Univerzita Hradec Králové

Přírodovědecká fakulta

BAKALÁŘSKÁ PRÁCE

2021 **Alex El Gharred**

Univerzita Hradec Králové

Přírodovědecká fakulta

Katedra Kybernetiky

MS Excel jako nástroj řešení úloh numerických metod

Bakalářská práce

Autor: Alex El Gharred

Studijní program: B1101 Matematika

Studijní obor: BIN/BMAT

Vedoucí práce: Hubálovský Štěpán, doc. RNDr. Ph.D.

Hradec Králové březen 2021

Univerzita Hradec Králové
Přírodovědecká fakulta

Zadání bakalářské práce

Autor:	**Alex El Gharred**
Studium:	S18MA028BP
Studijní program:	B1101 Matematika
Studijní obor:	Informatika se zaměřením na vzdělávání, Matematika se zaměřením na vzdělávání
Název bakalářské práce:	**MS Excel jako nástroj řešení úloh numerických metod**
Název bakalářské práce AJ:	MS Excel as toll for solution of numerical method exercises

Cíl, metody, literatura, předpoklady:

Cílem bakalářské práce je vytvoření vzorových příkladů z oblasti numerických metod řešených v tabulkové procesoru MS Excel. V rámci teoretické části práce bude formou literární rešerše vybrána sada vzorových příkladů vhodných pro řešení v tabulkovém procesoru. V rámci praktické části práce bude realizováno řešení vzorových příkladů v rámci MS Excel.

xx

Garantující pracoviště:	Katedra aplikované kybernetiky, Přírodovědecká fakulta
Vedoucí práce:	doc. RNDr. Štěpán Hubálovský, Ph.D.
Oponent:	Ing. Petr Voborník, Ph.D.

Datum zadání závěrečné práce: 9.4.2019

Prohlášení

Prohlašuji, že tato bakalářská práce byla vypracována mnou, samostatně a že v seznamu použité literatury uvádím všechny prameny, z nichž práce vychází.

V Hradci Králové dne 19. března 2021 Alex El Gharred

Anotace

EL GHARRED, Alex. *MS Excel jako nástroj řešení úloh numerických metod*. Hradec Králové, 2021. Bakalářská práce na Přírodovědecké fakultě Univerzity Hradec Králové. Vedoucí bakalářské práce Štěpán Hubálovský. 38s.

Bakalářská práce se zabývá možností využití kancelářského softwaru MS Excel k řešení matematických příkladů s pomocí numerických metod. V práci jsou vybrány konkrétní numerické metody řešení soustavy lineárních rovnic, které jsou ve stručnosti představeny, algoritmizovány a následně implementovány v prostředí MS Excel. Tyto metody jsou též za účelem srovnání algoritmizovány pro prostředí programovacího jazyka. Postup je v obou případech podrobně rozebrán a vysvětlen. V práci jsou též shrnuty výhody a nevýhody jednotlivých prostředí.

Klíčová slova

algoritmus, MS Excel, numerické metody, soustava lineárních rovnic

Annotation

EL GHARRED, Alex. *MS Excel as a tool for solving problems of numerical methods*. Hradec Králové, 2021. Bachelor thesis at Faculty of Science, University of Hradec Králové. Thesis supervisor Štěpán Hubálovský. 38s.

This bachelor thesis deals with the possibility of using the office software MS Excel to solve mathematical examples using numerical methods. The work selects specific numerical methods for solving linear equations systems, which are briefly introduced, algorithmized and then implemented in MS Excel. These methods are also algorithmized for programming language environment for comparison. In both cases, the procedure is discussed and explained in detail. The work also summarizes the advantages and disadvantages of each environment.

Keywords

algorithm, MS Excel, numerical methods, linear equations system

Obsah

Úvod

Tato bakalářská práce má za cíl seznámit čtenáře s možnostmi využití numerických metod pro řešení matematických příkladů s pomocí kancelářského softwaru MS Excel. Pro srovnání uvádím i implementaci v prostředí programovacího jazyka a v zájmu lepšího pochopení samotné práce v MS Excel popisuji i algoritmizaci přímo pro tento software.

Motivací ke zvolení tohoto tématu mi byl fakt, že numerické metody bývají často vyučovány za pomoci vysoce specializovaných softwarů, které jsou sice vhodné pro odborníky v dané oblasti, nicméně studenty, kteří se s tématem numerických výpočtů seznámí pouze velmi povrchově, mohou tato specializovaná prostředí, která navíc často postrádají lokalizaci do českého jazyka, poněkud mást.

MS Excel je naproti tomu snadno dostupnou alternativou, s níž bývají studenti většinou již seznámeni z nižších stupňů. Tento kancelářský software navíc nabízí i relativně snadnou implementaci, kterou se v této práci budu snažit přiblížit tak, aby jí porozuměli nejen vyučující, ale i samotní studenti.

Práce je rozdělena na stručnou teoretickou část, v níž čtenáře obeznámím se samotným pojmem numerických výpočtů, jejich využitím ve výuce a různými vhodnými softwary. V praktické části pak představím konkrétní metody řešení soustavy čtyř lineárních rovnic o čtyřech neznámých. U každé metody nejprve představím její matematický princip, který by měl pomoci pochopit samotný mechanismus, který budu později využívat.

Dále pak v jednotlivých krocích popíšu algoritmizaci pro MS Excel, jež by měla přiblížit logické kroky využité v pozdější implementaci. Pro srovnání využití stejného mechanismu v rozdílném prostředí popisuji též algoritmizaci za pomoci programovacího jazyka, které by též mohli využít studenti informatiky, kteří již měli možnost se programováním zabývat. Nakonec čtenáře seznámím se samotnou implementací v prostředí MS Excel, včetně doporučení na konkrétní rozložení daných funkcí v tabulce a také doporučené formátování pro co nejlepší vizuální dojem.

Mým cílem je ukázat, že numerické metody je možné využít i v méně specializovaných prostředích, v nichž se navíc studenti již orientují, díky čemuž odpadne nutnost seznamování s novým, neznámým prostředím.

1 Úvod do problematiky

1.1 Numerické metody

Nejdříve by bylo vhodné objasnit, co to znamená numerická matematika. Je to vědní disciplína, která se snaží vyvíjet a analyzovat metody, jež využívají manipulaci s čísly. Numerická metoda je pak postup, kterým se využívá k řešení numerické úlohy. Numerické metody se v praxi využívají k přibližnému výpočtu (aproximaci) matematických modelů, jejichž analytické řešení by bylo buď to nemožné nebo neúměrně obtížné. [1]

v případě, že řešíme problémy reálného světa, velmi často se stane, že se setkáme s nutností popsat zkoumaný problém pomocí věrohodného matematického modelu, který potom musíme správně vyřešit. Tyto matematické modely bývají komplikované, mnohdy popsané za pomoci diferenciálních či algebraických rovnic. Výhoda dnešní doby je, že máme výkonné počítače, které problém dokáží velmi rychle vyřešit. [1]

Další pojem, který musíme vysvětlit, je algoritmus numerické metody. Jedná se o přesný popis kroků, s jejichž pomocí je realizován výpočet numerické úlohy. Tento popis lze vyjádřit jako posloupnost akcí, které k přesně specifikovanému souboru vstupních dat přiřadí příslušný soubor dat výstupních. Příprava objemných souborů vstupních dat je označována jako předzpracování (preprocessing). Výsledné údaje mohou být velkého rozsahu, což pro člověka představuje problém, proto se výsledky musí upravit tak, aby je uživatel vůbec mohl vyhodnotit. Těmto metodám se říká následné zpracování (postprocessing). Příkladem následného zpracování může být vizualizace výsledku. [1]

Úlohy, jež jsou jednoznačně řešitelné, a jejich řešení závisí na vstupních datech, se nazývají korektní úlohy. V praxi se ovšem často stane, že i korektní úloha vykazuje velké chyby ve výsledcích, a to i při zanedbatelné změně vstupních dat. Takové úlohy se nazývají špatně podmíněné a jejich řešení je velmi problematické, v případě, že jejich vstupní data obsahují chyby (např. chyby měřících zařízení, chyby v zaokrouhlování a pod.). Úloha je dobře podmíněná tehdy, když má malá změna ve vstupních datech za následek pouze malou změnu v řešení. [2] [3]

Numerického modelu se využívá například při předpovědi počasí po celém světě. Je možné tak nejen předpovědět, zda bude v příštích dnech slunečno, zataženo či dokonce deštivo, ale též například vypočítat trasu hurikánu a včas tak varovat před jeho ničivými následky. Numerické modely se v meteorologii rozlišují na globální a lokální, přičemž výsledky globálních modelů navazují na modely lokální. Při interpretaci dat je však třeba brát v potaz, že podoba predikce numerického modelu atmosféry je zjednodušená a může proto docházet k nepřesnostem, což je také důvodem, proč se předpověď počasí někdy zmýlí. [4]

Pro výpočet různých standardních úloh máme k dispozici již vytvořené programy či balíčky programů. Pro jejich správné použití je však třeba se orientovat v numerických metodách, které hodláme používat, neboť nekvalifikované užívání těchto programů by mohlo vést k nepřesným, nebo dokonce chybným výpočtům. [5]

1.1.1 Chyby při numerických výpočtech

Získat přesné výsledky při řešení reálných problémů je velmi obtížné, téměř vždy máme řešení pouze přibližné, které obsahuje určitou chybu. Musíme se při naší práci zaměřit na to, aby byl výpočet co nejvíce přesný a celková chyba co nejmenší. Chyby totiž mohou ovlivnit, případně kompletně znehodnotit získaný výsledek. Důležitým faktorem při této práci jsou lidské chyby, které mohou vzniknout tím, že člověk nepochopí daný problém. U lidí musíme počítat také s nepozorností nebo nedbalostí. [1]

Vytváříme-li matematický model skutečného problému, pokaždé provádíme určité idealizace. Pokud se vyskytnou chyby ve vstupních údajích nebo idealizace daného modelu, samozřejmě to ovlivní výsledek a dojde k rozdílu mezi řešením idealizovaného problému a řešením problému reálného, který se pak nazývá chyba matematického modelu. [1]

Existuje i chyba numerické metody, ke které dochází, pokud k řešení numerické úlohy použijeme metodu, které neposkytne přesné řešení dané úlohy. Důležité je provést odhad chyby numerické metody a to již při návrhu numerické metody, kterou se chystáme použít. [1]

Vzhledem k tomu, že při práci na počítači můžeme využít pouze proměnné s konečným počtem cifer, pracujeme pouze s přibližnými hodnotami čísel, jež získáme pomocí zaokrouhlení přesných hodnot. Tyto chyby se nazývají zaokrouhlovací chyby a vznikají již během vkládání dat do počítače, další chyby pak vznikají při číselných výpočtech. [1]

1.1.2 Numerické metody ve výuce

Na univerzitách, kde se vyučují obory, které potřebují znalosti z numerických výpočtů, se obvykle vyučují tradiční témata z této problematiky, jako jsou zdroje chyb a jejich řešení, hledání kořenů nelineárních rovnic, řešení soustav lineárních rovnic, aproximace funkcí, numerické výpočty derivací a integrálů a tak dále. Pro účast na předmětech věnovaných numerickým metodám se předpokládá základní znalost témat lineární algebry a diferenciálního a integrálního počtu funkcí jedné proměnné. [1] [3]

Ačkoliv jsou klasické metody mnohdy dávno překonané a nahrazené efektivnějšími postupy, tak se ve školách stále vyučují, neboť moderní algoritmy bývají často poměrně komplikované a složité pro porozumění. Vždy je ve výuce numerických metod vhodné danou metodu napřed zdůvodnit, aby studenti pochopili její princip, netřeba však usilovat o formálně dokonalé zdůvodnění, neboť by to na porozumění studentů mohlo působit kontraproduktivně. Řešené příklady je vhodné volit tak, aby díky nim bylo možné pochopit a následně i použít někdy poněkud špatně srozumitelně řečené formule a postupy. Také je více než žádoucí dát studentům prostor k samostatnému procvičení příkladů, protože to je nejlepší způsob, jak se danou látku člověk naučí, a také má skvělou příležitost zjistit, co mu ještě dělá potíže, a poté se na to zeptat vyučujícího. [1]

Pro někoho může být problémem i to, že není k dispozici moderní česky psaná monografie numerických metod a pro některé studenty může jazyková bariéra představovat

velkou překážku. [1] Na univerzitách se studentům většinou dostává pouze seznámení se základní orientací v numerických metodách, ale i přesto by jim to mělo umožnit, aby mohli kvalifikovaně používat velké množství programů, což se může hodit v profesním životě. [1]

Existují různé prostředky pro výuku numerických metod, mezi než patří např.: klasická prezenční přednáška, webová přednáška, samostatné webové samostudium a webové samostudium kombinované s následnou diskusí ve třídě. Během posledních let proběhlo značné množství výzkumů ohledně možností zlepšení výuky studentů napříč obory, včetně přírodních věd, matematiky, strojírenství a informatiky. Právě pro tyto obory existuje značné využití numerických metod v praxi. [6]

Mimo jiné se zkoumaly i různé možnosti distanční výuky, která má jedinečnou schopnost studentům poskytnout různé způsoby učení, využívaje různých zdrojů a strategií, čímž pokrývá rozmanité zájmy a potřeby jednotlivých studentů. [6]

Je důležité nezapomínat, že stejně jako u ostatních disciplín, i u výuky numerických metod je zapotřebí přizpůsobit úroveň a složitost výuky dosavadní úrovni znalostí studentů. Hlavním cílem výuky je připravit studenty na flexibilní adaptaci na nové problémy, s nimiž se setkají v profesním životě. [6]

K maximalizaci pravděpodobnosti trvalého a flexibilního přenosu znalostí a schopností obsažených v kurzech věnovaným numerickým metodám je vhodné použít interaktivní výukové moduly, aby se tak studentům krom pasivní konzumace informací dostalo též příležitosti nabyté znalosti vyzkoušet na praktickém problému. [6]

1.1.3 Nástroje

Za posledních několik let byla díky rozmachu programování vyvinuta celá řada specializovaných programů pro různé oblasti zaměření, mezi něž se řadí i matematika. Mezi specializované programy se řadí například Mathematica, Maple a Matlab, které jsou vybaveny jednoduchým programovacím jazykem i funkcemi umožňujícími práci s matematickými operacemi všech možných druhů, jako např. derivování, integrování, algebraické úpravy a v neposlední řadě řešení úloh pomocí odpovídajících numerických metod. [5]

Program Mathematica od firmy Wolfram Research, jehož vývoj probíhá již po tři dekády, představuje světově nejznámější programový systém pro využití numerických i analytických výpočtů a vizualizaci dat. Tento program je snadno přístupný na webu, použitelný prostřednictvím libovolného webového prohlížeče, stejně tak je dostupný v desktopové formě pro všechny moderní operační systémy. Tento program se řadí mezi komerční, přičemž nabízí za jednorázovou částku stažení desktopové aplikace, či online verzi, kde je nutné licenci obnovovat na měsíční či roční bázi. Výrobce též nabízí zdarma patnáctidenní zkušební verzi. Nevýhoda programu Mathematica je absence české lokalizace. [7] [8]

Dalším specializovaným programem je Maple od české společnosti Czech Software First s.r.o. Maple je počítačový software pro výuku a další využití matematiky v různých oblastech vědy. Jeho vývoj sahá do devadesátých let minulého století, software podporuje

analytické i numerické výpočty a jejich vizualizaci, dokumentaci i publikaci, přičemž poskytuje přívětivé uživatelské prostředí. Firma nabízí různé druhy licencí, mezi něž patří akademická, profesionální, studentská a vládní. Nabízí též patnáctidenní zkušební verzi zdarma, pro studenty však nyní nabízí zkušební verzi až v délce 30 dní. Tento software také podporuje českou lokalizaci. [9]

Nástroj MATLAB od firmy MathWorks je vhodným prostředím pro vědecké a technické výpočty, analýzu a vizualizaci dat i vývoj algoritmů, jenž je využíván vědci a inženýry po celém světě. Stejně jako předchozí softwary umožňuje řešit různé úlohy a problémy napříč obory, jako je například aplikovaná matematika nebo strojové učení. Nabízí výkonný programovací jazyk pro vývoj algoritmů a numerické výpočty a tisíce vestavěných funkcí nejen z oblasti matematiky. Stejně jako u předchozích dvou se jedná o komerční software s různými druhy licencí, který nabízí třicetidenní zkušební verzi. Stejně jako program Mathematica, ani MATLAB nepodporuje českou lokalizaci. [10] [11]

Pro numerické výpočty se však dají využít i méně specializovaná prostředí, jako například tabulkový kalkulátor MS Excel, který umožňuje práci s daty, tabulkami a disponuje množstvím dalších užitečných funkcí. Excel patří mezi základní softwary kancelářského balíčku Microsoft Office, jehož existuje hned několik verzí. Nejnovější je MS Office 365, jenž má na rozdíl od předchozích verzí časově omezenou licenci, kterou je potřeba obnovovat. Umožňuje však sdílené využití produktu například pro firemní kolektiv nebo školní třídu. Ze starších verzí stojí za zmínku MS Office 2003, jenž je poslední verzí s klasickým uživatelským prostředím, jenž bylo od verze 2007 kompletně změněno. MS Excel podporuje kompletní českou lokalizaci a nabízí třicetidenní zkušební verzi. Existuje i open source varianta Excelu, známá jako Libre Office Calc, jejíž uživatelské prostředí se podobá klasickému vzhledu Excelu. Jelikož se nejedná o specializovaný software, je třeba být při vývoji postupů pro výpočty numerických metod vynalézavý a vzít v úvahu všechny eventuality. [17]

Další možností je využít k vývoji algoritmů pro numerické metody využít přímo programovací jazyk, což na jednu stranu nabízí značnou volnost, na stranu druhou je třeba algoritmické myšlení a především správné ošetření všech výjimek, neboť pokud nějaká část programu vrátí chybu, naprogramované řešení se může zaseknout či přímo spadnout. Další výhodou přímého programování je, že se není nutné vázat na konkrétní software, neboť vývojářských nástrojů existuje celá řada, z nich spousta podporuje českou lokalizaci a lze snadno nalézt nekomerční řešení. Například známý, velice univerzální nástroj Visual Studio nabízí verzi Community, jež je kompletně bezplatná.

1.2 Definice pojmů

1.2.1 Matematika

Lineární rovnice je taková rovnice, která obsahuje jedinou neznámou, a to pouze v první mocnině. Je možné ji upravit na tzv. základní tvar, jenž vypadá takto: $ax + b = 0$, kde x

je neznámá a symboly a a b jsou libovolná reálná čísla. Výraz ax se nazývá lineární člen, výraz b pak absolutní člen. [13]

Soustava lineárních rovnic se dá definovat jako množina m lineárních rovnic o n neznámých. Jejím vyřešením získáme hodnoty všech n neznámých, které když dosadíme do původní soustavy, všechny její rovnice dávají smysl. Stejně jako klasická lineární rovnice může mít i soustava rovnic různý počet řešení – jedno, žádné a nebo nekonečně mnoho. Nicméně v případě soustavy lineárních rovnic je otázka nalezení řešení a jejich počtu poněkud složitější, než v případě jednoduché rovnice. [13]

Obecně vyjádřená soustava lineárních rovnic vypadá takto:

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{array}$$

Kde x_j $(j \in \{1, 2, ..., n\})$ jsou neznámé, výrazy a_{ij} a b_i $(i, j \in \{1, 2, ..., n\})$ jsou pak libovolná reálná čísla.

v této bakalářské práci však budeme pracovat se soustavami čtyř lineárních rovnic o čtyřech neznámých, značených následovně:

$$\begin{array}{c} a_{11}w + a_{12}x + a_{13}y + a_{14}z = b_1 \\ a_{21}w + a_{22}x + a_{23}y + a_{24}z = b_2 \\ a_{31}w + a_{32}x + a_{33}y + a_{34}z = b_3 \\ a_{41}w + a_{42}x + a_{43}y + a_{44}z = b_4 \end{array}$$

Kde w, x, y a z jsou neznámé a výrazy a_{ij}, b_i libovolná reálná čísla.

Matice je schéma čtvercového nebo obdélníkového tvaru, obsahující číselné hodnoty, které indexujeme a_{ij} $(i \in \{1, 2, ..., n\}$ a $j \in \{1, 2, ..., m\})$. V případě matice soustavy lineárních rovnic v naprosté většině případů platí, že m = n. Sloupec pravých stran (absolutních členů) rovnic soustavy indexujeme b_i $(i \in \{1, 2, ..., n\})$ a od hodnot lineárních členů ho oddělujeme svislou čarou. [13] Máme-li výše zapsanou soustavu čtyř rovnic o čtyřech neznámých, její maticové vyjádření bude vypadat následovně:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ a_{11} & a_{12} & a_{13} & a_{14} & b_1 \end{array}\right)$$

Ekvivalentní úpravy neboli řádkové úpravy soustavy rovnic jsou takové typy úprav, které nezmění platnost soustavy. Jejich smyslem je zjednodušit soustavu do tvaru, z něhož je možné snadno dostat výsledek.

Mezi ekvivalentní úpravy se řadí:

1. Vzájemná výměna libovolných řádků
2. Vynásobení jedné rovnice libovolným reálným číslem k, $k \neq 0$

3. Přičtení k-násobku jedné rovnice soustavy k jiné rovnici soustavy

[13]

Hlavní diagonála matice je určena členy a_{ij}, kde $i = j$.

Trojúhelníkový tvar matice, konkrétně horní trojúhelníkový tvar, se vyznačuje tím, že všechny hodnoty a_{ij} pod hlavní diagonálou jsou rovny nule.

Lineární kombinace n řádků či vektorů x_1 až x_n se značí $v = k_1x_1+k_2x_2+...+k_nx_n$, kde k_i $(i \in \{1, 2, ..., n\})$ je libovolné reálné číslo.

Lineárně nezávislé řádky nebo vektory se vyznačují tím, že žádný z nich nemůže být získán lineární kombinací ostatních.

Hodnost matice je počet lineárně nezávislých řádků této matice.

Řídící prvek řádku je první prvek zleva, který má nenulový koeficient.

Jednotková matice je čtvercová matice, na jejíž hlavní diagonále se nachází jedničky, zatímco na ostatních místech nuly.

Redukovaný trojúhelníkový tvar matice je odvozen od trojúhelníkového tvaru, nicméně každý řídící prvek řádku je zde rovný jedné a pro každý sloupec platí, že pokud je v něm zastoupen řídící prvek některého řádku, všechny ostatní prvky v tomto sloupci jsou nulové.

Absolutní hodnota z čísla je vždy číslo větší nebo rovné nule a jeho hodnota se rovná vzdálenosti daného čísla od nuly. Pokud je dané číslo x kladné, jeho absolutní hodnota bude totéž číslo x, pokud jde však číslo x záporné, jeho absolutní hodnota bude číslo opačné, tedy $-x$. Absolutní hodnota z čísla x se značí $|x|$. [13]

Determinant matice je zobrazení v lineární algebře, jenž každé číselné čtvercové matici A přiřadí číslo, které označujeme $|A|$ nebo $det\ A$. Permutace množiny obsahující n prvků je uspořádaná n-tice, která každý z těchto prvků obsahuje právě jednou. Počet všech možných permutací množiny o n prvcích je určen vztahem $P(n) = n!$, kde $n!$ znamená faktoriál, tedy součin všech přirozených čísel menších nebo rovných číslu n. [13]

1.2.2 Informatika

Proměnná je objekt, jehož hodnota se může za běhu programu měnit. Zjednodušeně řečeno se jedná o místo v paměti, kam si můžeme ukládat data a potom s nimi pracovat. Proměnná se pojmenovává libovolným názvem bez mezer a diakritiky, obvykle by název měl začínat písmenem. V některých jazycích (např. PHP nebo Javascript) se před název proměnné píše znak $ (dolar), neoddělený mezerou. [14]

Bit je nejmenší jednotka kapacity paměti, která může nabývat pouze dvou hodnot, a to `1` a `0`. Obě hodnoty je možné interpretovat jako logické hodnoty pravda/nepravda, případně zapnuto/vypnuto. Značí se malým písmenem *b*, případně přímo *bit*.

Byte je základní jednotka počítačové paměti a objemu počítačových dat. Označuje jednotku osmi bitů, které tak tvoří osmiciferné binární číslo v rozmezí 0 až 255. Obvykle se jedná o nejmenší objem dat, se kterým dokáže počítač najednou pracovat. Zpravidla se značí velkým písmenem *B*.

Datový typ určuje druh a rozsah hodnot, kterých proměnná nabývá a určuje též možný způsob jejího využití. Specifikuje vlastnosti, jako např. velikost a strukturu proměnné. Každý datový typ má též definováno, jaké operace s ním je a není možné provádět, kupříkladu že od sebe nelze odečíst dvě proměnné obsahující řetězec znaků. Datové typy se dělí na jednoduché a složené, přičemž složené datové typy vznikají skládáním jednoduchých datových typů. Mezi složené datové typy se řadí například `pole`. Různé jazyky mohou mít různé a různě pojmenované datové typy, ačkoliv některé z nich má většina jazyků společné. Mezi základní jednoduché datové typy se řadí logická hodnota (`bool`, případně `boolean`), čili binární rozhodnutí, zda je určitý výraz či možnost pravda nebo nepravda. Tento datový typ má velikost pouze jeden bit, ačkoliv bývá obvykle ukládán jako jeden byte. Dalším důležitým datovým typem je celé číslo, dále se dělí v závislosti na velikosti a tedy rozsahu hodnot. Nejpoužívanější typ `int` bývá většinou čtyřbytový, existují však i menší a větší varianty. Číselné typy se dělí též podle toho, zda mohou obsahovat znaménko (signed) či nikoliv (unsigned). Číselné datové typy se znaménkem mají poloviční rozsah hodnot nad nulou, neboť musí rezervovat jeden bit pro znaménko. Dále jsou hojně užívané datové typy obsahující desetinná čísla (`float, double, long double, decimal` a další), které se liší v závislosti na přesnosti, tedy počtu desetinných míst. Speciální datový typ, obvykle značený jako `char`, uchovává v paměti právě jeden znak. Poslední jednoduchý datový typ, který stojí za zmínku, je řetězec znaků, běžně označovaný jako `string`. Ačkoliv je jedná o jednoduchý datový typ, v mnoha ohledech se chová jako pole hodnot typu `char`.

Staticky typované programovací jazyky se vyznačují tím, že u každé deklarované proměnné striktně vyžadují, aby byl uveden její datový typ, a tento typ je dále neměnný. V každém okamžiku je tak jasné, jakého datového typu je proměnná, s níž se právě pracuje. Programy vygenerované se statickou typovou kontrolou bývají velmi bezpečné. Její velká výhoda tkví též v tom, že případné typové chyby odhalí již při kompilaci a tudíž odpadá nutnost opakovat typovou kontrolu při každém spuštění programu, což zvyšuje efektivitu jeho chodu – program může běžet rychleji nebo je méně náročný na paměť. Mezi staticky typované jazyky patří C, C++, C# nebo Java. [14] [15]

Dynamicky typované programovací jazyky nepožadují uvedení datového typu u deklarovaných proměnných, mohou tedy odkazovat na hodnotu jakéhokoliv typu. Při přiřazení hodnoty proměnné jí jazyk sám přiřadí typ na základě druhu vložené hodnoty. Pokud je hodnota proměnné přepsána, jazyk opět vyhodnotí správný typ. Výhodou dynamické typové kontroly je rychlejší vývoj díky menšímu množství kódu. Některé jazyky (např. PHP) mezi typy dokonce samy převádí, pokud jsou použity v operaci, která to vyžaduje. Mezi zástupce dynamicky typovaných jazyků patří PHP, Python nebo Ruby. [14] [15]

Index je celé číslo, které odpovídá pořadí prvku v poli. Různé programovací jazyky se liší v tom, jakým indexem označují první prvek v poli. Nejtypičtější je indexování počínající nulou, které používá většina programovacích jazyků (např. C, C++, C#, Java, PHP, Python a další) a index vynásobený velikostí prvku v bytech vyjadřuje posunutí příslušného prvku v paměti od počátku pole. Některé jazyky (např. BASIC) indexují od

jedničky, což odpovídá intuitivnímu pohledu na počty a též matematickému značení. Jiné jazyky (např. Visual Basic nebo Pascal) umožňují nastavit horní i dolní mez pole individuálně. Máme-li pole nazvané pole, k prvku na indexu i ve většině jazyků přistupujeme zápisem `pole[i]`. [14]

Pole patří mezi základní prvky vyšších programovacích jazyků, jedná se o datovou strukturu sdružující daný počet prvků, který je vždy konečný. Matematickou obdobou pole je množina. V závislosti na konkrétním programovacím jazyce je počet prvků pole buď to nutné deklarovat předem a je neměnný (např. C#), nebo je možné prvky průběžně mazat a přidávat (např. PHP nebo Python). Pokud se jedná o dynamicky typovaný jazyk, pole může většinou obsahovat prvky různých datových typů, a to včetně polí samotných – bavíme se tak o vícerozměrných polích. Staticky typované jazyky poli typicky přiřazují konkrétní datový typ, který je pak jediným datovým typem, jenž může pole obsahovat. K jednotlivým prvkům pole se přistupuje pomocí jejich indexu. [14]

Vícerozměrná pole se od polí jednorozměrných liší tím, že zatímco jednorozměrné pole je možné chápat jako řádek hodnot, čili uspořádání hodnot má pouze jeden rozměr, jehož hodnotu vyjadřuje index, vícerozměrná pole, jak již název napovídá, používají pro indexování více rozměrů, jejichž hodnoty se zapisují do uspořádaných n-tic, jako např. `[1, 5, 2]` – tato uspořádaná trojice značí trojrozměrné pole. Dvourozměrné pole, které bude pro tuto práci klíčové, pak představuje tabulku nebo matici. Některé jazyky nepodporují vícerozměrná pole přímo, ale reprezentují je s pomocí tzv. pole polí. Např. pokud bychom chtěli vyjádřit dvourozměrné pole, jednalo by se o pole, jehož prvky by představovaly řádky či sloupce, tedy jednorozměrná pole, jenž by pak obsahovala jednotlivé hodnoty těchto řádků či sloupců. Namísto uspořádaných n-tic se souřadnice v poli polí zapisuje každá zvlášť v hranatých závorkách, např. takto: `[1] [5] [2]`. [14]

Podmínky neboli větvení, či podmíněný příkaz patří mezi základní řídící struktury programu. Jedná se o prostředek programovacího jazyka, který umožňuje rozdílné chování programu. V závislosti na splnění určité zadané podmínky, která je vyhodnocena jako pravda nebo nepravda, buď to provede určitý příkaz nebo neprovede nic. V naprosté většině programovacích jazyků se podmínka zapisuje klíčovým slovem `if`. Je možné použít příkaz označený klíčovým slovem `else`, který se provede, pokud podmínka splněna nebyla. Pro zjednodušení budu v této práci používat české varianty když a jinak. [14]

Cykly, stejně jako podmínky, patří mezi základní řídící struktury programu. Využívají se k opakování určité posloupnosti příkazů v závislosti na ukončovaní podmínce (`while` cykly) nebo předem zadaném počtu opakování (`for` cykly). Speciálním případem je cyklus `foreach`, který provede zadané příkazy pro každý prvek jednorozměrného pole. Cyklus `foreach` je specifický v tom, že nepracuje se samotným polem, ale s jeho kopií, a tak s jeho pomocí nelze prvky pole přepisovat. Hodí se však například pokud chceme všechny prvky pole zobrazit, nebo vypočítat hodnotu proměnné, která se v poli nenachází. Pro předčasné ukončení cyklu se v některých programovacích jazycích dá použít příkaz `break`, který by se do češtiny dal přeložit jako přerušit. [16]

Příkaz `return`, česky návrat, je klíčové slovo, které slouží pro návrat z metody, v níž bylo použito. V některých programovacích jazycích může `return` vrátit hodnotu libovolného typu. Pokud se jedná o staticky typovaný jazyk, je třeba typ návratové hodnoty předem definovat.

1.2.3 MS Excel

Sešit v MS Excel 2003 se standardně skládá ze tří listů, které je možné mazat či přidávat nové. Jednotlivé listy se pak skládají z buněk.

Buňky jsou definovány sloupci, které se značí pomocí písmen, a řádky, které se značí pomocí čísel. Do každé buňky lze umístit hodnotu různého typu, např. číslo, text nebo datum, či vzorec. Každou buňku zvlášť lze též libovolně naformátovat. [17]

Vzorec v buňce začíná znaménkem = a provádí různé operace, jak matematické, tak např. statistické, finanční, textové nebo logické. Vzorec může krom správného výpočtu vracet i chybovou hodnotu, a to v případě, že je prováděna operace s neplatnými daty, jako např. dělení nulou nebo matematické operace s textovými řetězci. [17] [18]

Formát zahrnuje barvu a styl písma, barvu pozadí, barvu, styl a tloušťku ohraničení, atd. Formát může být zadán ručně, což bývá nejčastější, pak také ale existuje formátování automatické, kde se zvolí konkrétní přednastavené schéma přímo od Excelu, a v neposlední řadě formátování podmíněné, které budeme využívat k zvýraznění výsledku v závislosti na dosažení určité přesnosti. [17]

Automatické vyplňování buněk je funkce, která umožňuje zkopírovat obsah jedné buňky či celé oblasti do většího množství buněk. Funguje tak, že se zvolená buňka/oblast označí myší a v dolním pravém rohu se objeví malý čtvereček, který podržíme levým tlačítkem myši a přetáhneme přes oblast, kam si přejeme obsah nakopírovat. Při automatickém vyplňování se však mění hodnota vzorců. Například pokud v buňce A1 máme vzorec odkazující na buňku B5, automatickým vyplněním obsahu z buňky A1 do A2 se změní i vzorec, a bude odkazovat na buňku B6. Na stejném principu se mění odkaz na buňku nejen při posunu nahoru a dolů, ale také do stran. Pokud chceme, aby vzorec zůstal stejný nehledě na to, do jaké buňky se pomocí automatického vyplňování zkopíruje, je nutné před označení řádku i sloupce napsat znak $ (dolar), např. B5. Pokud chceme, aby se měnil jen odkaz na sloupec, ale na řádek ne, $ umístíme jen před řádek (B$5) a vice versa.

2 Metody řešení soustavy lineárních rovnic

Numerické metody představují elegantnější variantu řešení soustavy lineárních rovnic, o nichž jsme mluvili v kapitole 1.2, především v případě, že se jedná o soustavy více rovnic o více neznámých, kde jsou ve většině případů efektivnější než běžně používané středoškolské metody, o nichž se zmiňuji níže.

Dosazovací metoda spočívá v tom, že se z jedné z rovnic soustavy vyjádří jedna z neznámých, za niž se poté dosadí do rovnic ostatních. Tento postup se opakuje tolikrát, dokud nezůstane jen jediná neznámá, kterou lze z rovnice vypočítat a poté za ni dosadit do rovnic ostatních, čímž postupně získáme všechny neznámé. Je vhodné zvolit proměnnou, kterou lze vyjádřit snadněji, a tedy za ni získáme co nejjednodušší výraz. U lineárních rovnic se vyhýbáme především zlomkům, u složitějších typů pak například odmocninám, se kterými se poté obtížně počítá. Za nevýhodu této metody můžeme považovat právě složitější výrazy, které dostáváme při opakované aplikaci na soustavu s více než dvěma neznámými. [13]

Srovnávací metoda funguje na podobném principu jako metoda dosazovací. Ze všech rovnic se tentokrát vyjádří jedna a ta samá neznámá, jejíž vyjádření se poté mezi sebou vhodně porovnávají. Vzhledem k tomu, že ze vzniklých rovnic se poté vyjadřují zbylé neznámé, které je v případě, že má soustava více než dvě neznámé, nutné pomocí dalších úprav zjišťovat, není tato metoda příliš vhodná pro řešení soustav více rovnic o více neznámých.

Sčítací metoda se nejvíce blíží Gaussově a Jordanově eliminační metodě, neboť používá stejných prostředků, a to řádkových úprav. Zásadní rozdíl tkví v tom, že při využití sčítací metody se zaměřujeme především na to, abychom pomocí vhodně zvoleného součtu k-násobku jednoho řádku s l-násobkem jiného řádku eliminovali vybranou neznámou, nicméně se mohou sčítat libovolné kombinace dvou řádků, pokud tím dostaneme smysluplný výsledek, tj. například nesmíme dostat nulový řádek. Za výhodu sčítací metody bychom mohli považovat relativní volnost ve výběru sčítaných řádků, a tedy i možnost zvolit si jednodušší výpočty, za nevýhodu pak, že v případě soustavy více rovnic o více neznámých může být výpočet chaotičtější než při použití striktně algoritmizované Gaussovy nebo Jordanovy eliminační metody. [13]

2.1 Gaussova eliminační metoda

Jedná se o jednu z přímých metod řešení soustavy lineárních rovnic. Jakožto přímá metoda obsahuje algoritmus přesného řešení, což znamená, že při provedení přesných výpočtů, tj. bez zaokrouhlování, dostáváme přesný výsledek. K provedení této metody je třeba soustavu lineárních rovnic nejprve přepsat do maticového tvaru, o němž jsme se zmiňovali v kapitole 1.2, a to včetně pravých stran, tedy absolutních členů jednotlivých rovnic. Je zvykem oddělit sloupec s absolutními členy svislou čarou. Výsledná matice se poté za pomoci ekvivalentních (řádkových) úprav, jak již bylo zmíněno v kapitole 1.2, upravuje na horní trojúhelníkový tvar. Upravená matice soustavy je pak ekvivalentní s původní sou-

stavou rovnic a snadno se z ní vypočítají neznámé. Při manuálním výpočtu často dochází k zaokrouhlovacím chybám, kterým se lze vyvarovat využitím počítačového softwaru. [19]

2.1.1 Algoritmizace pro MS Excel

1. krok

   Když $a_{11} \neq 0$, následuje 2. krok.

   Jinak:

   Když existuje $i \in \{2,3,4\}$ tak, že $a_{i1} \neq 0$, vyměníme první řádek s i-tým řádkem a následuje 2. krok.

   Jinak se nejedná o soustavu čtyř rovnic o čtyřech neznámých a algoritmus končí.

2. krok

   Pro $i \in \{2,3,4\}$:

   Celý i-tý řádek vynásobíme a_{11} a odečteme od něj a_{i1}-násobek řádku prvního.

   Následuje 3. krok.

3. krok

   Když $a_{22} \neq 0$, následuje 4. krok.

   Jinak:

   Když existuje $i \in \{3,4\}$ tak, že $a_{i2} \neq 0$, vyměníme druhý řádek s i-tým řádkem a následuje 4. krok.

   Jinak se za w nebo x musí zvolit parametr a rovnice nemá konkrétní řešení, algoritmus tedy končí.

4. krok

   Pro $i \in \{3,4\}$:

   Celý i-tý řádek vynásobíme a_{22} a odečteme od něj a_{i2}-násobek řádku druhého.

   Následuje 5. krok.

5. krok

   Když $a_{33} \neq 0$, následuje 6. krok.

   Jinak:

Když $a_{43} \neq 0$, vyměníme třetí řádek se čtvrtým řádkem a následuje 6. krok.

Jinak se za w, x nebo y musí zvolit parametr a rovnice nemá konkrétní řešení, algoritmus tedy končí.

6. krok

Celý čtvrtý řádek vynásobíme a_{33} a odečteme od něj a_{43}–násobek řádku třetího.

Následuje 7. krok.

7. krok

Když $a_{44} \neq 0$, následuje 9. krok.

Jinak následuje 8. krok.

8. krok

Když $a_{45} = 0$, má soustava nekonečně mnoho řešení.

Jinak nemá žádné řešení.

9. krok

Neznámá z se vypočítá s pomocí posledního řádku matice následujícím způsobem: $z = b_4/a_{44}$

Neznámá y se vypočítá s pomocí třetího řádku matice a nyní již známé proměnné z: $y = (b_3 – a_{34} * z)/a_{33}$

Neznámá x se vypočítá s pomocí druhého řádku matice a nyní již známých proměnných y a z: $x = (b_2 – a_{24} * z – a_{23} * y)/a_{22}$

Neznámá w se vypočítá s pomocí prvního řádku matice a nyní již známých proměnných x, y a z: $w = (b_1 – a_{14} * z – a_{13} * y – a_{12} * x)/a_{11}$

2.1.2 Implementace pro programovací jazyk

v zájmu porozumění ze strany těch, kteří se nevěnují programování, nevyužiji konkrétní programovací jazyk, nýbrž zvolím zápis ve zjednodušeném českém prostředí.

Pro úplné pochopení toho, jak je možné Gaussovu eliminační metodu použít za pomoci softwarového inženýrství (programování), bude nutné si definovat zobrazení matic v programovacích jazycích. Jak již víme, matice bývá reprezentována pomocí dvourozměrného pole, o němž jsme hovořili v kapitole 1.2, které může být konkrétním jazykem podporováno buď to přímo, a nebo se dá vyjádřit za pomoci pole polí. Pro větší

přehlednost v této práci zvolím dvourozměrné pole podporované přímo. Co se týče indexování, o němž se hovoří v kapitole 1.2, z matematického hlediska by bylo nejspíše vhodnější začínat indexem `1`, nicméně je třeba brát v potaz, že ačkoliv se některé jazyky od tohoto standardu odchylují, v informatice bývá zvykem začít indexem `0`. Další otázkou je, zda zvolit statické nebo dynamické typování, o čemž jsme mluvili v kapitole 1.2. Vzhledem k tomu, že všechny naše proměnné, tedy prvky matice, jsou stejného typu (`float`) a všechny operace, které použijeme, mají návratové hodnoty také tohoto typu, je volba typování víceméně libovolná. Já pro zjednodušení zápisu zvolím dynamické typování.

Deklarovaná matice bude tedy vypadat takto: `matice = pole[4, 5]`

Hranatá závorka udává rozměry pole, kde první z čísel určuje počet řádků a druhé z čísel určuje počet sloupců. Počet sloupců je o číslo vyšší, neboť matice obsahuje i pravé strany rovnic, které se v programování nedají oddělit tak snadno jako v matematice. Vzhledem k dynamickému typování by bylo možné matici rozdělit na dvourozměrné pole s rozměry [4, 4], které by obsahovalo levé strany, a jednorozměrné pole s velikostí [4], které by obsahovalo pravé strany, nicméně by to poněkud komplikovalo jak zápis, tak samotné provedení programu.

Na rozdíl od statického prostředí MS Excel, v němž je počet buněk a tedy i počet provedených kroků pevně dán, je programování pro výpočet značně pružnější, což nám umožňuje zapsat opakující se kroky, včetně těch, v nichž hrají roli podmínky, jež byly zmíněny v kapitole 1.2, pomocí cyklů, a výsledný kód tak bude kratší a přehlednější.

Podmínky se v Gaussově eliminační metodě využívají dvěma způsoby. Jedním z nich je rozhodnutí o tom, jakým způsobem se bude program chovat, podle hodnot řídících prvků řádků, o nichž jsme se zmiňovali v kapitole 1.2 a s nimiž pracuje. Konkrétně podle toho, zda se dané řídící prvky rovnají nule či nikoliv, a také zda mezi nimi existuje alespoň jeden nenulový koeficient. Zadruhé se pak podmínka využívá k zjištění, zda má daná soustava rovnic řešení, a pokud ano, pak zda je právě jedno konkrétní, parametrické řešení nezjistitelné s pomocí numerického výpočtu, či zda je řešení nekonečně mnoho.

Cykly, o nichž jsme hovořili v kapitole 1.2, se v Gaussově eliminační metodě používají k provedení opakujících se kroků, jakými jsou například operace s řádky, kdy k-násobek jednoho z řádků odčítáme od těch, jež jsou umístěny pod ním.

Matici je třeba naplnit manuálně, tento krok však vzhledem k uvedení postupu výpočtu není nutný, a tak jej můžeme přeskočit s tím, že je obsah matice pevně zadaný.

Pro samotný výpočet bude nejvhodnější vytvořit vlastní metodu, která bude mít jako proměnnou naši matici a vracet bude výsledek ve formě textového řetězce, ačkoliv vzhledem k dynamickému typování by v případě konkrétního výsledku mohla vracet jednorozměrné pole obsahující vypočítané proměnné, bude jednotný návratový typ jednodušší pro zápis.

v úvodu je třeba matici převést na horní trojúhelníkový tvar, což zajišťuje cyklus v první části metody. Pro každý krok nejprve zjistí, zda má na určeném místě nenulový koeficient řádek, jehož pořadí je totožné s pořadím kroku. Pokud ne, s pomocí cyklu hledá takový řádek, který zadaným požadavkům vyhovuje, a pokud jej najde, za dříve

zmíněný řádek jej vymění. V opačném případě metodu ukončí a vrátí informaci, že soustava má parametrické řešení. V dalším kroku s pomocí cyklu odečte k_i-násobek tohoto řádku od k-násobků řádků pod ním, přičemž k a k_i jsou zvolena tak, aby pod řídícím prvkem odčítaného řádku vznikly samé nulové koeficienty.

Po ukončení úvodního cyklu, a tedy převodu na trojúhelníkový tvar, je nutné za pomoci podmínek určit počet řešení. Pokud se program dostal v kódu až sem, je jisté, že jediné místo na hlavní diagonále, kde by se potenciálně mohl nacházet nulový koeficient, je prvek a_{44}, čili prvek v poli matice s indexy `[3, 3]`. Pokud se tento prvek rovná nule, soustava má buď žádné nebo nekonečně mnoho řešení. Pro rozlišení těchto dvou možností je třeba zjistit, jestli je prvek b_4, tedy `matice[3, 4]`, rovněž nulový, což by znamenalo, že soustava má nekonečně mnoho řešení, neboť by poslední řádek soustavy říkal, že nula je rovna nule, což je výrok platný nezávisle na hodnotě proměnných. Pokud by prvek b_4 byl nenulový, soustava nemá naopak žádné řešení, neboť by poslední řádek soustavy říkal, že nula je rovna nenulovému číslu, což není možné.

v případě, že se prvek a_{44} rovná nenulovému číslu, soustava rovnic má právě jedno řešení, které se s pomocí vzorců postupně vyjádří z matice. Jelikož jsme na začátku určili, že návratová hodnota bude jednotného typu, je třeba i v tomto případě vrátit textový řetězec. Jelikož se jeho obsah bude lišit v závislosti na hodnotách proměnných, je třeba je do textu dodatečně vložit. Každý programovací jazyk nabízí různé možnosti řešení, nejčastější je prosté skládání textu, já však použiji elegantnější zápis, který umožňuje vkládat proměnné přímo do textu v uvozovkách, tímto způsobem: `$`"Text {proměnná}".

```
GaussovaEliminace(matice)
{
 pro (i od 0 do 2)
 {
    když(matice[i, i] == 0)
    {
        konec = pravda;
        pro (j od i + 1 do 3)
            když(matice[j, i] != 0)
            {
                pro (k od 0 do 4)
                {
                    zastupce = matice[i, k];
                    matice[i, k] = matice[j, k];
                    matice[j, k] = zastupce;
                }
                konec = nepravda;
```

```
                přerušení;
            }
        když (konec == pravda)
            návrat "Soustava rovnic má parametrické
            řešení";
    }
    pro (j od i + 1 do 3)
    {
        koefi = matice[i, i];
        koefj = matice[j, i];
        pro (k od i do 4)
            matice[j, k] = matice[j, k] * koefi -
            matice[i, k] * koefj;
    }
}
když (matice[3, 3] == 0)
{
    když (matice[3, 4] == 0)
        návrat "Soustava rovnic má nekonečně mnoho
        řešení";
    jinak
        návrat "Soustava rovnic nemá řešení";
}
jinak
{
    z = matice[3, 4] / matice[3, 3];
    y = (matice[2, 4] - z * matice[2, 3]) / matice[2, 2];
    x = (matice[1, 4] - z * matice[1, 3] - y *
    matice[1, 2]) / matice[1, 1];
    w = (matice[0, 4] - z * matice[0, 3] - y *
    matice[0, 2] - x * matice[0, 1]) / matice[0, 0];
    návrat $"w = {w}, x = {x}, y = {y}, z = {z}";
}
}
```

2.1.3 Implementace pro MS Excel

Jako první je třeba si uvědomit, že na rozdíl od prostředí programovacího jazyka v informatice a manuálního výpočtu v matematice má MS Excel přesně daný počet a polohu jednotlivých buněk, a tudíž je nutné provést vždy přesně tentýž počet kroků, nezávisle na tom, zda jsou nebo nejsou splněny určité podmínky. Pro určení počtu kroků maticových úprav si vezmeme na pomoc algoritmizaci pro MS Excel, jež je popsána výše. Ve zmíněné algoritmizaci se v krocích 1 až 6 střídalo použití podmínek a použití cyklů. Této posloupnosti si můžeme všimnout i v algoritmu pro programovací jazyk, kde se v hlavním cyklu třikrát opakuje nejprve podmínka, která určí, zda může algoritmus okamžitě pokračovat dalším krokem, a pokud ne, pak zda je nutné prohodit řádky matice, a nebo zda vychází parametrické řešení a nemá tedy vůbec smysl pokračovat. Vzhledem k omezeným možnostem větvení v Excelu musí algoritmus pokračovat v každém případě, proto je nutné postup poupravit a možnost parametrického řešení dohledat zpětně.

Jak již bylo zmíněno, v případě, že se algoritmus nepřeruší, maticové úpravy mají celkem šest kroků. V Excelu si tedy vytvoříme sedm tabulek představujících matice, neboť krom matic obsluhujících jednotlivé kroky výpočtu potřebujeme ještě výchozí matici, do níž uživatel zadá vstupní tvar, na nějž se ostatní tabulky budou odkazovat.

v zájmu přehlednosti je vhodné si tabulky nadepsat názvy proměnných w, x, y a z, a pravý sloupec označením absolutního členu b. Též doporučuji tabulkám nastavit vhodné ohraničení a zabarvení buněk, pro lepší a přehlednější vizuální dojem. Nicméně tyto úpravy je lepší nechat až na konec, neboť v případě řešení opakujících se vzorců v buňce pomocí automatického vyplnění, o němž jsme mluvili v kapitole 1.2, se mimo vzorce zkopíruje i vizuální nastavení, a je pak nutné jej předělat, případně po každém vyplnění kliknout na objevivší se okénko Možnosti automatického vyplnění a zaškrtnou možnost *Vyplnit bez formátování*.

První tabulka bude tedy představovat výchozí matici. Jelikož v uživatelem zadaných hodnotách může být koeficient prvku a_{11} roven nule, je nutné v dalším kroku pomocí vhodně stanovených podmínek případně změnit pořadí řádků tak, aby byl koeficient a_{11} různý od nuly. Aby však toto bylo možné, je nejprve nutné se ujistit, že existuje alespoň jeden řádek, jehož koeficient a_{i1} je různý od nuly. K tomu můžeme použít pomocnou funkci, umístěnou vedle tabulky, ideálně vedle pravého horního rohu. Tato funkce bude vracet hodnotu `0` v případě, že pro všechny prvky a_{i1} platí, že jsou rovny nule, jinak bude vracet hodnotu `1`. Aby nenarušovala vizuální dojem, je možné nastavit barvu textu této buňky shodnou s barvou pozadí.

v případě, že funkce vrací hodnotu `0`, se další dva kroky vynechají, neboť by jejich provedení postrádalo smysl. Vynechání kroku v Excelu ovšem není možné řešit stejným způsobem jako v programování, a tak je nutné stanovit podmínku druhé a třetí matice, že ve zmíněném případě nebude počítat žádný vzorec, ale prostě bude odkazovat na hodnotu prvku v matici předchozí.

Pokud funkce vrací hodnotu `1`, řádky budou mít ve druhé matici buď to stejné anebo odlišné pořadí v závislosti na tom, jestli je koeficient a_{11} roven nule. V dalším kroku,

tedy ve třetí matici, se provede samotná eliminace a dostaneme tak krom prvního řádku všechny koeficienty v prvním sloupci nulové.

Následující čtyři matice opakují obdobné kroky, s tím rozdílem, že pomocná funkce hledá nenulový koeficient jen na souřadnicích a_{22}, resp. a_{33} a pod nimi. V sedmé matici tak dostáváme horní trojúhelníkový tvar, s jehož pomocí již můžeme vypočítat výsledky.

Nejprve je však nutné určit počet možných řešení, a tedy jestli má smysl snažit se vypočítat jedno konkrétní. Jako první ověříme možnost existence parametrického řešení, které v prostředí MS Excel nelze vypočítat. Parametrické řešení existuje právě když alespoň jedna z výše zmíněných pomocných funkcí vrací hodnotu `0`. V opačném případě již bezpečně víme, že koeficienty a_{11}, a_{22} a a_{33} jsou nenulové a netřeba jejich hodnoty v podmínkách nadále ověřovat.

o řešení nyní rozhodne poslední řádek, obsahující pouze koeficienty $a_{44}a\ b_4$. Zde mohou nastat pouze tři možné kombinace. Zaprvé $a_{44} \neq 0$, pak se b_4 může rovnat libovolnému reálnému číslu včetně nuly a soustava rovnic má právě jedno řešení. Pokud $a_{44} = 0$ a zároveň $b_4 = 0$, soustava má nekonečně mnoho řešení. V případě, že $a_{44} = 0$ a zároveň $b_4 \neq 0$, soustava nemá řešení žádné.

Pro ověření těchto podmínek se hodí zavést ještě jednu pomocnou funkci, která vrací hodnotu `0`, pokud nemá soustava žádné řešení, `1` pokud má právě jedno řešení, `2` pokud má nekonečně mnoho řešení a `3` pokud má parametrické řešení. Zbývá již vytvořit jen tabulku s výsledky, která bude zobrazovat konkrétní výsledek vypočítaný s pomocí dříve zmíněných vzorců, nebo text oznamující, že soustava rovnic má jiný počet řešení.

2.2 Jordanova metoda

Stejně jako Gaussova eliminační metoda, jíž se v mnohém podobá, je Jordanova metoda přímou metodou řešení soustav lineárních rovnic. Stejně jako Gaussova metoda, upravuje Jordanova metoda matici soustavy rovnic s pomocí ekvivalentních úprav na horní trojúhelníkový tvar. V následujícím postupu se však tato metoda liší. zatímco Gaussova metoda napřímo vyjadřuje jednotlivé proměnné z upravené matice soustavy, Jordanova metoda pokračuje v ekvivalentních úpravách matice soustavy na redukovaný trojúhelníkový tvar, jenž je zmíněn v kapitole 1.2. Řešení je tak zřejmé přímo z pohledu na matici, která má v každém řádku pouze jednu proměnnou na souřadnicích a_{ii}, kde i je číslo řádku, s jednotkovým koeficientem a k ní odpovídající hodnotu absolutního členu b_i. Výhodu oproti Gaussově eliminaci představuje velice přehledné řešení soustavy, nevýhodu pak pracné výpočty často zahrnující zlomky či desetinná čísla. [19]

2.2.1 Algoritmizace pro MS Excel

Jak již bylo řečeno, Jordanova metoda má část postupu společnou s Gaussovou eliminací, a tím pádem bude z větší části algoritmizace totožná. Obě metody zahrnují nejen úpravu na horní trojúhelníkový tvar, ale taktéž mají stejný postup, jímž se určuje počet řešení. Až teprve v případě, že má soustava právě jedno řešení, postupuje Jordanova metoda odlišně.

Její algoritmizace bude mít tedy s Gaussovou eliminací společných prvních osm kroků. Postup se liší až počínaje krokem devátým, který však v tomto případě není poslední.

9. krok

Pro $i \in \{1, 2, 3, 4\}$:

Celý i-tý řádek vydělíme koeficientem a_{ii}

Následuje 10. krok.

10. krok

Pro $i \in \{1, 2, 3\}$:

Od celého i-tého řádku odečteme a_{i4} násobek řádku čtvrtého.

Následuje 11. krok.

11. krok

Pro $i \in \{1, 2\}$:

Od celého i-tého řádku odečteme a_{i3} násobek řádku třetího.

Následuje 12. krok.

12. krok

Od celého prvního řádku odečteme a_{12} násobek řádku druhého.

Následuje 13. krok.

13. krok

z upravené matice určíme jednotlivé proměnné následujícím způsobem: $w = b_1$, $x = b_2$, $y = b_3$, $z = b_4$

2.2.2 Implementace pro programovací jazyk

Jak bylo již zavedeno v případě Gaussovy eliminace, pro popis algoritmu v prostředí nespecifikovaného programovacího jazyka bude použito dynamické typování a indexovat se bude od nuly. Až na samotné určení konkrétního řešení bude postup obdobný. Jediný rozdíl bude spočívat v tom, že namísto využití vzorců pro výpočet řešení zde budou využity cykly, s jejichž pomocí se matice upraví na tvar, z něhož již bude patrný výsledek. První ze zmíněných cyklů vydělí každý řádek koeficientem jeho řídícího prvku, aby tak každý řídící prvek byl roven 1, zatímco druhý cyklus odečte k-násobek každého řádku od řádků nad ním, čímž získáme redukovaný trojúhelníkový tvar matice.

```
JordanovaEliminace(matice)
{
```

```
pro (i = 0; i <= 2; i++)
{
    když(matice[i, i] == 0)
    {
        konec = pravda;
        pro (j od i + 1 do 3)
            když(matice[j, i] != 0)
            {
                pro (k od 0 do 4)
                {
                    zastupce = matice[i, k];
                    matice[i, k] = matice[j, k];
                    matice[j, k] = zastupce;
                }
                konec = nepravda;
                přerušení;
            }
        když (konec == pravda)
            návrat "Soustava rovnic má parametrické
            řešení";
    }
    pro (j od i + 1 do 3)
    {
        koefi = matice[i, i];
        koefj = matice[j, i];
        pro (k od i do 4)
            matice[j, k] = matice[j, k] * koefi –
            matice[i, k] * koefj;
    }
}
když (matice[3, 3] != 0)
{
    pro (i od 0 do 3)
    {
```

```
            koef = matice[i, i];
            pro (j od 0 do 4)
                matice[i, j] = matice[i, j] / koef;
        }
        pro (i od 3 do 1)
            pro (j od 0 do i)
            {
                koef = matice[j, i];
                pro (k od 0 do 4)
                    matice[j, k] = matice[j, k] - matice[i, k]
                    * koef;
            }
        návrat $"w = {matice[0, 4]}, x = {matice[1, 4]}, y =
        {matice[2, 4]}, z = {matice[3, 4]}";
    }
    jinak když (matice[3, 4] == 0)
        návrat "Soustava rovnic má nekonečně mnoho řešení";
    jinak
        návrat "Soustava rovnic nemá řešení";
}
```

Jak je vidět, většina kódu je totožná s Gaussovou eliminací a liší se skutečně jen podmínka, jež ošetřuje případ, kdy má soustava právě jedno řešení.

2.2.3 Implementace pro MS Excel

i zde bude základ velice podobný tomu, s nímž jsme se seznámili v kapitole 2.1. Celý postup úprav matice na horní trojúhelníkový tvar bude fungovat naprosto stejně, a to včetně využití pomocných funkcí určujících, zda existuje alespoň jeden nenulový koeficient v daném sloupci upravované části matice, a pomocné funkce určující počet a typ řešení.

Právě na poslední zmíněnou přímo naváže další výpočet, který se, jak víme, provádí pouze v případě, že existuje právě jedno řešení. Nejdříve je ale třeba určit počet kroků a tedy i počet tabulek pro úpravy matice. K tomu si opět vypůjčíme algoritmizaci pro MS Excel, v níž se v krocích 9 až 12 matice upravuje na redukovaný trojúhelníkový tvar. Kroky jsou tedy celkem čtyři, a jelikož jako výchozí matici, na niž budou tyto kroky odkazovat, již máme ve formě posledního kroku úpravy na horní trojúhelníkový tvar, budeme potřebovat opravdu jen čtyři tabulky. Stejně jako u Gaussovy eliminace je vhodné

si je správně nadepsat názvy proměnných a pravý sloupec označením absolutního členu. Po zadání všech potřebných vzorců také doporučuji tabulky vhodně naformátovat, pro lepší přehlednost a vizuální dojem.

Nyní, když máme tabulky zavedené, využijeme zmíněnou pomocnou funkci, a jako základní podmínku do všech buněk zobrazujících hodnoty v matici nastavíme, že pokud soustava rovnic nemá právě jedno řešení, bude místo číselného výsledku, ať už je jakýkoliv, zobrazovat znak nebo řetězec znaků, oznamující, že pro tuto soustavu nemá provádění úprav na redukovaný trojúhelníkový tvar význam. Pro příklad můžeme za tento znak zvolit pomlčku.

v případě, že soustava rovnic má právě jedno řešení, budeme pokračovat následovně. V prvním kroku, tedy i v první tabulce, vydělíme každý řádek koeficientem jeho řídícího prvku, abychom na místech koeficientů a_{ii} dostali samé jedničky a mohli si tak zjednodušit následující výpočty, v nichž budou tyto koeficienty hrát zásadní roli.

Nyní vezmeme čtvrtý řádek a od každého řádku nad ním odečteme jeho a_{i4}-násobek, abychom tak na místo koeficientů a_{i4}, kde $i \in \{1, 2, 3\}$. Totéž zopakujeme pro třetí řádek a koeficienty a_{i3}, kde $i \in \{1, 2\}$, a druhý řádek a koeficient a_{i2}, kde $i = 1$. Získáváme tak jednotkovou matici, o níž se zmiňujeme v kapitole 1.2, na levé straně, a sloupec odpovídající hodnotám neznámých na straně pravé.

Zbývá už jen vytvořit tabulku s výsledky, která se bude, stejně jako u Gaussovy eliminace, odkazovat na pomocnou funkci, jež nám určuje počet a typ řešení, a v závislosti na její hodnotě se zde zobrazí buď konkrétní řešení, a pakliže je počet a typ řešení jiný, tabulka nás o tom informuje.

2.3 Jacobiho metoda

Jedná se o iterační metodu řešení soustavy lineárních rovnic. Znamená to, že při jejím použití vytváříme posloupnost postupných aproximací, která by měla konvergovat k řešení dané soustavy. Jacobiho metoda však nemusí vždy konvergovat. V případě, že konverguje, nahrazuje se řešení dané soustavy aproximací o předem zadané přesnosti, případně aproximací vzniklou po předem daném počtu kroků. Nekonečný proces konvergence je tak nahrazen konečným, za vzniku určité nepřesnosti. K dosažení požadované přesnosti je někdy zapotřebí většího objemu výpočtů než u metod přímých. Na druhou stranu lze použít mnohem jednodušší algoritmus, čímž se šetří paměť počítače i snižuje náročnost implementace v prostředí programovacího jazyka i v prostředí MS Excel. Proto je tato metoda mnohem vhodnější pro využití v informatice nežli v matematice, kde se výpočty provádějí ručně. [19]

Úskalím Jacobiho metody je, že abychom vyjádřili proměnnou, musíme vzorec v určitém místě dělit koeficientem u této proměnné. Zde narážíme na problém, že aby výpočet dával smysl, tento koeficient musí být nenulový. Pokud si proměnné vyjádříme systematicky tak, že z prvního řádku vyjádříme první proměnnou, z druhého druhou a tak dále, dostaneme se k závěru, že nuly se nesmí nacházet na hlavní diagonále. Abychom se ujistili, že je vůbec možné toho docílit, a také že daná soustava má právě jedno

řešení, a má tedy smysl se jej snažit vypočítat, je třeba si ověřit, zda je matice levých stran soustavy regulární, tj. jestli jsou její řádky lineárně nezávislé, jak bylo vysvětleno v kapitole 1.2.

2.3.1 Algoritmizace pro MS Excel

1. krok

   s pomocí funkce DETERMINANT zjistíme determinant matice levých stran soustavy.

   Když je determinant roven nule, soustava nemá konkrétní řešení a algoritmus končí.

   Jinak následuje 2. krok.

2. krok

   Když pro všechna $i \in \{1,2,3,4\}$ platí:$a_{ii} \neq 0$, následuje 4. krok.

   Jinak následuje 3. krok.

3. krok

   Vytvoříme všechny možné permutace pořadí řádků {1, 2, 3, 4} a pro každou z nich určíme, zda na hlavní diagonále obsahuje nulu, či nikoliv.

   První permutaci, která nulu na hlavní diagonále neobsahuje, použijeme k výměně řádků původní matice. Následuje 4. krok.

4. krok

   Od uživatele získáme požadovanou přesnost zadanou ve formě kladného desetinného čísla.

   Zadáme libovolnou počáteční aproximaci w_0, x_0, y_0 a z_0, kterou využijeme k výpočtu další aproximace w_0, x_0, y_0 a z_0.

   Dokud pro všechna $e \in \{w,x,y,z\}$ platí $|e_{100}-e_{99}| >$ požadovaná přesnost a $i <= 100$:

   $$w_i = (b_1-x_{i-1} * a_{12}-y_{i-1} * a_{13}-z_{i-1} * a_{14})/a_{11}$$
   $$x_i = (b_2-w_{i-1} * a_{21}-y_{i-1} * a_{23}-z_{i-1} * a_{24})/a_{22}$$
   $$y_i = (b_3-w_{i-1} * a_{31}-x_{i-1} * a_{32}-z_{i-1} * a_{34})/a_{33}$$
   $$z_i = (b_4-w_{i-1} * a_{41}-x_{i-1} * a_{42}-y_{i-1} * a_{43})/a_{44}$$

5. krok

```
Když pro všechna e ∈ {w, x, y, z} platí |e100−e99| <= |e99−e98|
    Když pro všechna e ∈ w, x, y, z platí |e100−e99| <=
    požadovaná přesnost, algoritmus našel řešení
    o požadované přesnosti.
    Jinak počet kroků nestačil pro dosažení
    požadované přesnosti
Jinak metoda diverguje.
```

2.3.2 Implementace pro programovací jazyk

Na rozdíl od předchozích dvou metod, které řeší všechny alternativy během postupu, Jacobiho metoda se zaměřuje pouze na samotnou iteraci, aproximující hledané řešení. Je proto třeba případné další alternativy ověřit zvlášť. Pro přehlednost bude nejlepší pro tento účel vytvořit samostatné metody, které bude samotná Jacobiho metoda pouze volat. Na rozdíl od metod předchozích nebude ztvárnění Jacobiho metody v prostředí programovacího jazyka nutně jednodušší, neboť vlastnost Excelu, kde je pevně daný počet a poloha buněk, je v případě použití Jacobiho metody mnohdy užitečná.

Jako první je třeba zjistit, zda má soustava jedno jediné řešení, čili zda jsou řádky matice levých stran soustavy lineárně nezávislé. Toho lze dosáhnout buď převodem na trojúhelníkový tvar, jak jsme činili u předchozích metod, nebo za pomoci determinantu, jehož princip byl vysvětlen v kapitole 1.2. Zde narážíme na první úskalí, neboť naše matice soustavy obsahuje i pravé strany a pro výpočet determinantu potřebujeme matici čtvercovou, neboli pouze matici levých stran. V Excelu bychom toho docílili velice snadno, a sice tak, že bychom označili pouze buňky s koeficienty proměnných, čili levých stran soustavy. Nicméně v prostředí programovacího jazyka máme pole velikosti 4x5 a pro získání pole 4x4 budeme muset vytvoří pole nové, do nějž za pomoci cyklů nakopírujeme matici levých stran soustavy. Poté, v závislosti na konkrétním programovacím jazyku, buď to použijeme přímo funkci pro determinant, má-li ji daný jazyk definovanou, případně budeme muset pole nejprve převést na jiný datový typ (např. typ Matrix v jazyce C#) a funkci determinant aplikovat až poté, a v nejhorším případě bude nutné funkci determinant naprogramovat ručně. V zájmu zjednodušení zápisu předpokládejme, že funkce determinant již naprogramovaná je, a to včetně separace matice levých stran, proto její kód nebudu v algoritmizaci uvádět. V případě, že je determinant roven nule, Jacobiho metoda končí a vrací oznámení, že soustava nemá konkrétní řešení. Jinak algoritmus pokračuje.

Další, co je třeba ověřit, než začne iterace samotná, je, zda se na hlavní diagonále matice nenacházejí nuly, neboť jak již bylo zmíněno, při vyjádření proměnných bude třeba vzorce dělit koeficienty na hlavní diagonále. Metoda určená pro uvěření a případnou nápravu tohoto případu nejprve ověří, zda se na hlavní diagonále nachází pouze nenulová čísla, a v případě, že ne, určí, jakým způsobem změnit pořadí řádků, aby se na hlavní diagonále žádná nula nenacházela. Bude k tomu potřeba postupně vyzkoušet všechny permutace pořadí $\{0, 1, 2, 3\}$, dokud metoda nenajde tu, pro niž zadaný požadavek platí. Pokud nechceme všech 24 permutací vypisovat ručně, poslouží nám vnořené cykly proměnných

i, j, k a l, kde každá z nich bude nabývat hodnot od 0 do 3. Pro každý krok bude tedy metoda nejprve ověřovat, zda je uspořádaná čtveřice $\{i, j, k, l\}$ jednou z permutací, které potřebujeme, čili zda obsahuje každá ze zmíněných čísel právě jednou. V případě, že ano, ověří, jestli se při tomto pořadí na hlavní diagonále nachází jen nenulové prvky, a pokud ano, permutaci uloží do proměnné a cyklus ukončí. Následně metoda vytvoří nové pole o stejných rozměrech a vloží do něj řádky původní matice s ohledem na permutaci získanou v předchozím kroku.

Jelikož metodu samotnou díky možnosti využití while cyklu nemusíme omezovat počtem kroků, je nutné nejprve zjistit, zda metoda nebude divergovat, neboť v takovém případě by cyklus omezený pouze požadovanou přesností by pak nikdy neskončil. Metoda ověřující konvergenci krom počáteční aproximace zadané jako $w = x = y = z = 0$ vypočítá další tři aproximace a s pomocí podmínek ověří, zda se při dalším kroku rozdíly mezi jednotlivými aproximacemi nezvětšují, a jestli to platí pro všechny proměnné. Pokud ano, Jacobiho metoda konverguje a pomocná metoda ověřující konvergenci vrací hodnotu pravda. Jinak vrací hodnotu nepravda a Jacobiho metoda končí a vrací sdělení, že pro tuto soustavu diverguje.

v případě, že soustava rovnic splňuje všechny potřebné požadavky, můžeme přejít k samotné iteraci. Jelikož v prostředí programovacího jazyka můžeme použít while cyklus, jehož pokračování je podmíněno konkrétní podmínkou, namísto pevně daného počtu kroků, není nutné se omezovat na konkrétní počet kroků. Pro začátek je nutné správně implementovat podmínku, při jejímž splnění cyklus skončí. Jak bylo již dříve uvedeno, onou podmínkou bude přesnost, a sice zda je rozdíl posledních dvou aproximací pro každou z proměnných větší nebo rovný požadované přesnosti. Pro začátek si zvolme proměnnou typu bool, tedy logická hodnota, což v prostředí dynamicky typovaného prostředí není nutné předem specifikovat, já tuto informaci však uvádím, aby bylo jasné, že tato proměnná bude nabývat pouze hodnot pravda či nepravda. Pro začátek si ji nastavme na hodnotu nepravda, čímž také podmíníme pokračování while cyklu. Abychom nemuseli podmínku do kódu vypisovat celou, a zároveň abychom nemuseli psát vzorec pro výpočet aproximací jednotlivých proměnných čtyřikrát, je vhodné uvnitř while cyklu vytvořit for cyklus, který vypočítá novou hodnotu proměnné a zároveň ověří podmínku, která pokud je nepravdivá, nastaví se hodnota kontrolní proměnné na nepravda a cyklus bude pokračovat i v případě, že ostatní tři proměnné požadované přesnosti již dosáhly. Menší problém by mohlo představovat jednotné řešení výpočtu aproximace pro všechny proměnné. Pokud si však uvědomíme, že vzorec pro každou z proměnných představuje absolutní prvek, od něhož se odečítají ostatní proměnné násobené koeficienty k nim náležející, a celý tento rozdíl se vydělí koeficientem proměnné, kterou právě počítáme, můžeme dojít k zobecněnému vzorci, který od absolutního členu odečte všechny čtyři proměnné, a zároveň přičte zpět tu proměnnou, již právě počítáme a jejímž koeficientem budeme dělit.

```
Konvergence(matice)

{
```

```
    konvg = pole[4];
    pro(i od 0 do 3)
        konvg[i] = { 0, 0, 0, 0 };
    pro(i od 0 do 2)
        pro (j od 0 do 3)
            konvg[i+1][j] = Hodnota(matice, konvg[i], j);
    pro (i od 0 do 3)
        když (AbsHodnota(konvg[1][i] – konvg[2][i]) <
        AbsHodnota(konvg[2][i] – konvg[3][i]))
            návrat nepravda;
    návrat pravda;
}
Řazení(matice)
{
    permutace = pole[4];
    pro(i od 0 do 3)
        pro(j od 0 do 3)
            pro (k od 0 do 3)
                pro (l od 0 do 3)
                {
                    permutace =  i, j, k, l;
                    když (permutace.Delka() ==
                    permutace.Distinct().Delka() &&
                    matice[i, 0] * matice[j, 1] *
                    matice[k, 2] * matice[l, 3] != 0)
                        přerušit;
                }
    matice2 = pole[4, 5];
    pro (i od 0 do 3)
        pro (j od 0 do 4)
            matice2[i, j] = matice[permutace[i], j];
    návrat matice2;
}
Hodnota(matice, promenne0, i)
```

```
{
    návrat (matice[i, 4] – promenne0[0] * matice[i, 0]
    – promenne0[1] * matice[i, 1] – promenne0[2]
    * matice[i, 2] – promenne0[3] * matice[i, 3] +
    promenne0[i] *
    matice[i, i]) / matice[i, i]);
}
JacobihoMetoda(matice)
{
    když(Determinant(matice) == 0)
        návrat "Soustava nemá konkrétní řešení";
    matice = Řazení(matice);
    když (Konvergence(matice) == nepravda)
        návrat "Jacobiho metoda diverguje";
    Napsat("Zadejte požadovanou přesnost: ");
    presnost = Číst();
    promenne0 =  0, 0, 0, 0 ;
    promenne1 = pole[4];
    hotovo = nepravda;
    while(hotovo == nepravda)
    {
        hotovo = pravda;
        pro (i od 0 do 3)
        {
            promenne1[i] = Hodnota(matice, promenne0, i);
            Když (AbsHodnota(promenne0[i] – promenne1[i])
            > presnost)
                hotovo = nepravda;
        }
        promenne0 = promenne1;
    }
    návrat $"w = promenne0[0], x = promenne0[1], y =
    promenne0[2], z = promenne0[3]";
}
```

2.3.3 Implementace pro MS Excel

Na rozdíl od prostředí programovacího jazyka, zpracování Jacobiho metody v Excelu musí vždy provést všechny kroky bez ohledu na to, zda má vlastně smysl je provádět. Nicméně na rozdíl od programovatelného softwaru, v prostředí MS Excel pro chod programu samotného nevadí, když některá z funkcí skončí chybou. Proto je zde vhodnější nejprve vytvořit metodu samotnou a postup pro všechny eventuality. Různé možnosti, které jsme v programování ošetřovali předem, musíme ověřit až poté, a přizpůsobit jim funkci a zobrazení programu.

Stejně jako u předchozích dvou metod, i zde je třeba nejprve vytvořit tabulku pro vstupní hodnoty matice, které zadá uživatel, a od nichž se budou odvíjet další kroky. V zájmu přehlednosti doporučuji tabulku vhodně označit a naformátovat.

Pod tuto tabulku vytvoříme ještě jednu totožnou, do níž vložíme odkazy na řádky původní tabulky pro případ, že bude nutné změnit jejich pořadí. V algoritmizaci pro programovací jazyk jsme s pomocí cyklů hledali první permutaci pořadí řádků, která by vyhovovala požadavku, že se na hlavní diagonále nesmějí nacházet nuly. Vzhledem ke statické povaze prostředí MS Excel bude zde nutné prověřit všechny možné permutace, nehledě na to, zda bude tento krok pro konkrétní matici nutný či nikoliv.

Jelikož má naše matice čtyři řádky, počet permutací bude roven $4!$, tedy 24. Doporučuji si pro permutace vyhradit tabulku o velikosti 4×6, kde v každém ze čtyř řádků budou permutace začínající jedním z čísel, každý řádek pak bude rozdělen na tři části po dvou buňkách, kde v každé části bude na druhém místě jedno ze tří zbývajících čísel, a nakonec v každé ze dvou buněk bude třetí a čtvrté místo zaujímat jedna z možností seřazení dvou zbylých čísel. Je třeba dát si pozor, abychom vlivem nepozornosti nevytvořili dvě totožné permutace. Jelikož MS Excel nepodporuje datový typ pole, permutaci zapíšeme ve formě čtyřciferného čísla, kde pořadí jednotlivé číslice bude odkazovat na nové pořadí řádku, a hodnota číslice na původní pořadí řádku.

Nyní když máme jednotlivé permutace rozepsané, musíme pro každou z nich vytvořit podmínku, která ověří, zda se při zadaném pořadí řádků na hlavní diagonále nachází nuly, či nikoliv. Permutace samotná nám zde poslouží jako návod pro lokalizaci jednotlivých buněk, jejichž hodnotu je třeba ověřovat, v tabulce. Pořadí každé číslice bude značit pořadí sloupce a hodnota číslice bude značit pořadí řádku. Podmínková funkce poté vrátí čtyřciferné číslo odpovídající dané permutaci v případě, že hodnota ani jedné buňky, jejíž hodnotu ověřuje, není rovna nule, a nečíselný znak, například pomlčku, v případě, že hodnota alespoň jedné z daných buněk rovna nule je.

Pokud jsou řádky matice lineárně nezávislé, bude existovat alespoň jedna permutace, pro níž na hlavní diagonále nejsou žádné nuly. Tuto permutaci separujeme s použití funkce `MIN`, případně `MAX`, kterou aplikujeme na celou tabulku s permutacemi. Z čísla, které tato funkce vrátí, poté z pomocí celočíselného dělení, zastoupeného dělením vloženým do funkce `ZAOKR.DOLŮ`, a funkce `MOD`, jež vrací zbytek po celočíselném dělení, postupně vyseparujeme jednotlivé cifry, každou do jiné buňky. Čili kombinaci zmíněných funkcí bude třeba použít čtyřikrát a pokaždé vhodně zvolit, kterým číslem budeme dělit. Celý

tento krok, včetně tabulky s permutacemi, můžeme naformátovat tak, aby barva textu odpovídala barvě pozadí, a výpočty tak pro lepší vizuální dojem skrýt.

Nyní se můžeme vrátit k připravené druhé tabulce, do níž v závislosti na pořadí řádků určeném v předchozích krocích postupně vložíme řádky z tabulky původní. Nyní máme tabulku, na jejíž hlavní diagonále by se neměly nacházet nuly, a můžeme přejít k samotné iteraci. Jediný případ, pro nějž se na hlavní diagonále nuly nacházet budou, je ten, kdy jsou řádky matice lineárně závislé. V tomto případě funkce `MIN`, případně `MAX` z předchozího kroku vrací nulu. Jelikož však pro tento případ není třeba určovat podmínku v samotných výpočtech, k jeho ověření se vrátíme až posléze a tabulky podle něj podmíněně naformátujeme.

Na řadě je již samotná iterace. Pro tento krok vytvoříme tabulku, která bude bez záhlaví obsahovat 100 řádků (tento počet je však volitelný). První řádek necháme volný pro prvotní aproximaci, kterou může uživatel libovolně zadat. Druhým řádkem počínaje vyjádříme další aproximace za pomoci vzorců vyjádřených ze druhé tabulky s prvky matice, s dosazením předchozí aproximace. Jelikož opisovat tyto vzorce pro všech 100 řádků by bylo velmi nepraktické, použijeme pro třetí až stý řádek automatické vyplnění řádkem druhým, takže je potřeba uzamknout pořadí řádků u odkazů na buňky tabulky s prvky matice. Jelikož automatické vyplňování provádíme jen ve vertikálním směru, pořadí sloupců není nutné zamykat. Tuto tabulku si vhodně naformátujeme, barvu pozadí zvolíme například žlutou.

Tímto máme všechny výpočty hotové a na řadu přichází ověřování podmínek. Jako první je potřeba ověřit, zda jsou řádky matice lineárně nezávislé. K tomu nám pomůže funkce `DETERMINANT`, do níž jako oblast hodnot zadáme levou stranu první tabulky s prvky matice. Nastavíme podmíněné formátování druhé tabulky s prvky matice tak, že pokud buňka s funkcí `DETERMINANT` obsahuje nulovou hodnotu, tabulka nebude zobrazovat hodnoty, tedy barvu textu nastavíme na totožnou s pozadím, které můžeme nastavit na červenou, či jinou varovnou barvu. Při splnění zmíněné podmínky nastavíme tutéž barvu pozadí pro první řádek tabulky s iterací, zatímco ostatním řádkům nastavíme pozadí na průsvitné a barvu textu na bílou, čili se bude zdát, že tabulka více řádků než jeden nemá. Buňku s funkcí determinant můžeme též skrýt tím, že nastavíme barvu písma na bílou.

Pro ověřování zbylých podmínek si nastavíme dvě speciální okénka, za tímto účelem je možné například sloučit více buněk. Jedno bude ověřovat všechny možnosti a v závislosti na nich zobrazí odpovídající informaci, zatímco do druhého bude uživatel zadávat požadovanou přesnost ve formě kladného, většinou desetinného, čísla. První podmínka, kterou bude informační okénko ověřovat, bude již zmíněná buňka s funkcí determinant. Pokud bude její hodnota rovna nule, okénko nás informuje, že soustava nemá konkrétní řešení. Další podmínkou bude divergence, čili zda Jacobiho metoda pro konkrétní soustavu diverguje. Pro ověření této podmínky můžeme využít tři libovolné po sobě jdoucí řádky postupných aproximací, kde pro každý ze sloupců zjistíme, jestli je absolutní hodnota, jež byla zmíněna v kapitole 1.2, rozdílu prvních dvou menší než

absolutní hodnota rozdílu druhých dvou. Pokud ano, metoda diverguje a okénko o tom informuje uživatele.

Poslední podmínka se bude týkat přesnosti, pro níž bude potřeba ověřovat každý řádek iterační tabulky zvlášť. Pro pomocnou funkci si tentokrát vybereme celý sloupec, který by neměl obsahovat nic viditelného, neboť na konci zmenšíme jeho šířku na nulu, aby nekomplikoval vizuální dojem. V každém řádku tohoto sloupce, počínaje řádkem třetím v aproximační tabulce, ověříme, zda již přesnost v každém ze sloupců dosáhla požadované přesnosti, tedy zda pro všechny sloupce aproximační tabulky platí, že absolutní hodnota rozdílu současného řádku a předchozího řádku je menší nebo rovna hodnotě zadané v okénku pro požadovanou přesnost. Pokud ne, funkce vrátí hodnotu -1. Pokud ano a předchozí řádek vrátil hodnotu -1 (čili tento řádek je první), funkce vrátí 0, jinak vrátí 1. Nyní pro řádky 3 až 99 iterační tabulky nastavíme podmíněné formátovaní, kde v případě, že se hodnota pomocné funkce pro daný řádek rovná 0, formátování řásku se zvýrazní – například použitím tučného fontu, ohraničení a jiného zabarvení, například zeleného. Pokud pomocná funkce bude rovnat 1, řádek pomocí formátování „zneviditelníme“. Pokud se pomocná funkce bude rovnat -1, formátování se nezmění, a to s výjimkou posledního řádku, pro který bychom v tomto případě nastavili barvu pozadí na varovnou barvu, v tomto případě značící neúspěch.

Pomocné funkce u stého řádku využije též informační okénko, které v případě, že se funkce bude rovnat -1, uživateli sdělí, že nebylo dosaženo požadované přesnosti. Pokud ani jedna ze tří ověřovaných podmínek nebyla pravdivá, metoda požadovaný výsledek nalezla a okénko zobrazí informaci, že výsledek je k nalezení na konci iterační tabulky. Též je možné v tomto případě nastavit podmíněné formátování okénka na zelené pozadí, zatímco v ostatních případech bude červené.

Závěr

Cílem práce bylo seznámit čtenáře s možnostmi a postupy využití numerických metod pro řešení soustavy lineárních rovnic v kancelářském softwaru MS Excel. Tento cíl byl splněn ve třech podkapitolách věnujících se konkrétním třem metodám, mezi něž patří Gaussova eliminace, Jordanova eliminace a Jacobiho metoda. U každé ze zmíněných metod byl podrobně rozebrán postup implementace v programu MS Excel, a to včetně vysvětlení, proč bylo postupováno právě takto, přičemž nechybělo ani doplnění o optimalizaci vizuální stránky.

Řešení pro každou metodu je zvoleno tak, aby bylo co nejvíce přehledné a smysluplné. V některých případech, jako například určení správného pořadí řádků pro Jacobiho metodu, bylo hledání elegantního řešení skutečnou výzvou, avšak mohu říci, že má snaha byla úspěšná.

Pro čtenáře, jenž jsou obeznámeni s programováním, obsahuje práce též algoritmizaci pro prostředí programovacího jazyka, který v zájmu zachování univerzality nebyl přesně určen, místo konkrétního jazyka bylo použito zjednodušené české prostředí s dynamickou typovou kontrolou, podporou vícerozměrných polí a indexováním od nuly. Pro využití v programování bude nutné algoritmus více či méně upravit, v závislosti na konkrétním programovacím jazyce. Neměl by to však být problém, neboť i tato algoritmizace je doplněna o slovní vysvětlení postupu.

Tato práce může být v praxi využita jako podklad pro přípravu výuky numerických metod, které rozebírá, stejně tak jako pro samostudium případných zájemců, či studentů, kteří se z nějakého důvodu na samotnou výuku dostavit nemohou.

v dalším studiu by bylo možné práci rozšířit o některé metody pro výpočet kořenů nelineárních rovnic, u nichž některé aspekty mohou představovat skutečnou výzvu, a to jak v prostředí programovacího jazyka, tak v samotném programu MS Excel.

Reference

[1] ČERMÁK, Libor a Rudolf HLAVIČKA. *Numerické metody*. Brno: AKADEMICKÉ NAKLADATELSTVÍ CERM, s.r.o. Brno, 2016. ISBN 978-80-214-5437-8.

[2] VONDRÁK, Vít a Lukáš POSPÍŠIL. *Numerické metody I.*

[3] HASÍK, Karel. *Numerické metody*.

[4] Numerický model - nápověda *IN-POČASÍ* [online]. [cit. 2021-03-13]. Dostupné z: https://www.in-pocasi.cz/model/napoveda/

[5] KUBÍČEK, Milan, Miroslava DUBCOVÁ a Drahoslava JANOVSKÁ. *Numerické metody a algoritmy* [online]. 2. Praha: Vysoká škola chemicko-technologická v Praze, 2005 [cit. 2021-03-13]. ISBN 80-7080-558-7. Dostupné z: http://147.33.74.135/knihy/uid_isbn-80-7080-558-7/pages-img/

[6] KAW, Autar a Melinda HESS. Assessing Teaching Methods for a Course in Numerical Methods. *ASEE Annual Conference & Exposition* [online]. 2006 [cit. 2021-03-13]. Dostupné z: doi:10.18260/1-2–547

[7] Wolfram Mathematica: The world's definitive system for modern technical computing. *Wolfram* [online]. [cit. 2021-03-13]. Dostupné z: https://www.wolfram.com/mathematica/

[8] Wolfram Mathematica 8. *Mathematica* [online]. [cit. 2021-03-13]. Dostupné z: http://www.mathematica.cz/produkty.php?p_mathematica

[9] *Czech Software First: Autorizovaný distributor Maplesoft Inc.* [online]. [cit. 2021-03-13]. Dostupné z: https://www.maplesoft.cz/

[10] *MathWorks* [online]. [cit. 2021-03-13]. Dostupné z: https://uk.mathworks.com/

[11] MATLAB: Jazyk pro technické výpočty. *Humusoft* [online]. [cit. 2021-03-13]. Dostupné z: https://www.humusoft.cz/matlab/details/

[12] Microsoft Excel. *Microsoft* [online]. [cit. 2021-03-13]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/excel

[13] HAVRLANT, Lukáš. *Matematika.cz* [online]. Nová média [cit. 2021-03-13]. Dostupné z: http://www.matematika.cz/

[14] ČÁPKA, David. Základní konstrukce jazyka C# .NET: Online kurz. *ITnetwork* [online]. [cit. 2021-03-13]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady

[15] KŘIVÁNEK, Pavel. Statická vs. dynamická typová kontrola. *ROOT.CZ* [online]. 24. 6. 2004 [cit. 2021-03-13]. Dostupné z: https://www.root.cz/clanky/staticka-dynamicka-typova-kontrola/

[16] Cykly. *Programování: Gymnázium Nový Jičín* [online]. [cit. 2021-03-13]. Dostupné z: http://programovani.gnj.cz/visual-basic-6-0/cykly

[17] LASÁK, Pavel. *Jak na Excel: Ať pracuje za vás* [online]. [cit. 2021-03-13]. Dostupné z: https://office.lasakovi.com/excel/

[18] *Nápověda a výuka pro Excel* [online]. [cit. 2021-03-13]. Dostupné z: https://support.microsoft.com/cs-cz/excel

[19] TROJOVSKÁ, Eva. *Základy numerické matematiky.*