stĺpci

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | matica, ktorej stĺpce sú predmetom hľadania |
| *stĺpec* | analyzovaný stĺpec |
| *startFrom* | počiatočná pozícia |

##### Returns:

riadok zahŕňajpci maximálnu hodnotu

Definícia na riadku 92 súboru GaussianMethodAdapted.java.

#### String linearSystems.methods.GaussianMethodAdapted.getName () [virtual]

vracia meno metódy

##### Returns:

meno metódy

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 110 súboru GaussianMethodAdapted.java.

#### String linearSystems.methods.GaussianMethodAdapted.getPathToExample () [virtual]

##### Returns:

vracia cestu k súboru s príkladom

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 122 súboru GaussianMethodAdapted.java.

#### VectorD linearSystems.methods.GaussianMethodAdapted.getResults () [virtual]

vracia výsledok výpočtu

##### Returns:

Vracia výsledok výpočtu

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 63 súboru GaussianMethodAdapted.java.

#### void linearSystems.methods.GaussianMethodAdapted.matrixToAdaptedForm (MatrixD *matrixD*) [private]

upravuje maticu do formy, ktorá má lepšie vlastnosti pre vykonávanie

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | upravovaná matica |

Definícia na riadku 72 súboru GaussianMethodAdapted.java.

#### String linearSystems.methods.GaussianMethodAdapted.toString ()

Definícia na riadku 115 súboru GaussianMethodAdapted.java.

Member Data Documentation

#### GaussianMethod linearSystems.methods.GaussianMethodAdapted.gaussMethod [private]

klasická Gaussova eliminačná metóda

Definícia na riadku 35 súboru GaussianMethodAdapted.java.

#### String linearSystems.methods.GaussianMethodAdapted.name = "Gaussian with partial pivoting" [private]

oficiálne meno metódy

Definícia na riadku 30 súboru GaussianMethodAdapted.java.

#### String linearSystems.methods.GaussianMethodAdapted.pathToExample = "/examples/gaussAdapted.txt" [private]

cesta k súboru s príkladom

Definícia na riadku 40 súboru GaussianMethodAdapted.java.

* + 1. linearSystems.methods.GaussSeidelMethod

Obr. Diagram dedičnosti pre linearSystems.methods.GaussSeidelMethod

Public Member Functions

* void execute (MatrixD matrixD)
* void setStepsLeft (int stepsLeft)
* int getStepsLeft ()
* void setTolerance (double tolerance)
* double getTolerance ()
* String getName ()
* String toString ()
* VectorD getResults ()
* String getPathToExample ()

Private Member Functions

* void checkDominance () throws DiagonalyIndominantMatrixException
* void beginIterations ()
* void iterateByStepCount ()
* void iterateByTolerance ()
* boolean isToleranceKept ()
* void computeActualTolerance ()
* double EBeforeXi (int XiPosition)
* double EAfterXi (int XiPosition)

Private Attributes

* String name = "Gauss-Seidel"
* String pathToExample = "/examples/gaussSeidel.txt"
* MatrixD matrixD
* double[][] iterationArray
* double requiredTolerance = 0
* int stepsDone = 0
* int stepsLeft = 0

Detailed Description

Funkcia: Vykonáva Gauss-Seidelovu nepriamu metódu pre riešenie sústav lineárnych rovníc.

##### Author:

Pavol Dano

Definícia na riadku 31 súboru GaussSeidelMethod.java.

Member Function Documentation

#### void linearSystems.methods.GaussSeidelMethod.beginIterations () [private]

inicializuje iteračný proces

Definícia na riadku 112 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.checkDominance () throws DiagonalyIndominantMatrixException [private]

kontroluje dominanciu matice

##### Exceptions:

|  |  |
| --- | --- |
| *DiagonalyIndominantMatrixException* | hádzaná, keď matica nie je dominantná podľa riadkov ani stĺpcov |

Definícia na riadku 95 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.computeActualTolerance () [private]

Počíta aktuálne rozdiely v aproximáciach

##### Parameters:

|  |  |
| --- | --- |
| *toleranceDeviations* | aktuálne rozdiely v aproximáciach |

Definícia na riadku 185 súboru GaussSeidelMethod.java.

#### double linearSystems.methods.GaussSeidelMethod.EAfterXi (int *XiPosition*) [private]

sumácia hodnôt xi po začiatku analýzy

##### Parameters:

|  |  |
| --- | --- |
| *XiPosition* | pozícia analyzovaného prvku |

##### Returns:

sumácia všetkých hodnôt (Xi,Xn)

Definícia na riadku 211 súboru GaussSeidelMethod.java.

#### double linearSystems.methods.GaussSeidelMethod.EBeforeXi (int *XiPosition*) [private]

sumácia hodnôt xi pred začiatkom analýzy

##### Parameters:

|  |  |
| --- | --- |
| *XiPosition* | position of analysed elements |

##### Returns:

sumácia všetkých hodnôt, pre ktoré platí: <X0,Xi)

Definícia na riadku 197 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.execute (MatrixD *matrixD*) [virtual]

vykonanie metódy

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | riešená matica |

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 77 súboru GaussSeidelMethod.java.

#### String linearSystems.methods.GaussSeidelMethod.getName () [virtual]

getter pre meno metódy

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 257 súboru GaussSeidelMethod.java.

#### String linearSystems.methods.GaussSeidelMethod.getPathToExample () [virtual]

getter pre cestu k súboru s príkladom

##### Returns:

cesta k súboru s príkladom

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 278 súboru GaussSeidelMethod.java.

#### VectorD linearSystems.methods.GaussSeidelMethod.getResults () [virtual]

vracia výsledok výpočtu

##### Returns:

Vracia výsledok výpočtu

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 270 súboru GaussSeidelMethod.java.

#### int linearSystems.methods.GaussSeidelMethod.getStepsLeft ()

getter pre počet zostávajúcich krokov vykonávania

##### Returns:

počet zvyšných krokov vykonania

Implementuje linearSystems.interfaces.Stepable.

Definícia na riadku 233 súboru GaussSeidelMethod.java.

#### double linearSystems.methods.GaussSeidelMethod.getTolerance ()

getter pre toleranciu

##### Returns:

aktuálnu toleranciu

Implementuje linearSystems.interfaces.ToleranceSetable.

Definícia na riadku 250 súboru GaussSeidelMethod.java.

#### boolean linearSystems.methods.GaussSeidelMethod.isToleranceKept () [private]

kontroluje, či momentálne tolerancia je dosiahnutá alebo nie

##### Returns:

true, ak tolerancia je dosiahnutá, inak false

Definícia na riadku 166 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.iterateByStepCount () [private]

vykonáva iteračný proces na základe definovaného počtu krokov

Definícia na riadku 132 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.iterateByTolerance () [private]

vykonáva iteračný proces na základe vopred definovanej tolerancie

Definícia na riadku 149 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.setStepsLeft (int *stepsLeft*)

setter pre zostávajúce kroky

##### Parameters:

|  |  |
| --- | --- |
| *stepsLeft* | nový počet zostávajúcich krokov vykonávania |

Implementuje linearSystems.interfaces.Stepable.

Definícia na riadku 224 súboru GaussSeidelMethod.java.

#### void linearSystems.methods.GaussSeidelMethod.setTolerance (double *tolerance*)

setter pre toleranciu

##### Parameters:

|  |  |
| --- | --- |
| *tolerance* | nová hodnota tolerancie |

Implementuje linearSystems.interfaces.ToleranceSetable.

Definícia na riadku 242 súboru GaussSeidelMethod.java.

#### String linearSystems.methods.GaussSeidelMethod.toString ()

Definícia na riadku 262 súboru GaussSeidelMethod.java.

Member Data Documentation

#### double [][] linearSystems.methods.GaussSeidelMethod.iterationArray [private]

dočasné pomocné iteračné pole

Definícia na riadku 51 súboru GaussSeidelMethod.java.

#### MatrixD linearSystems.methods.GaussSeidelMethod.matrixD [private]

riešená matica

Definícia na riadku 46 súboru GaussSeidelMethod.java.

#### String linearSystems.methods.GaussSeidelMethod.name = "Gauss-Seidel" [private]

meno metódy

Definícia na riadku 36 súboru GaussSeidelMethod.java.

#### String linearSystems.methods.GaussSeidelMethod.pathToExample = "/examples/gaussSeidel.txt" [private]

cesta k súboru s príkladom

Definícia na riadku 41 súboru GaussSeidelMethod.java.

#### double linearSystems.methods.GaussSeidelMethod.requiredTolerance = 0 [private]

požadovaná tolerancia

Definícia na riadku 56 súboru GaussSeidelMethod.java.

#### int linearSystems.methods.GaussSeidelMethod.stepsDone = 0 [private]

počet už vykonaných krokov

Definícia na riadku 61 súboru GaussSeidelMethod.java.

#### int linearSystems.methods.GaussSeidelMethod.stepsLeft = 0 [private]

počet zostávajúcich krokov

Definícia na riadku 66 súboru GaussSeidelMethod.java.

* + 1. linearSystems.methods.JacobiMethod

Obr. Diagram dedičnosti pre linearSystems.methods.JacobiMethod

Public Member Functions

* void execute (MatrixD matrixD)
* void setStepsLeft (int stepsLeft)
* int getStepsLeft ()
* void setTolerance (double tolerance)
* double getTolerance ()
* String getName ()
* String toString ()
* VectorD getResults ()
* String getPathToExample ()

Private Member Functions

* void checkDominance () throws DiagonalyIndominantMatrixException
* void beginIterations ()
* void computeRequiredTolerance ()
* void iterateByStepCount ()
* void iterateByTolerance ()
* boolean isToleranceKept ()
* void computeActualTolerance ()
* double ERow (int row)

Private Attributes

* String name = "Jacobi"
* String pathToExample = "/examples/jacobi.txt"
* MatrixD matrixD
* double[][] iterationArray
* double requiredTolerance = 0
* int stepsDone = 0
* int stepsLeft = 0
* double q = 0

Detailed Description

Funkcia: objekt reprezentujúci Jacobiho nepriamu metódu pre riešenie sústav lineárnych rovníc.

##### Author:

Pavol Dano

Definícia na riadku 30 súboru JacobiMethod.java.

Member Function Documentation

#### void linearSystems.methods.JacobiMethod.beginIterations () [private]

zahájenie iteračného procesu

Definícia na riadku 119 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.checkDominance () throws DiagonalyIndominantMatrixException [private]

kontroluje riadkovú a stĺpcovu dominanciu

##### Exceptions:

|  |  |
| --- | --- |
| *DiagonalyIndominantMatrixException* | hádzaná, keď matica nie je dominantná podľa riadkov ani stĺpcov |

Definícia na riadku 102 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.computeActualTolerance () [private]

vypočíta aktuálnu toleranciu

##### Parameters:

|  |  |
| --- | --- |
| *toleranceDeviations* | aktuálny rozdiel vo vypočítaných hodnotách aproximácií |

Definícia na riadku 205 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.computeRequiredTolerance () [private]

počíta požadovanú toleranciu pre špecifické *q* a požadovanú presnosť

Definícia na riadku 141 súboru JacobiMethod.java.

#### double linearSystems.methods.JacobiMethod.ERow (int *row*) [private]

vypočíta hodnotu, o ktorú ma byť vektor b znížený

##### Parameters:

|  |  |
| --- | --- |
| *row* | riadok matice |

##### Returns:

Suma hodnôt

Definícia na riadku 217 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.execute (MatrixD *matrixD*) [virtual]

vykonanie metódy

##### Parameters:

|  |  |
| --- | --- |
| *matrixD* | riešená matica |

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 81 súboru JacobiMethod.java.

#### String linearSystems.methods.JacobiMethod.getName () [virtual]

getter pre meno metódy

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 265 súboru JacobiMethod.java.

#### String linearSystems.methods.JacobiMethod.getPathToExample () [virtual]

getter pre cestu k súboru s príkladom

##### Returns:

cesta k súboru s príkladom

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 286 súboru JacobiMethod.java.

#### VectorD linearSystems.methods.JacobiMethod.getResults () [virtual]

vracia výsledok výpočtu

##### Returns:

vracia výsledok výpočtu

Implementuje linearSystems.interfaces.LinearSystemsSolvable.

Definícia na riadku 278 súboru JacobiMethod.java.

#### int linearSystems.methods.JacobiMethod.getStepsLeft ()

getter pre počet zostávajúcich krokov

##### Returns:

Počet zostávajúcich krokov

Implementuje linearSystems.interfaces.Stepable.

Definícia na riadku 241 súboru JacobiMethod.java.

#### double linearSystems.methods.JacobiMethod.getTolerance ()

getter pre toleranciu

##### Returns:

vracia aktuálnu toleranciu

Implementuje linearSystems.interfaces.ToleranceSetable.

Definícia na riadku 258 súboru JacobiMethod.java.

#### boolean linearSystems.methods.JacobiMethod.isToleranceKept () [private]

kontroluje, či tolerancia je už dosiahnutá alebo nie

##### Returns:

true, ak je tolerancia dosiahnutá, inak false

Definícia na riadku 185 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.iterateByStepCount () [private]

vykonáva výpočet na základe požadovaného počtu krokov

Definícia na riadku 150 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.iterateByTolerance () [private]

vykonáva výpočet na základe požadovanej tolerancie

Definícia na riadku 165 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.setStepsLeft (int *stepsLeft*)

setter pre zostávajúce kroky

##### Parameters:

|  |  |
| --- | --- |
| *stepsLeft* | nový počet zostávajúcich krokov vykonávania |

Implementuje linearSystems.interfaces.Stepable.

Definícia na riadku 232 súboru JacobiMethod.java.

#### void linearSystems.methods.JacobiMethod.setTolerance (double *tolerance*)

setter pre toleranciu

##### Parameters:

|  |  |
| --- | --- |
| *tolerance* | nová tolerancia |

Implementuje linearSystems.interfaces.ToleranceSetable.

Definícia na riadku 250 súboru JacobiMethod.java.

#### String linearSystems.methods.JacobiMethod.toString ()

Definícia na riadku 270 súboru JacobiMethod.java.

Member Data Documentation

#### double [][] linearSystems.methods.JacobiMethod.iterationArray [private]

dočasné pomocné iteračné pole

Definícia na riadku 50 súboru JacobiMethod.java.

#### MatrixD linearSystems.methods.JacobiMethod.matrixD [private]

riešená matica

Definícia na riadku 45 súboru JacobiMethod.java.

#### String linearSystems.methods.JacobiMethod.name = "Jacobi" [private]

meno metódy

Definícia na riadku 35 súboru JacobiMethod.java.

#### String linearSystems.methods.JacobiMethod.pathToExample = "/examples/jacobi.txt" [private]

cesta k súboru s príkladom

Definícia na riadku 40 súboru JacobiMethod.java.

#### double linearSystems.methods.JacobiMethod.q = 0 [private]

max norma matrice

Definícia na riadku 75 súboru JacobiMethod.java.

#### double linearSystems.methods.JacobiMethod.requiredTolerance = 0 [private]

požadovaná tolerancia

Definícia na riadku 55 súboru JacobiMethod.java.

#### int linearSystems.methods.JacobiMethod.stepsDone = 0 [private]

počet už vykonaných krokov

Definícia na riadku 60 súboru JacobiMethod.java.

#### int linearSystems.methods.JacobiMethod.stepsLeft = 0 [private]

počet zostávajúcich krokov

Definícia na riadku 65 súboru JacobiMethod.java.

* + 1. rounder.DoubleRounder

Static Public Member Functions

* static double round (double number)
* static double round (double number, int decimalPlace)

Private Member Functions

* DoubleRounder ()

Static Private Attributes

* static int precision = 2

Detailed Description

Funkcia: Zaokrúhľuje čísla s desatinnou rádovou čiarkou.

##### Author:

Pavol Dano

Definícia na riadku 24 súboru DoubleRounder.java.

Constructor & Destructor Documentation

#### rounder.DoubleRounder.DoubleRounder () [private]

konštruktor

Definícia na riadku 34 súboru DoubleRounder.java.

Member Function Documentation

#### static double rounder.DoubleRounder.round (double *number*) [static]

zaokrúhľuje čísla s desatinnou rádovou čiarkou   
Parameters:

|  |  |
| --- | --- |
| *number* | zaokrúhľované číslo |

##### Returns:

zaokrúhľované číslo

Definícia na riadku 42 súboru DoubleRounder.java.

#### static double rounder.DoubleRounder.round (double *number*, int *decimalPlace*) [static]

Zaokrúhľuje čísla s desatinnou rádovou čiarkou

##### Parameters:

|  |  |
| --- | --- |
| *number* | zaokrúhľované číslo |
| *decimalPlace* | počet desatinných miest, na ktoré a číslo *number* zaokrúhľuje |

##### Returns:

zaokrúhľované číslo

Definícia na riadku 55 súboru DoubleRounder.java.

Member Data Documentation

#### int rounder.DoubleRounder.precision = 2 [static, private]

implicitná precíznosť

Definícia na riadku 29 súboru DoubleRounder.java.

* + 1. swingUI.AboutFrame

Obr. Diagram dedičnosti pre swingUI.AboutFrame

Public Member Functions

* AboutFrame ()

Private Member Functions

* void setTukeIcon ()
* void initComponents ()
* void jButton\_returnActionPerformed (java.awt.event.ActionEvent evt)

Private Attributes

* javax.swing.JButton jButton\_return
* javax.swing.JScrollPane jScrollPane1
* javax.swing.JTextArea jTextArea1

Detailed Description

Funkcia: zobrazuje základné informácie o aplikácii.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 25 súboru AboutFrame.java.

Constructor & Destructor Documentation

#### swingUI.AboutFrame.AboutFrame ()

vytvára nový rámec AboutFrame

Definícia na riadku 28 súboru AboutFrame.java.

Member Function Documentation

#### void swingUI.AboutFrame.initComponents () [private]

inicializuje komponenty

Reimplementované z swingUI.ParentFrame.

Definícia na riadku 52 súboru AboutFrame.java.

#### void swingUI.AboutFrame.jButton\_returnActionPerformed (java.awt.event.ActionEvent *evt*) [private]

uvoľňuje rámec po tom, čo je zneviditeľnený

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý actionEvent |

Definícia na riadku 100 súboru AboutFrame.java.

#### void swingUI.AboutFrame.setTukeIcon () [private]

nastavuje "tukeIcon.png" ikonu

Definícia na riadku 40 súboru AboutFrame.java.

Member Data Documentation

#### javax.swing.JButton swingUI.AboutFrame.jButton\_return [private]

Definícia na riadku 106 súboru AboutFrame.java.

#### javax.swing.JScrollPane swingUI.AboutFrame.jScrollPane1 [private]

Definícia na riadku 107 súboru AboutFrame.java.

#### javax.swing.JTextArea swingUI.AboutFrame.jTextArea1 [private]

Definícia na riadku 108 súboru AboutFrame.java.

* + 1. swingUI.BasicGeneratorFrame

Obr. Diagram dedičnosti pre swingUI.BasicGeneratorFrame

Classes

* class OpenFileButton

Public Member Functions

* BasicGeneratorFrame (Generator generator)

Static Public Attributes

* static DynamicHTMLGenerator htmlGenerator
* static final String baseDataDirPath = "LinearSystemsSolver"
* static File baseDataDirFile = null

Private Member Functions

* void initComponents ()
* void setCalculatorIcon ()
* void jButton\_resetActionPerformed (java.awt.event.ActionEvent evt)
* void jButton\_generateActionPerformed (java.awt.event.ActionEvent evt)
* void jButtonReturnActionPerformed (java.awt.event.ActionEvent evt)
* void jCheckBox\_top\_ShowHereActionPerformed (java.awt.event.ActionEvent evt)
* void radioButton\_iterationableMatrixItemStateChanged (java.awt.event.ItemEvent evt)
* void generateMatrix () throws Exception
* void printMatrixIntoInfoArea () throws BaseDirectoryCreationException
* void printMatrixIntoAFile (String path)
* void checkIfAllFieldsAreSelected () throws Exception

Static Private Member Functions

* static File createBaseDirectoryFile () throws BaseDirectoryCreationException

Private Attributes

* Generator generator
* javax.swing.ButtonGroup buttonGroup\_methods
* javax.swing.JButton jButtonReturn
* javax.swing.JButton jButton\_generate
* javax.swing.JButton jButton\_reset
* javax.swing.JButton jButton\_search
* javax.swing.JCheckBox jCheckBox\_left\_intOnly
* javax.swing.JCheckBox jCheckBox\_showUnknows
* javax.swing.JCheckBox jCheckBox\_top\_ShowHere
* javax.swing.JCheckBox jCheckBox\_top\_saveToAFile
* javax.swing.JEditorPane jEditorPane
* javax.swing.JLabel jLabel1
* javax.swing.JLabel jLabelPrecision
* javax.swing.JLabel jLabelRange
* javax.swing.JLabel jLabel\_Size
* javax.swing.JLabel jLabel\_pathToFile
* javax.swing.JPanel jPanel\_left
* javax.swing.JPanel jPanel\_leftAdditional
* javax.swing.JPanel jPanel\_top
* javax.swing.JScrollPane jScrollPane1
* javax.swing.JTextField jTextField\_pathToFile
* javax.swing.JTextField jTextField\_precision
* javax.swing.JTextField jTextField\_rangeFrom
* javax.swing.JTextField jTextField\_rangeTo
* javax.swing.JTextField jTextField\_sizeOfMatrixGenerated
* javax.swing.JRadioButton radioButton\_classicMatrix
* javax.swing.JRadioButton radioButton\_iterationableMatrix

Detailed Description

Funkcia: obsluhuje stavy GUI.

##### Author:

Pavol Dano

##### Version:

1.0

Definícia na riadku 41 súboru BasicGeneratorFrame.java.

Constructor & Destructor Documentation

#### swingUI.BasicGeneratorFrame.BasicGeneratorFrame (Generator *generator*)

Vytvára nový rámec BasicGeneratorFrame

##### Parameters:

|  |  |
| --- | --- |
| *generator* | objekt zviazaný s rámcom |

Definícia na riadku 78 súboru BasicGeneratorFrame.java.

Member Function Documentation

#### void swingUI.BasicGeneratorFrame.checkIfAllFieldsAreSelected () throws Exception [private]

kontroluje výber všetkých nevyhnutných polí

##### Exceptions:

|  |  |
| --- | --- |
| *Exception* | výnimka hádzaná v prípade nekorektného zvolenia check boxov a radio buttons |

Definícia na riadku 644 súboru BasicGeneratorFrame.java.

#### static File swingUI.BasicGeneratorFrame.createBaseDirectoryFile () throws BaseDirectoryCreationException [static, private]

vytvára základný dátový adresár, ak je potrebný

##### Returns:

reprezentácia základného údajového adresára aplikácie

##### Exceptions:

|  |  |
| --- | --- |
| *BaseDirectoryCreationException* | výnimka hádzaná v prípade neúspešného vytvorenia základného údajového adresára |

Definícia na riadku 598 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.generateMatrix () throws Exception [private]

generuje maticu na základe používateľských požiadavok

##### Exceptions:

|  |  |
| --- | --- |
| *Exception* | Výnimka propagovaná vyššie v hierarchii volaní, keď vzniká |

Definícia na riadku 479 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.initComponents () [private]

inicializuje komponenty

Reimplementované z swingUI.ParentFrame.

Definícia na riadku 90 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.jButton\_generateActionPerformed (java.awt.event.ActionEvent *evt*) [private]

generuje novú maticu na základe používateľských požiadavok

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 380 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.jButton\_resetActionPerformed (java.awt.event.ActionEvent *evt*) [private]

resetuje cestu k súboru s maticou

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 372 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.jButtonReturnActionPerformed (java.awt.event.ActionEvent *evt*) [private]

rámec zaniká po kliknutí na tlačidlo return

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 415 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.jCheckBox\_top\_ShowHereActionPerformed (java.awt.event.ActionEvent *evt*) [private]

vyberá alebo odstraňuje výber check boxov "show unknowns"

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 423 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.printMatrixIntoAFile (String *path*) [private]

metóda, ktorá tlačí generovanú maticu do súboru

##### Parameters:

|  |  |
| --- | --- |
| *path* | tlač matice do súboru lokalizovaného parametrom path |

Definícia na riadku 617 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.printMatrixIntoInfoArea () throws BaseDirectoryCreationException [private]

tlačí generovanú maticu a vektor riešenia do textového poľa okna aplikácie

##### Exceptions:

|  |  |
| --- | --- |
| *BaseDirectoryCreationException* | výnimka hádzaná v prípade neúspešného vytvorenia základného údajového adresára |

Definícia na riadku 551 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.radioButton\_iterationableMatrixItemStateChanged (java.awt.event.ItemEvent *evt*) [private]

mení stav len v prípade výberu “int-only” možnosti

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 437 súboru BasicGeneratorFrame.java.

#### void swingUI.BasicGeneratorFrame.setCalculatorIcon () [private]

nastavuje ikonu kalkulačky v záhlaví rámca

Definícia na riadku 360 súboru BasicGeneratorFrame.java.

Member Data Documentation

#### File swingUI.BasicGeneratorFrame.baseDataDirFile = null [static]

reprezentácia základného údajového adresára aplikácie

Definícia na riadku 65 súboru BasicGeneratorFrame.java.

#### final String swingUI.BasicGeneratorFrame.baseDataDirPath = "LinearSystemsSolver" [static]

cesta k dátovému adresáru

Definícia na riadku 57 súboru BasicGeneratorFrame.java.

#### javax.swing.ButtonGroup swingUI.BasicGeneratorFrame.buttonGroup\_methods [private]

Definícia na riadku 447 súboru BasicGeneratorFrame.java.

#### Generator swingUI.BasicGeneratorFrame.generator [private]

inštancia generátora

Definícia na riadku 46 súboru BasicGeneratorFrame.java.

#### DynamicHTMLGenerator swingUI.BasicGeneratorFrame.htmlGenerator [static]

Initial value: DynamicHTMLGenerator

.getInstance()

inštancia objektu HTML generátora

Definícia na riadku 51 súboru BasicGeneratorFrame.java.

#### javax.swing.JButton swingUI.BasicGeneratorFrame.jButton\_generate [private]

Definícia na riadku 449 súboru BasicGeneratorFrame.java.

#### javax.swing.JButton swingUI.BasicGeneratorFrame.jButton\_reset [private]

Definícia na riadku 450 súboru BasicGeneratorFrame.java.

#### javax.swing.JButton swingUI.BasicGeneratorFrame.jButton\_search [private]

Definícia na riadku 451 súboru BasicGeneratorFrame.java.

#### javax.swing.JButton swingUI.BasicGeneratorFrame.jButtonReturn [private]

Definícia na riadku 448 súboru BasicGeneratorFrame.java.

#### javax.swing.JCheckBox swingUI.BasicGeneratorFrame.jCheckBox\_left\_intOnly [private]

Definícia na riadku 452 súboru BasicGeneratorFrame.java.

#### javax.swing.JCheckBox swingUI.BasicGeneratorFrame.jCheckBox\_showUnknows [private]

Definícia na riadku 453 súboru BasicGeneratorFrame.java.

#### javax.swing.JCheckBox swingUI.BasicGeneratorFrame.jCheckBox\_top\_saveToAFile [private]

Definícia na riadku 455 súboru BasicGeneratorFrame.java.

#### javax.swing.JCheckBox swingUI.BasicGeneratorFrame.jCheckBox\_top\_ShowHere [private]

Definícia na riadku 454 súboru BasicGeneratorFrame.java.

#### javax.swing.JEditorPane swingUI.BasicGeneratorFrame.jEditorPane [private]

Definícia na riadku 456 súboru BasicGeneratorFrame.java.

#### javax.swing.JLabel swingUI.BasicGeneratorFrame.jLabel1 [private]

Definícia na riadku 457 súboru BasicGeneratorFrame.java.

#### javax.swing.JLabel swingUI.BasicGeneratorFrame.jLabel\_pathToFile [private]

Definícia na riadku 461 súboru BasicGeneratorFrame.java.

#### javax.swing.JLabel swingUI.BasicGeneratorFrame.jLabel\_Size [private]

Definícia na riadku 460 súboru BasicGeneratorFrame.java.

#### javax.swing.JLabel swingUI.BasicGeneratorFrame.jLabelPrecision [private]

Definícia na riadku 458 súboru BasicGeneratorFrame.java.

#### javax.swing.JLabel swingUI.BasicGeneratorFrame.jLabelRange [private]

Definícia na riadku 459 súboru BasicGeneratorFrame.java.

#### javax.swing.JPanel swingUI.BasicGeneratorFrame.jPanel\_left [private]

Definícia na riadku 462 súboru BasicGeneratorFrame.java.

#### javax.swing.JPanel swingUI.BasicGeneratorFrame.jPanel\_leftAdditional [private]

Definícia na riadku 463 súboru BasicGeneratorFrame.java.

#### javax.swing.JPanel swingUI.BasicGeneratorFrame.jPanel\_top [private]

Definícia na riadku 464 súboru BasicGeneratorFrame.java.

#### javax.swing.JScrollPane swingUI.BasicGeneratorFrame.jScrollPane1 [private]

Definícia na riadku 465 súboru BasicGeneratorFrame.java.

#### javax.swing.JTextField swingUI.BasicGeneratorFrame.jTextField\_pathToFile [private]

Definícia na riadku 466 súboru BasicGeneratorFrame.java.

#### javax.swing.JTextField swingUI.BasicGeneratorFrame.jTextField\_precision [private]

Definícia na riadku 467 súboru BasicGeneratorFrame.java.

#### javax.swing.JTextField swingUI.BasicGeneratorFrame.jTextField\_rangeFrom [private]

Definícia na riadku 468 súboru BasicGeneratorFrame.java.

#### javax.swing.JTextField swingUI.BasicGeneratorFrame.jTextField\_rangeTo [private]

Definícia na riadku 469 súboru BasicGeneratorFrame.java.

#### javax.swing.JTextField swingUI.BasicGeneratorFrame.jTextField\_sizeOfMatrixGenerated [private]

Definícia na riadku 470 súboru BasicGeneratorFrame.java.

#### javax.swing.JRadioButton swingUI.BasicGeneratorFrame.radioButton\_classicMatrix [private]

Definícia na riadku 471 súboru BasicGeneratorFrame.java.

#### javax.swing.JRadioButton swingUI.BasicGeneratorFrame.radioButton\_iterationableMatrix [private]

Definícia na riadku 472 súboru BasicGeneratorFrame.java.

* + 1. swingUI.BasicGeneratorFrame.OpenFileButton

Dedí od ActionListener.

Public Member Functions

* void actionPerformed (ActionEvent e)

Detailed Description

Funkcia: táto trieda reprezentuje okno pre lokalizáciu súboru v súborovom systéme používateľa.

##### Author:

Pavol Dano

Definícia na riadku 663 súboru BasicGeneratorFrame.java.

Member Function Documentation

#### void swingUI.BasicGeneratorFrame.OpenFileButton.actionPerformed (ActionEvent *e*)

Definícia na riadku 665 súboru BasicGeneratorFrame.java.

* + 1. swingUI.BasicSolverFrame

Obr. Diagram dedičnosti pre swingUI.BasicSolverFrame

Classes

* class OpenFileButton

Public Member Functions

* BasicSolverFrame (App app)

Private Member Functions

* void initComponents ()
* void resetButtonActionPerformed (java.awt.event.ActionEvent evt)
* void jButton\_executeActionPerformed (java.awt.event.ActionEvent evt)
* void ApplicationMenuItemExitActionPerformed (java.awt.event.ActionEvent evt)
* void jButtonReturnActionPerformed (java.awt.event.ActionEvent evt)
* String choosePathAccorToMethodAndSource (LinearSystemsSolvable method)
* void setCalculatorIcon ()
* VectorD getResultsRoundedByUsersRequiry (VectorD results, int precision) throws NegativeValueException
* void setStepsOrTolerance (LinearSystemsSolvable method)
* void setMethodsButtons ()

Private Attributes

* App app
* javax.swing.JPanel basicFramePanelBottom
* javax.swing.JPanel basicFramePanelTop
* javax.swing.JButton jButtonReturn
* javax.swing.JButton jButton\_execute
* javax.swing.JCheckBox jCheckBox\_generatePDF
* javax.swing.JCheckBox jCheckBox\_showMatrix
* javax.swing.JCheckBox jCheckBox\_showResults
* javax.swing.JComboBox jComboBox\_methods
* javax.swing.JEditorPane jEditorPane
* javax.swing.JLabel jLabelPrecision
* javax.swing.JLabel jLabelStepsCount
* javax.swing.JLabel jLabelTolerance
* javax.swing.JMenuItem jMenuItem\_Exit
* javax.swing.JMenu jMenu\_Application
* javax.swing.JScrollPane jScrollPane1
* javax.swing.JScrollPane jScrollPane2
* javax.swing.JTextArea jTextArea\_info
* javax.swing.JTextField jTextFieldTop\_path
* javax.swing.JTextField jTextField\_precision
* javax.swing.JTextField jTextField\_stepsCount
* javax.swing.JTextField jTextField\_tolerance
* javax.swing.JLabel labelTop
* javax.swing.JRadioButton radioButtonUseOwnMatrix
* javax.swing.JRadioButton radioButtonUseTestFiles
* javax.swing.JButton resetButton
* javax.swing.JButton searchButton
* javax.swing.ButtonGroup sourceButtonGroup
* javax.swing.JMenuBar topMenuBar

Detailed Description

Funkcia: rodič vytváraného rámca.

##### Author:

Pavol Dano

Definícia na riadku 46 súboru BasicSolverFrame.java.

Constructor & Destructor Documentation

#### swingUI.BasicSolverFrame.BasicSolverFrame (App *app*)

konštruktor

##### Parameters:

|  |  |
| --- | --- |
| *app* | reprezentácia bežiacej aplikácie |

Definícia na riadku 57 súboru BasicSolverFrame.java.

Member Function Documentation

#### void swingUI.BasicSolverFrame.ApplicationMenuItemExitActionPerformed (java.awt.event.ActionEvent *evt*) [private]

Aplikácia končí kliknutím používateľa na tlačidlo “Exit” v aplikačnom menu

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 396 súboru BasicSolverFrame.java.

#### String swingUI.BasicSolverFrame.choosePathAccorToMethodAndSource (LinearSystemsSolvable *method*) [private]

na základe zvolenej metódy vyberá cestu k základnému príkladu

##### Returns:

Cesta k súboru s príkladom

Definícia na riadku 445 súboru BasicSolverFrame.java.

#### VectorD swingUI.BasicSolverFrame.getResultsRoundedByUsersRequiry (VectorD *results*, int *precision*) throws NegativeValueException [private]

zaokrúhľuje prvky vektora riešenia na základe používateľských požiadavok

##### Parameters:

|  |  |
| --- | --- |
| *results* | vector, ktorého prvky budú zaokrúhlené |
| *precision* | precíznosť zaokrúhľovania |

##### Returns:

vector riešenia so zaokrúhlenými hodnotami

Definícia na riadku 471 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.initComponents () [private]

inicializuje komponenty

Reimplementované z swingUI.ParentFrame.

Definícia na riadku 70 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.jButton\_executeActionPerformed (java.awt.event.ActionEvent *evt*) [private]

vykonáva zvolenú metódu

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 315 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.jButtonReturnActionPerformed (java.awt.event.ActionEvent *evt*) [private]

rámec zaniká po kliknutí na tlačidlo return

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 406 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.resetButtonActionPerformed (java.awt.event.ActionEvent *evt*) [private]

resetovanie cesty k súboru s maticou

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 307 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.setCalculatorIcon () [private]

nastavuje ikonu kalkulačky v záhlaví rámca

Definícia na riadku 457 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.setMethodsButtons () [private]

nastavuje obsah combo boxu, t.j. načítané metódy, použitím reflexive v App.classLoader.loadMethodsClasses()

Definícia na riadku 514 súboru BasicSolverFrame.java.

#### void swingUI.BasicSolverFrame.setStepsOrTolerance (LinearSystemsSolvable *method*) [private]

nastavuje počet krokov, ktoré sa majú vykonať alebo toleranciu dodržanú počas iterácií

##### Parameters:

|  |  |
| --- | --- |
| *method* | LinearSystemsSolvable metóda, ktorej tolerancia alebo počet krokov má byť nastavená |

Definícia na riadku 489 súboru BasicSolverFrame.java.

Member Data Documentation

#### App swingUI.BasicSolverFrame.app [private]

aplikácia – modul solver

Definícia na riadku 51 súboru BasicSolverFrame.java.

#### javax.swing.JPanel swingUI.BasicSolverFrame.basicFramePanelBottom [private]

Definícia na riadku 411 súboru BasicSolverFrame.java.

#### javax.swing.JPanel swingUI.BasicSolverFrame.basicFramePanelTop [private]

Definícia na riadku 412 súboru BasicSolverFrame.java.

#### javax.swing.JButton swingUI.BasicSolverFrame.jButton\_execute [private]

Definícia na riadku 414 súboru BasicSolverFrame.java.

#### javax.swing.JButton swingUI.BasicSolverFrame.jButtonReturn [private]

Definícia na riadku 413 súboru BasicSolverFrame.java.

#### javax.swing.JCheckBox swingUI.BasicSolverFrame.jCheckBox\_generatePDF [private]

Definícia na riadku 415 súboru BasicSolverFrame.java.

#### javax.swing.JCheckBox swingUI.BasicSolverFrame.jCheckBox\_showMatrix [private]

Definícia na riadku 416 súboru BasicSolverFrame.java.

#### javax.swing.JCheckBox swingUI.BasicSolverFrame.jCheckBox\_showResults [private]

Definícia na riadku 417 súboru BasicSolverFrame.java.

#### javax.swing.JComboBox swingUI.BasicSolverFrame.jComboBox\_methods [private]

Definícia na riadku 418 súboru BasicSolverFrame.java.

#### javax.swing.JEditorPane swingUI.BasicSolverFrame.jEditorPane [private]

Definícia na riadku 419 súboru BasicSolverFrame.java.

#### javax.swing.JLabel swingUI.BasicSolverFrame.jLabelPrecision [private]

Definícia na riadku 420 súboru BasicSolverFrame.java.

#### javax.swing.JLabel swingUI.BasicSolverFrame.jLabelStepsCount [private]

Definícia na riadku 421 súboru BasicSolverFrame.java.

#### javax.swing.JLabel swingUI.BasicSolverFrame.jLabelTolerance [private]

Definícia na riadku 422 súboru BasicSolverFrame.java.

#### javax.swing.JMenu swingUI.BasicSolverFrame.jMenu\_Application [private]

Definícia na riadku 424 súboru BasicSolverFrame.java.

#### javax.swing.JMenuItem swingUI.BasicSolverFrame.jMenuItem\_Exit [private]

Definícia na riadku 423 súboru BasicSolverFrame.java.

#### javax.swing.JScrollPane swingUI.BasicSolverFrame.jScrollPane1 [private]

Definícia na riadku 425 súboru BasicSolverFrame.java.

#### javax.swing.JScrollPane swingUI.BasicSolverFrame.jScrollPane2 [private]

Definícia na riadku 426 súboru BasicSolverFrame.java.

#### javax.swing.JTextArea swingUI.BasicSolverFrame.jTextArea\_info [private]

Definícia na riadku 427 súboru BasicSolverFrame.java.

#### javax.swing.JTextField swingUI.BasicSolverFrame.jTextField\_precision [private]

Definícia na riadku 429 súboru BasicSolverFrame.java.

#### javax.swing.JTextField swingUI.BasicSolverFrame.jTextField\_stepsCount [private]

Definícia na riadku 430 súboru BasicSolverFrame.java.

#### javax.swing.JTextField swingUI.BasicSolverFrame.jTextField\_tolerance [private]

Definícia na riadku 431 súboru BasicSolverFrame.java.

#### javax.swing.JTextField swingUI.BasicSolverFrame.jTextFieldTop\_path [private]

Definícia na riadku 428 súboru BasicSolverFrame.java.

#### javax.swing.JLabel swingUI.BasicSolverFrame.labelTop [private]

Definícia na riadku 432 súboru BasicSolverFrame.java.

#### javax.swing.JRadioButton swingUI.BasicSolverFrame.radioButtonUseOwnMatrix [private]

Definícia na riadku 433 súboru BasicSolverFrame.java.

#### javax.swing.JRadioButton swingUI.BasicSolverFrame.radioButtonUseTestFiles [private]

Definícia na riadku 434 súboru BasicSolverFrame.java.

#### javax.swing.JButton swingUI.BasicSolverFrame.resetButton [private]

Definícia na riadku 435 súboru BasicSolverFrame.java.

#### javax.swing.JButton swingUI.BasicSolverFrame.searchButton [private]

Definícia na riadku 436 súboru BasicSolverFrame.java.

#### javax.swing.ButtonGroup swingUI.BasicSolverFrame.sourceButtonGroup [private]

Definícia na riadku 437 súboru BasicSolverFrame.java.

#### javax.swing.JMenuBar swingUI.BasicSolverFrame.topMenuBar [private]

Definícia na riadku 438 súboru BasicSolverFrame.java.

* + 1. swingUI.BasicSolverFrame.OpenFileButton

Dedí od ActionListener.

Public Member Functions

* void actionPerformed (ActionEvent e)

Detailed Description

Funkcia: táto trieda reprezentuje okno pre lokalizáciu súboru v súborovom systéme používateľa.

##### Author:

Pavol Dano

Definícia na riadku 538 súboru BasicSolverFrame.java.

Member Function Documentation

#### void swingUI.BasicSolverFrame.OpenFileButton.actionPerformed (ActionEvent *e*)

Definícia na riadku 540 súboru BasicSolverFrame.java.

* + 1. swingUI.ParentFrame

Obr. Diagram dedičnosti pre swingUI.ParentFrame

Public Member Functions

* ParentFrame ()
* void locateToCenter (int width, int height)
* void showDialog (String message, int type)

Private Member Functions

* void initComponents ()

Detailed Description

Funkcia: rodičovský rámec pre všekty rámce vytvárané aplikáciou.

##### Author:

Pavol Dano

Definícia na riadku 25 súboru ParentFrame.java.

Constructor & Destructor Documentation

#### swingUI.ParentFrame.ParentFrame ()

konštruktor

Definícia na riadku 28 súboru ParentFrame.java.

Member Function Documentation

#### void swingUI.ParentFrame.initComponents () [private]

inicializuje komponenty

Reimplementovaný v swingUI.BasicGeneratorFrame, swingUI.BasicSolverFrame, swingUI.AboutFrame, and swingUI.StartFrame.

Definícia na riadku 35 súboru ParentFrame.java.

#### void swingUI.ParentFrame.locateToCenter (int *width*, int *height*)

posúva rámec do stredu zobrazovacej plochy štandardného výstupu

##### Parameters:

|  |  |
| --- | --- |
| *width* | šírka okna |
| *height* | výška okna |

Definícia na riadku 54 súboru ParentFrame.java.

#### void swingUI.ParentFrame.showDialog (String *message*, int *type*)

Kratšia verzia pre volanie showDialog() na objekte JOptionPane

##### Parameters:

|  |  |
| --- | --- |
| *message* | správa, ktorá má byť zobrazená |
| *type* | typ zobrazenej správy |

Definícia na riadku 68 súboru ParentFrame.java.

* + 1. swingUI.StartFrame

Obr. Diagram dedičnosti pre swingUI.StartFrame

Public Member Functions

* StartFrame ()

Private Member Functions

* void initComponents ()
* void setTukeIcon ()
* void startFrameExitButtonActionPerformed (java.awt.event.ActionEvent evt)
* void startFrameRunButtonActionPerformed (java.awt.event.ActionEvent evt)
* void button\_generatorActionPerformed (java.awt.event.ActionEvent evt)
* void aboutButtonActionPerformed (java.awt.event.ActionEvent evt)

Private Attributes

* javax.swing.JPanel bottonPanel
* javax.swing.JButton button\_generator
* javax.swing.JButton startFrameExitButton
* javax.swing.JButton startFrameRunButton
* javax.swing.JButton btnAbout

Detailed Description

Funkcia: zobrazuje hlavné menu.

##### Author:

Pavol Dano

Definícia na riadku 31 súboru StartFrame.java.

Constructor & Destructor Documentation

#### swingUI.StartFrame.StartFrame ()

konštruktor

Definícia na riadku 36 súboru StartFrame.java.

Member Function Documentation

#### void swingUI.StartFrame.aboutButtonActionPerformed (java.awt.event.ActionEvent *evt*) [private]

Spúšťa zobrazenie informačného okna

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 184 súboru StartFrame.java.

#### void swingUI.StartFrame.button\_generatorActionPerformed (java.awt.event.ActionEvent *evt*) [private]

Spúšťa generátor

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 175 súboru StartFrame.java.

#### void swingUI.StartFrame.initComponents () [private]

inicializuje komponenty

Reimplementované z swingUI.ParentFrame.

Definícia na riadku 49 súboru StartFrame.java.

#### void swingUI.StartFrame.setTukeIcon () [private]

nastavuje "tukeIcon.png" ikonu

Definícia na riadku 145 súboru StartFrame.java.

#### void swingUI.StartFrame.startFrameExitButtonActionPerformed (java.awt.event.ActionEvent *evt*) [private]

korektne ukončuje aplikáciu

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 157 súboru StartFrame.java.

#### void swingUI.StartFrame.startFrameRunButtonActionPerformed (java.awt.event.ActionEvent *evt*) [private]

spúšťa modul pre riešenie systémov rovníc

##### Parameters:

|  |  |
| --- | --- |
| *evt* | vzniknutý ActionEvent |

Definícia na riadku 166 súboru StartFrame.java.

Member Data Documentation

#### javax.swing.JPanel swingUI.StartFrame.bottonPanel [private]

Definícia na riadku 188 súboru StartFrame.java.

#### javax.swing.JButton swingUI.StartFrame.btnAbout [private]

Definícia na riadku 192 súboru StartFrame.java.

#### javax.swing.JButton swingUI.StartFrame.button\_generator [private]

Definícia na riadku 189 súboru StartFrame.java.

#### javax.swing.JButton swingUI.StartFrame.startFrameExitButton [private]

Definícia na riadku 190 súboru StartFrame.java.

#### javax.swing.JButton swingUI.StartFrame.startFrameRunButton [private]

Definícia na riadku 191 súboru StartFrame.java.

1. Ekvivalentne označovaný aj ako stĺpcový vektor pravých strán. [↑](#footnote-ref-1)
2. Ekvivalentne označovaný aj ako vektor hľadaného riešenia. [↑](#footnote-ref-2)
3. Pozri DĚMIDOVIČ, MARON [3]. [↑](#footnote-ref-3)
4. vyplývajúc z toho, že ak , potom k matici  existuje inverzná matica . [↑](#footnote-ref-4)
5. Čo by v prípade použitia Gaussovej eliminačnej metódy znamenalo zbytočné vytváranie nových nenulových prvkov, čo by v konečnom dôsledku mohlo viesť k ďalším nepresnostiam výpočtu . [↑](#footnote-ref-5)
6. V bol vektor  volený tak, že jeho prvky boli tvorené hodnotami  z (). [↑](#footnote-ref-6)
7. Pre viac informácií pozri . [↑](#footnote-ref-7)
8. V tomto prípade ilustrujeme výpočet a uchovanie rozdielov hodnôt aproximácií len medzi predposledným a posledným krokom iterácie, samozrejme, v prípade potreby je možné priebežne uchovávať hodnoty rozdielov pri každom vykonanom kroku iterovania (platí aj pre nasledujúcu metódu). [↑](#footnote-ref-8)
9. V tomto prípade ilustrujeme výpočet a uchovanie rozdielov hodnôt aproximácií len medzi predposledným a posledným krokom iterácie, samozrejme, v prípade potreby je možné priebežne uchovávať hodnoty rozdielov pri každom vykonanom kroku iterovania. [↑](#footnote-ref-9)
10. „Plugin“ v tomto kontexte chápeme ako samostatnú metódu, ktorej úlohou je vypočítať korene sústavy rovníc prijatej na vstupe. Postup dotváranie vlastných matematických metód je bližšie popísaný v používateľskej príručke, časť . [↑](#footnote-ref-10)
11. Viac na http://www.eclipse.org/aspectj/. [↑](#footnote-ref-11)
12. Viac na http://www.eclipse.org/. [↑](#footnote-ref-12)
13. Na tento účel bol prispôsobený aj výstup generátora, ktorý je plne kompatibilný so vstupom pre modul na riešenie. [↑](#footnote-ref-13)
14. Ak používateľ generuje maticu, ale nezaškrtne políčko „Show matrix“ alebo „Show results“, výstup do HTML sa nevykoná. [↑](#footnote-ref-14)
15. Tabuľka má čisto informatívny charakter . [↑](#footnote-ref-15)
16. Všetky merania boli realizované na Intel(R) Core(TM) Duo CPU T6400, 2.00 GHz, 4GB operačnej pamäte, OS Windows Vista. [↑](#footnote-ref-16)
17. Matice, ktorých časy generovania sú uvedené v tabuľke-, boli zároveň použité pre získanie časov výpočtov, uvedených v tabuľke- GHz. Podmienky generovania/výpočtu sú dané nielen rozsahom hodnôt matíc, ale aj napríklad počtom desatinných miest braných do úvahy. Preto, je pravdepodobné, že pri pokuse zopakovať podobné merania, získané výsledky budú čiastočne odlišné od uvedených v tabuľke. [↑](#footnote-ref-17)
18. V prípade potreby riešenia matíc náročných na pamäťové miesto (kde rádovo *n* sa udáva v stovkách, tisíckach) je potrebné špecifikovať pamäťové nároky pomocou prepínača -Xms –Xmx.   
    Napríklad: java –Xms32m –Xmx512m -classpath classes/.;lib/aspectjrt.jar;lib/fop.jar core.App  
    pri spustení aplikácie pripraví pamäťové miesto o veľkosti 32 MB a dokáže ho rozšíriť v prípade potreby na požadovaných 512 MB. [↑](#footnote-ref-18)
19. V tomto prípade sa predchádzajúca požiadavka ignoruje. [↑](#footnote-ref-19)
20. Neplatí pre absolútne členy . [↑](#footnote-ref-20)
21. Poprípade aj v domovskom adresári používateľa, a to v adresári LinearSystemsSolver. Jedná sa o súbor s koncovkou „.html“, ktorý je zobraziteľný webovým prehliadačom. [↑](#footnote-ref-21)
22. Riešenie, samozrejme, je vypočítateľné podprogramom, o ktorý budeme popisovať v nasledujúcej kapitole. [↑](#footnote-ref-22)
23. Neplatí pre obsah súboru, keďže doň sa hodnoty neznámych zapíšu vždy. [↑](#footnote-ref-23)
24. Výpočet sa bude riadiť na základe zvolenej podmienky pre dosiahnutie určitej presnosti len v prípade, že položka „Steps count“ ostane prázdna (nebude obsahovať žiadne číslo). [↑](#footnote-ref-24)
25. Popis je realizovaný v anglickom jazyku. Súbor má názov „result.pdf“. Štruktúru generovaného PDF súboru pred jeho konverziou je možné zhliadnuť v príslušnom FO súbore. [↑](#footnote-ref-25)
26. Počas ukladania do súboru alebo aj počas zobrazenie matice/výsledku do aplikačného okna. [↑](#footnote-ref-26)
27. Domovská stránka : http://www.eclipse.org/ [↑](#footnote-ref-27)
28. Mohli by sme deklarovať princíp vytvárania a dodávania nových metód použitím textových súborov, ale takto to považujeme za prehľadnejšie. [↑](#footnote-ref-28)
29. Tým sa v súborovom systéme vytvoria naraz oba priečinky a ušetríme vykonanie jedného kroku. [↑](#footnote-ref-29)
30. a prekompilovaného (čo robí prostredie Eclipse pri uložení zmien (Ctrl+S) automaticky) [↑](#footnote-ref-30)
31. Odporúčame prepísať toString nasledovne: public String toString() { return getName(); }. [↑](#footnote-ref-31)
32. Myslíme tým návratovú hodnotu metódy getResult(). [↑](#footnote-ref-32)
33. Použitý Java Development Kit JDK 1.6. [↑](#footnote-ref-33)
34. Viac na http://www.eclipse.org/. [↑](#footnote-ref-34)