Univerzita Hradec Králové

Fakulta informatiky a managementu

DIPLOMOVÁ PRÁCE

2023 **Alex El Gharred**

Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Porovnání možností a účinnosti prostředků MS Excel a C# při řešení soustav lineárních rovnic

Diplomová práce

Autor: Alex El Gharred

Studijní program: N1802 Aplikovaná informatika

Studijní obor: ai2-p

Vedoucí práce: Slabý Antonín prof. RNDr. PhDr. CSc.

Hradec Králové leden 2023

Prohlášení

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

V Hradci Králové dne 16. března 2023 Alex El Gharred

Anotace

EL GHARRED, Alex. *Porovnání možností a účinnosti prostředků MS Excel a C# při řešení soustav lineárních rovnic*. Hradec Králové, 2023. Diplomová práce na Fakultě informatiky a managementu Univerzity Hradec Králové. Vedoucí bakalářské práce Antonín Slabý. 53s.

Tématem této diplomové práce je porovnání možností využití kancelářského softwaru MS Excel a programovacího jazyka C# v oblasti řešení soustav lineárních rovnic za pomoci numerických metod. Jakožto příklady jsou zde vybrány dvě eliminační metody, konkrétně Gaussova a Jordanova, a dvě eliminační metody, konkrétně Jacobiho a Gauss-Siedelova. Všechny zmíněné metody jsou zde ve stručnosti představeny a následně algoritmizovány a implementovány jak v prostředí MS Excel, tak v prostředí programovacího jazyka C#. V obou případech je zvolený postup dopodrobna rozebrán a vysvětlen včetně uvedení důvodu volby konkrétních kroků. Zmíněny jsou zde též výhody a nevýhody jednotlivých typů metod i přístupů implementace.

Klíčová slova

algoritmus, MS Excel, numerické metody, soustava lineárních rovnic, C#, programování

Annotation

EL GHARRED, Alex. *Comparison of the capabilities and effectiveness of MS Excel and C# resources when solving systems of linear equations*. Hradec Králové, 2023. Diploma thesis at Faculty of informatics and management, University of Hradec Králové. Thesis supervisor Antonín Slabý. 53s.

The topic of this diploma thesis is a comparison of the possibilities of using the MS Excel office software and the C# programming language in the area of solving systems of linear equations using numerical methods. Two elimination methods, namely Gauss and Jordan, and two elimination methods, namely Jacobi and Gauss-Siedel, are selected here as examples. All the mentioned methods are briefly presented here and subsequently algorithmized and implemented both in the MS Excel environment and in the C# programming language environment. In both cases, the chosen procedure is analyzed and explained in detail, including the reason for choosing specific steps. The advantages and disadvantages of individual types of methods and implementation approaches are also mentioned here.

Keywords

algorithm, MS Excel, numerical methods, linear equations system, C#, programming

Obsah

Úvod

Cílem této diplomové práce je seznámit čtenáře s různými možnostmi řešení soustav lineárních rovnic za pomoci numerických metod. Vybrané metody budou pro srovnání implementovány jak s pomocí softwaru Microsoft Office Excel, tak za pomoci programovacího jazyka C# za využití frameworku Xamarin Forms. Za účelem usnadnit čtenáři pochopení samotné práce s těmito nástroji zde bude uvedena i samotná algoritmizace problému.

K volbě tohoto tématu mi byl motivací fakt, že výuka numerických metod často probíhá za pomoci vysoce specializovaných nástrojů, jakými jsou například programovací jazyky Mathematica nebo MatLab. Tyto nástroje jsou pro odborníky v dané oblasti více než vhodné, nicméně studenti se ve většině případů s numerickými metodami střetávají spíše zběžně a nutnost osvojit si krom samotného matematického principu i znalost speciálního software může jejich pochopení daného problému znesnadňovat.

Naproti tomu se nástroje využité v této práci obecně těší větší míře porozumění. Zatímco v případě jazyka C# se jedná především o obory zaměřené na informatiku, MS Excel je snadno dostupnou alternativou, se kterou bývají studenti obeznámeni již z nižších stupňů. Ač jsou jeho možnosti ve srovnání s programovacími jazyky omezené, nabízí o mnoho snazší možnosti implementace, pochopitelné i pro osoby pohybující se mimo IT sféru. V rámci práce se budu snažit o přiblížení obou metod takovým způsobem, aby byly pochopitelné nejen pro vyučující, ale případně i pro samotné studenty.

Rozdělení práce odpovídá požadavku na prvotní seznámení s teoretickou částí problémů, ve které budou čtenáři seznámeni s pojmem numerických výpočtů jako takovým, s různými formami využití a vhodnými softwary. Též zde budou čtenáři seznámeni s matematickým principem jednotlivých metod, jenž by jim měl pomoci porozumět využitému mechanismu, a chybět nebude ani vysvětlení všech potřebných pojmů.

Praktická část pak představí jednotlivé metody řešení soustavy n lineárních rovnic o n neznámých, přičemž v případě MS Excel bude n nastaveno na hodnotu čtyři. Pro každou z použitých metod bude nejprve představena algoritmizace, jež problém rozdělí do jednotlivých kroků, v rámci nichž bude postupně vyřešen. Účelem této algoritmizace je především přiblížení logických kroků, kterých bude využito v následné implementaci.

Dále budou čtenáři seznámeni se samotnou implementací jak za pomoci programovacího jazyka C#, tak kancelářského softwaru MS Excel. Ve druhém případě též přistoupím k pár doporučením ohledně rozložení výpočtů v tabulce a úpravy formátování za účelem vytvořit přehledný vizuální dojem. V důsledku bude využito stejného principu ve dvou různých prostředích, jejichž výhody a nevýhody budou v závěru shrnuty.

Výsledným cílem je ukázat některé možnosti využití numerických metod i za pomoci méně specializovaných prostředí, u kterých je navíc vyšší šance, že se v nich studenti již orientují, a může tak odpadnout potřeba seznamovat je s novými, do té doby neznámými nástroji.

1 Úvod do problematiky

1.1 Numerické metody

Pro začátek by bylo nejlepší objasnit, co se rozumí pojmem numerická matematika. Jedná se o vědní disciplínu, jejímž cílem je vývoj a analýza metod, které fungují na principu manipulace s čísly. Numerickou metodou pak nazýváme postup, jež se k řešení numerických úloh využívá. V praxi se numerické metody používají za účelem aproximace (přibližného výpočtu) matematických modelů, které by bylo buď to nepřiměřeně složité, či dokonce nemožné řešit analyticky. [1]

Pokud řešíme matematické problémy, které mají svůj původ v reálném světě, dochází velmi často k nutnosti popsání zkoumaného fenoménu za pomoci věrohodného modelu, jež je poté zapotřebí správně vyřešit. Zmíněné modely jsou nezřídkakdy velice komplikované, často popsané pomocí algebraických nebo diferenciálních rovnic. V dnešní době naštěstí disponujeme výhodou v podobě výkonných počítačů, v jejichž moci je problém vyřešit v krátkém časovém měřítku. [1]

Mezi termíny, které je třeba zde vysvětlit, patří také algoritmus numerické metody. Tento pojem by se dal definovat jako přesný popis kroků, za pomoci kterých se realizuje výpočet numerické úlohy. Předešlý popis lze vyjádřit také jako „posloupnost akcí, které k přesně specifikovanému souboru vstupních dat přiřadí příslušný soubor dat výstupních.“ [2]

Za předzpracování, neboli preprocessing, označujeme přípravu rozsáhlých kolekcí vstupních dat. Naopak následné zpracováním, neboli postprocessing, představuje nutnost úpravy výsledných dat takovým způsobem, aby pro uživatele byla čitelná, což mu umožňuje jejich následné vyhodnocení. Výsledky totiž často nabývají značného rozsahu, z čehož pro člověka plyne problém, který je třeba za pomoci postprocessingu vyřešit. Jakožto příklad můžeme uvést vizualizaci výsledku. [3] [4]

Za korektní úlohy můžeme označit takové úlohy, jejichž řešení je jednoznačné a kompletně závislé na vstupních datech. Nicméně i korektní úloha může ve výsledcích vykazovat značné chyby, k čemuž v praxi často dochází, a to i v případě minimální změny vstupních dat. Tyto úlohy označujeme jako špatně podmíněné. „Jejich řešení je velmi problematické, v případě, že jejich vstupní data obsahují chyby (např. chyby měřících zařízení, chyby v zaokrouhlování a pod.).“ [2] Pokud nepatrná změna ve vstupních datech zapříčiní také jen nepatrnou změnu v řešení, znamená to, že je úloha dobře podmíněná. [5]

Na praktické využití numerického modelu narazíme například v předpovídání počasí po celém světě. Numerický model umožňuje krom předpovědi konkrétní podoby počasí (slunečno, zataženo, deštivo a pod.) například i výpočet předpokládané trasy hurikánu, díky čemuž je možné včas varovat před jeho ničivými následky. V meteorologii numerické modely rozlišujeme na globální a lokální, s tím že z lokálních modelů posléze vycházejí i výsledky modelů globálních. Forma predikce numerického modelu atmosféry je nicméně simplifikovaná a hrozí tudíž riziko nepřesností, což je během interpretace dat třeba brát v potaz. Právě proto se předpověď počasí někdy ukáže jako chybná a nelze na ni tedy úplně spoléhat. [5]

Existují již předprogramované programy a balíčky programů, které lze použít pro výpočet rozličných standardních úloh. Abychom je však mohli správně využít, je zapotřebí orientovat se v numerických metodách, jež plánujeme k našim výpočtům použít, neboť užívání těchto programů bez patřičných znalostí může vést spíše ke zmatení, či přinejmenším k nepřesným nebo dokonce chybným výpočtům. [6]

1.1.1 Chyby při numerických výpočtech

V případě řešení reálných problémů za pomoci numerických metod dostáváme téměř vždy pouze přibližné výsledky s jistou mírou chybovosti, neboť dosažení skutečně přesných výsledků je velice náročné. Při práci je proto zapotřebí se zaměřit na získání co nejpřesnějšího výsledku a minimalizaci celkové chyby. Získaný výsledek může být vlivem přítomnosti chyb ovlivněný, či dokonce kompletně znehodnocený. [4]

Jako ve všech oborech informatiky není vhodné brát na lehkou váhu lidské chyby, které i zde představují nezanedbatelný faktor. Chyby mohou vzniknout vlivem lidské nepozornosti, nedbalosti, případně úplným nebo částečným nepochopení zadaného problému.

Při vytváření matematického modelu popisujícího reálný problém pokaždé dochází k jistým idealizacím, neboť skutečnost je příliš složitá, než aby ji dokázal jakýkoliv matematický model přesně zachytit. Jestliže dojde k výskytu chyb v této idealiazaci modelu, případně ve vstupních datech, výstup tím bude samozřejmě ovlivněn a řešení idealizovaného modelu se bude od řešení skutečné podoby problému lišit. Tento rozdíl nazýváme chybou matematického modelu. [1]

V důsledku chybně zvolené metody pro danou úlohu dochází k takzvané chybě numerické metody, která se vyznačuje řešením nedosahujícím přesnosti, jakou bychom potřebovali. Již během návrhu numerické metody, jejíž použití plánujeme, je proto zapotřebí provést odhad chyby numerické metody. [1]

Jelikož číselné proměnné používané počítači disponují jen omezeným počtem číslic, který sice některé typy proměnných s pohyblivou desetinnou čárkou navyšují, nicméně se od tohoto limitu nikdy zcela neoprostí, vždy máme při práci s počítačem k dispozici pouze přibližné hodnoty daných čísel, vzniklé zaokrouhlením přesných hodnot. Jedná se o zaokrouhlovací chyby, jež vznikají jak při samotném zadávání dat do přístroje, tak pro samotných číselných výpočtech. [6]

V případě iteračních metod tedy vzniká s každou iterací určitá nepřesnost, která v některých případech dokáže výsledek kompletně znehodnotit. Dochází k tomu zejména když pracujeme s velmi malými čísly blížícími se nule, které v určitém momentě počítač zaokrouhlí na nulu, což v některých případech úplně znemožní provádění dalších iterací, jindy jen způsobí návrat chybného výsledku. [3]

1.1.2 Numerické metody ve výuce

par Stejně jako v případě většiny jiné látky náležející k jakémukoliv oboru jsou k dispozici různé prostředky využitelné během výuky numerických metod. Jako příklad může posloužit:

- Standardní prezenční přednáška či frontální výuka

- Webová přednáška s možností klást dotazy, jež ve většině případů velmi dobře supluje prezenční výuku a usnadňuje tak přístup k učivu studentům ze vzdálenějších lokací

- Samostudium s pomocí webových materiálů či tištěných skript, vhodné pro studenty, jimž nejvíce vyhovuje individuální tempo a možnost řídit si výuku sami

- Webové samostudium kombinované s následnou diskusí ve třídě, které doplňuje předešlou formu o možnost prodiskutovat s ostatními vše potřebné a případně si nechat poradit, či naopak poradit někomu ze spolužáků.

V posledních letech bylo provedeno významné množství výzkumů týkajících se zkvalitnění výuky napříč různými obory, včetně těch, v jejichž rámci se během studia i v praxi využívají numerické metody. Mezi tyto obory patří například přírodní vědy, informatika, matematika a strojírenství.

Výzkumem prošlo mimo jiné i možné uplatnění distanční výuky, mezi jejíž jednoznačné výhody patří schopnost poskytnutí různých forem výuky, jež využívají rozličné zdroje a strategie, což jí umožňuje pokrýt širší záběr zájmů a potřeb jednotlivých studentů. [1] [4]

Vždy je třeba mít na paměti nutnost adaptovat úroveň i složitost výuky stávající míře znalostí a schopností studentů, a to jak v numerických metodách, tak u všech ostatních disciplín. Výuka by si především měla klást za cíl přípravu na to, aby se dokázali flexibilně přizpůsobit novým problémům, s nimiž se setkají v profesním životě, namísto stále ještě oblíbeného memorování dat či pevně zadaných postupů krok za krokem, které jim sice zajistí dobré známky, nicméně je v praxi ponechá napospas dosud neznámým problémům, kterým nebudou schopni čelit. [1]

Abychom zvýšili pravděpodobnost, že si studenti trvale osvojí nejen znalosti, ale také potřebné schopnosti, jež jsou náplní kurzů věnovaných numerické matematice, je doporučeno zapojit interaktivní výukové moduly. Takto kromě pasivní konzumace přednášených informací dostanou studenti příležitost vyzkoušet své nové znalosti na praktických úlohách. Znalosti se tak nejen že prohloubí, ale též dojde k lepšímu rozvoji schopností nutných k samostatnému řešení problémů. [7]

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. [7]

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. [7]

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.

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.

1.1.3 Nástroje

V posledních letech došlo díky značnému rozvoji programování k vývoji velkého množství specializovaných softwarů usnadňujících práci ve všemožných oborech lidského působení, mezi něž se řadí i matematika. Mezi tyto specializované softwary patří například Mathematica, Maple a Matlab, jež díky jednoduchému programovacímu jazyku a celé řadě předprogramovaných funkcí umožňují pracovat se všemi různými druhy matematických operací, mezi něž patří například integrování, derivování, algebraické úpravy a pro tuto diplomovou práci nejdůležitější, řešení úloh numerické matematiky za pomoci odpovídajících metod. [6]

Jako první zde uvádím program Maple, neboť byl vyvinut českou společností Czech Software First s.r.o. a pro nás má tedy tu výhodu, že podporuje českou lokalizaci, díky čemuž odpadá nepříjemný problém jazykové bariéry. Jedná se o počítačový software, který se dá využít jak ve výuce matematiky, tak v její aplikaci napříč různými vědními obory. K jeho vývoji došlo v devadesátých letech minulého století . Program umožňuje provádět numerické i analytické výpočty a podporuje i jejich vizualizaci, dokumentaci a publikaci, k čemuž poskytuje uživatelsky přívětivé prostředí. Ti, jenž by si program chtěli opatřit, mají na výběr hned z několika druhů licencí, konkrétně se jedná o studentskou, akademickou, profesionální a vládní. Chcete-li si software nejprve vyzkoušet a zjistit, co všechno umí, výrobce nabízí zdarma patnáctidenní zkušební verzi, zatímco studentům nabízí dokonce třicetidenní. [8]

Dalším ze specializovaných programů je MATLAB od firmy MathWorks. Tento program poskytuje prostředí vhodné pro technické i vědecké výpočty, krom vizualizace podporuje i analýzu dat a vývoj vlastních algoritmů. Využívají jej vědci a inženýři po celém světě. Podobně jako ostatní nástroje, umožňuje i MATLAB řešení úloh a problémů z různorodých oborů, mezi nimiž můžeme uvést například strojové učení a aplikovanou matematiku. Součástí tohoto prostředí je výkonný programovací jazyk, vhodný pro tvorbu algoritmů i numerické výpočty, a tisíce zabudovaných nejen matematických funkcí. Taktéž patří mezi komerční produkt s několika druhy licencí, nabízí však i zkušební verzi na třicet dní. Mezi jeho nevýhody patří absence podpory českého jazyka. [9] [10]

Posledním specializovaným nástrojem, který zde zmíním, je program Mathematica od firmy Wolfram Research, vyvíjený již po třicet let. Jedná se o nejznámější software k výpočtu analytických i numerických výpočtů a vizualizaci dat na světě. Dá se k němu velmi snadno přistupovat na webu a používat ho přímo v jakémkoliv webovém prohlížeči. K dispozici je

i desktopová varianta pro každý moderní operační systém. Stejně jako předešlé dva, je i tento program komerční. Desktopovou verzi je však možné získat za jednorázovou částku, zatímco webovou verzi je bohužel třeba obnovovat každý měsíc nebo rok v závislosti na volbě uživatele. Existuje i patnáctidenní zkušební verze, která je zdarma. Obdobně jako MATLAB, ani Mathematica nenabízí českou lokalizaci. [11] [12]

Vedle specializovaných prostředí se k numerickým výpočtům dají využít i jiné nástroje, mezi něž patří i tabulkový kalkulátor MS Excel. Tento kancelářský nástroj uživateli umožňuje pracovat s daty seřazenými do tabulkového formátu, a obsahuje celou řadu dalších užitečných funkcí. V základu je Excel součástí kancelářského balíčku Microsoft Office, v jehož rámci se vedle Wordu a PowerPointu jedná o jeden z nejpoužívanějších programů. Existuje hned několik časově odlišených verzí zmíněného balíčku. Úplně nejnovějším produktem z této řady je MS Office 365, který umožňuje sdílené využití například pro firemní tým nebo školu, a usnadňuje tak synchronizaci práce. Na druhou stranu je ale k dispozici pouze časově omezená licence, již je nutné v pravidelných intervalech obnovovat. [13]

K velkému zlomu, co se podoby uživatelského prostředí týče, došlo mezi verzemi 2003 a 2007, kdy právě MS Office 2003 je poslední verze s klasickým vzhledem, který byl v následující verzi kompletně změněn, ať již k libosti či nelibosti uživatelů. MS Excel je kompletně přeložený do českého jazyka, díky čemuž umožňuje pohodlnou práci i našinci. Předposlední verze produktu je možné zakoupit za jednorázovou částku, ty starší se pak shánějí hůře, buďto ještě vypálené na instalačních CD, nebo pochybných webových stránkách. Pro novější produkty nabízí Microsoft třicetidenní zkušební verzi. Excel má však i svou open source variantu, Libre Office Calc, jejíž uživatelské prostředí je mnohem bližší klasické verzi Excelu. Vzhledem k tomu, že nejde o software specializovaný na numerické výpočty, je při tvorbě postupů zapotřebí být vynalézavý a vzít v úvahu možnosti a omezení nástroje. [14]

Nejuniverzálnější, avšak také nejspíše nejobtížnější možností, jak přistupovat k tvorbě algoritmů vhodných pro řešení numerických metod je využití obecného programovacího jazyka. Při volbě tohoto postupu získáme jistou volnost, s ní ale i nutnost ošetřit všechny možné výjimky, abychom tak předešli zacyklení nebo pádu programu při chybovém hlášení. Mezi výhody této volby patří i absence nutnosti vázat se na jediný či úzce vymezený software, vývojářských nástrojů je totiž k dispozici celá řada, mezi nimiž se nachází i spousta nekomerčních řešení. Příkladem budiž Visual Studio, známý nástroj, jenž je v základní verzi Community zcela zdarma.

1.2 Definice pojmů

1.2.1 Matematika

Lineární rovniceje typ rovnice, v níž se nachází pouze jedna mocnina, a to jen ve své první mocnině. S pomocí úprav je možné rovnici převést na takzvaný základní tvar, jehož obecný vzorec je $ax + b = 0$, přičemž x představuje neznámou a znaky a a b libovolná reálná čísla. Člen ax označujeme jako lineární, zatímco člen b jako absolutní. [15]

Soustava lineárních rovnic je definována jakožto množina m lineárních rovnic o n neznámých. Když soustavu vypočítáme, pro všech n neznámých tak najdeme jejich hodnotu. Správnost výpočtu se pozná tak, že při dosazení těchto hodnot do původní soustavy budou všechny obsazené rovnice dávat smysl. Podobně jako u jednoduchých lineárních rovnic, i neznámé v soustavě rovnic mohou nabývat různých počtů řešení, a to buď to žádné, jedno jediné, nebo nekonečně mnoho. Každý z těchto výsledků podléhá určitým zákonitostem, které je ze soustavy možno zjistit ještě před samotným výpočtem a ušetřit si tak práci. I tak je ale otázka nalezení počtu řešení a v případě, že je jen jedno, jeho konkrétní hodnotu, poněkud složitější než u rovnice jednoduché. [15]

Soustava lineárních rovnic se dá obecně vyjádřit v tomto tvaru:

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

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

Matice je pojem označující schéma obdélníkového či čtvercového tvaru, jenž obsahuje zpravidla číselné hodnoty. Tyto hodnoty indexujeme v závislosti na jejich postavení v matici typu $m \times n$ takto: $a_{ij}\,(i \in \{1, 2, ..., n\}$ a $j \in \{1, 2, ..., m\})$. Čísla m a n označují velikost matice, přičemž m značí počet řádků a n počet sloupců. Pokud je řeč o základních maticích soustav lineárních rovnic, v naprosté většině případů platí, že $m = n$. V případě matic rozšířených, tedy obsahujících i absolutní člen, je to $n = m + 1$. Sloupec absolutních členů, neboli pravých stran rovnic, se indexuje $b_i\,(i \in \{1, 2, ..., n\})$, přičemž se od koeficientů lineárních členů odděluje svislou čarou. [15]

Maticové vyjádření výše popsané soustavy m rovnic o n neznámých by vypadalo takto:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{14} & b_1 \\ a_{21} & a_{22} & \ldots & a_{24} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{n4} & b_n \end{array}\right)$$

Ekvivalentní úpravy nazývané též jako řádkové úpravy soustavy rovnic umožňují upravovat soustavu do potřebného tvaru, aniž by změnili její platnost. Účelem těchto úprav je dostat soustavu do jiného, často nějakým způsobem zjednodušeného tvaru, z něhož se snáze získává výsledek. Nejčastější bývá úprava na tzv. horní trojúhelníkový tvar.

Mezi ekvivalentní úpravy patří:

1. Prohození pořadí libovolných řádků
2. Vynásobení jedné rovnice jakýmkoliv nenulovým reálným číslem k
3. Přičtení k-násobku jedné rovnice soustavy k jiné rovnici soustavy, kde $k \neq 0$

[15]

Hlavní diagonála matice je tvořena všemi členy a_{ij}, kde $i = j$. Takové členy často zapisujeme jako a_{ii}.

Trojúhelníkový tvar matice se vyznačuje tím, že hodnota všech prvků pod nebo nad hlavní diagonálou je rovna nule. Konkrétně při numerických výpočtech se často využívá horní trojúhelníkový tvar matice, jenž má nulové prvky pod hlavní diagonálou.

Lineární kombinace n vektorů nebo řádků x_1 až x_n můžeme definovat jako sumu libovolných násobků těchto řádků/vektorů. Obecný zápis lineární kombinace vypadá následovně: $v = k_1x_1 + k_2x_2 + ... + k_nx_n$, kde k_i $(i \in \{1, 2, ..., n\})$ je libovolné reálné číslo.

Lineárně nezávislé jsou řádky či vektory, z nichž žádný nelze zapsat jako lineární kombinaci těch ostatních.

Hodnost matice je rovna počtu řádků této matice, jež jsou lineárně nezávislé.

Řídící prvek řádku je první prvek umístěný nalevo, jenž má zároveň nenulový koeficient.

Jednotková matice je taková čtvercová matice, která obsahuje samé jedničky na hlavní diagonále, zatímco na všech ostatních pozicích obsahuje nuly.

Redukovaný trojúhelníkový tvar tvar matice vychází z horního trojúhelníkového tvaru, liší se však v tom, že všechny řídící prvky v matici se rovnají jedné, a zároveň každý sloupce, v němž se nachází některý z řídících prvků, má všechny ostatní prvky nulové.

Absolutní hodnota čísla je určena pravidlem, že pokud je výchozí číslo x větší nebo rovné nule, jeho absolutní hodnota je rovna témuž číslu x. Pakliže je ale jeho výchozí hodnota záporná, je jeho absolutní hodnota rovna hodnotě opačné, a sice $-x$. Bez výjimky se tedy vždy jedná o číslo nezáporné. Absolutní hodnotu čísla x značíme jako $|x|$. [15]

Determinant je pojem spadající pod lineární algebru. Jedná se o zobrazení, které přiřadí každé čtvercové matici A číslo, jež označujeme jako $|A|$ nebo $det\, A$. První označení se od absolutní hodnoty liší použitím velkého písmene, kterým označujeme matici, zatímco malým písmenem značíme obyčejnou číselnou neznámou.

Permutace n-prvkové množiny je definována jako uspořádaná n-tice obsahující každý z prvků původní množiny právě jednou. Jedná se tedy o jisté přeuspořádání množiny. Počet všech permutací na n-prvkové množině je dán za pomoci vztahu $P(n) = n!$, kde $n!$ představuje matematické značení pro n faktoriál, což je struktura definovaná jako součin všech přirozených čísel menších nebo rovných číslu n. [15]

1.2.2 Informatika

Bit představuje základní jednotku kapacity paměti, a to nejen virtuálně, ale i fyzicky. Může nabývat pouze logických hodnot pravda a nepravda, v binární soustavě reprezentovaných jako 1 a0. Fyzicky se jedná o místo, jímž buď to prochází proud (1) nebo neprochází (0). Označován bývá buď to přímo jako *bit*, či zkráceně písmenem *b*.

Byte představuje základní jednotku užívanou při alokaci počítačové paměti a udávání objemu počítačových dat. Jedná se o jednotku tvořenou osmi bity, což dohromady dává prostor pro všechna osmiciferná binární čísla. V dekadické soustavě se jedná o hodnoty 0 až 255, tedy celkem 256 čísel. Byte je obvykle nejmenší možný objem dat, se kterým

dokáže počítač pracovat, a tudíž jej přiřazuje i proměnným typu logické hodnoty, kterým by z praktického hlediska stačil jeden bit. Aby se odlišil od bitu, označuje se byte velkým písmenem *B*.

Proměnná je datový objekt, jenž je deklarován v programu a jeho hodnota se za běhu programu měnit může měnit. Při deklaraci se přiřazuje pouze název a typ proměnné, zatímco definice již přiřazuje blok paměti pro zápis. Do tohoto bloku si v průběhu program zapisuje data, s nimiž může posléze pracovat. Jméno proměnné je řetězec obsahující libovolná písmena bez diakritiky, mimo první místo řetězce je možné používat i číslice a některé speciální znaky, jako například pomlčku nebo podtržítko. Je zvykem názvy proměnných zapisovat s malým počátečním písmenem, čímž se odlišují od názvů metod či tříd. Některé programovací jazyky název proměnné uvádí znakem $ (dolar), jenž není oddělen mezerou. [16]

Datový typ představuje třídu objektu určující druh a rozsah hodnot, kterých může daná proměnná nabývat, stejně tak i možnosti a omezení jejího využití. Každá taková třída specifikuje určité vlastnosti, mezi než patří kupříkladu velikost a struktura proměnné. Dále obsahuje předprogramované metody, které lze na konkrétní instanci dané třídy aplikovat, a v některých případech i atributy objektu přisouzené, k nimž lze poté přistupovat zvlášť. Většina metod, mezi něž patří i matematické operace, má určeno tzv. přetížení, což značí, s jakými datovými typy dokáže pracovat, jako příklad může posloužit pokus od sebe odečítat dvě proměnné obsahující řetězec znaků. U operací určených matematickými znaménky namísto metod stojí za zmínku operace plus, která v případě některých jazyků dokáže posloužit i k řetězení textových proměnných.

Datové typy v základu dělíme na jednoduché a složené. Složené datové typy jsou, jak již název napovídá, tvořeny skládáním jednoduchých typů do různých struktur, mezi něž patří například pole. V některých jazycích existují různé varianty polí, vzájemně se odlišujících chováním. Dále sem patří složitější struktury, jakými jsou např. slovníky. Ačkoliv základní datové typy má naprostá většina jazyků společné, někdy se pojmenování může lišit a je tedy vhodné při přechodu na jiný jazyk toto názvosloví zkontrolovat.

Nejjednodušší základní datový typ je logická hodnota, obvykle značená `bool` či `boolean`. Jedná se o binární proměnnou, jež může nabývat pouze dvou různých hodnot, a to buď pravda nebo nepravda. Ačkoliv technicky vzato tato proměnná nabývá pouze velikosti jednoho bitu, bývá ukládána jako jeden byte, neboť počítač menší blok paměti nedokáže přiřadit.

Jedním z nejdůležitějších datových typů je celé číslo, nazývané též `integer`, v základní velikosti 4 byte značené `int`. Existuje však i mnohem více variant, jež se dělí jednak podle velikosti (1 byte, 2 byte, 4 byte a 8 byte), a jednak podle toho, zda může číselná hodnota obsahovat znaménko. Proměnná se znaménkem může v oboru přirozených čísel dosáhnout pouze poloviční velikosti oproti stejně veliké proměnné bez znaménka, neboť polovina oboru jejích hodnot leží pod nulou, což v praxi znamená, že na určení znaménka je potřeba právě jeden bit.

Mezi jedny z nejčastěji používaných patří též datové typy zastupující reálná čísla. Nejznámějšími zástupci jsou typy s pohyblivou desetinnou čárkou, tedy `float` a `double`,

přičemž `double` může díky dvojnásobné velikosti nabývat mnohem větší přesnosti. Dalším rozšířením je typ `long double`, který přesnost ještě zvyšuje, a typ `decimal`, který je díky odlišné konstrukci vhodný pro kalkulace s financemi, neboť se snaží eliminovat ztráty. Bez ohledu na velikost má však každý z těchto typů omezenou přesnost a nedokáže tudíž uchovat čísla s nekonečným nebo dostatečně velkým desetinným rozvojem.

Krom číselných typů existuje i například datový typ `char`, který je specifický tím, že v paměti dokáže uchovat právě jeden znak, a to ve formě číselné hodnoty, kterou je daný znak v počítači reprezentován. Datový typ `string`, představující řetězec znaků, je sice standardně řazen mezi jednoduché datové typy, avšak z praktického hlediska se jedná o pole proměnných typu `char`, a je s ním v mnoha případech tudíž možné jako s polem pracovat.

Staticky typované programovací jazyky jsou typické tím, že vyžadují explicitní uvedení datového typu deklarované proměnné, přičemž po deklaraci je tento typ programově neměnitelný. Díky tomuto opatření je v každém okamžiku běhu programu jasné, jakého datového typu nabývá proměnná, se kterou se zrovna pracuje. Výhodou programů psaných ve staticky typovaných jazycích je velká bezpečnost. Nelze opomenout i zjednodušení typové kontroly, kterou není nutné provádět při každém spuštění programu, neboť se případné typové chyby ze strany programátora odhalí již při kompilaci. Díky tomu je chod programu efektivnější, v důsledku toho i méně náročný na paměť a může tudíž běžet rychleji. Stále je však nutné ošetřit vstupy od uživatele tak, aby se zajistilo, že se program nepokusí přiřadit do proměnné hodnotu špatného typu. Známými staticky typovanými jazyky jsou kupříkladu C, C++, C# nebo Java. [16] [17]

Dynamicky typované programovací jazyky naproti tomu při deklaraci proměnné nepožadují, aby programátor specifikoval datový typ. Ten se nastavuje dynamicky až když je do proměnné přiřazena hodnota, která se může i během výkonu programu přepsat hodnotou jiného typu. V takovém případě dokáže jazyk sám rozeznat druh hodnoty a přiřadit podle toho datový typ, který přepíše typ původní. Při využití dynamické typové kontroly je zapotřebí menší množství kódu, což umožňuje celkově rychlejší vývoj.

Existují i jazyky, mezi něž patří například PHP, dokážou datový typ proměnné i změnit v případě provádění operace vyžadující operandy určitého datového typu. K tomu může dojít třeba pokud programátor zadá číselnou hodnotu, ale ohraničí ji uvozovkami a bude tudíž uložena jako `string`. Pokud se takovou hodnotu pokusí použít v matematické operaci, PHP se automaticky pokusí přetypovat na `int` či `float`. Nevýhodou dynamicky typovaných jazyků je obtížnější odhalování typových chyb, kde je nutné provést kontrolu při každém spuštění programu, což program zpomalí. Další nevýhodou může být i horší přehlednost a vyšší obtížnost na uhlídání vstupů od uživatele. Jakožto příklady dynamicky typovaných jazyků můžeme uvést PHP, Python nebo Javascript. [16] [17]

Index označuje umístění prvku v poli, zpravidla se jedná o nezáporné celé číslo. Standardně čím vyšší číslo, tím je prvek v poli umístěn dále. Přístup programovacích jazyků k indexování prvků v poli však není úplně jednotný. Nejtypičtější a v praxi nejpoužívanější je indexování začínající nulou, které možná odporuje intuitivnímu vnímání pořadí, nicméně v informatice má své opodstatnění – při vynásobení indexu velikostí datového typu zjistíme,

o kolik je daný prvek posunut v paměti od začátku pole. Tento přístup využívají jazyky C, C++, C#, Java, PHP, Python a další.

Jiné jazyky, mezi něž patří například BASIC, se naopak drží toho, jak běžný člověk vnímá pořadí, a činí tak práci s poli více intuitivní a také bližší matematickému značení. Existují i jazyky, v nichž si může programátor nastavit horní i dolní mez pole individuálně. Mezi ně se řadí např. Visual Basic nebo Pascal. V případě pole nazvaného `pole` k prvku uloženému na indexu `i` zpravidla přistoupíme s pomocí zápisu `pole[i]`.

Takzvaná asociativní pole, podporovaná například jazykem PHP, umožňují místo číselného indexu zadat řetězec znaků, který pak používáme, pokud se na nějaký prvek pole chceme odvolat. V C# existuje struktura slovník, která funguje na podobném principu. [16]

Pole je jedním ze základních prvků vyšších programovacích jazyků. Tato datová struktura sdružuje určitý, vždy konečný počet prvků, do jedné složené proměnné. Ekvivalentem pole je v matematice konečná množina. V některých programovacích jazycích, jako např. C# nebo Java, je počet prvků v poli pevně zadaný při deklaraci a dále už ho není možné měnit. Naopak třeba v PHP nebo Pythonu je velikost pole proměnlivá, a je tudíž možné přidávat nové prvky či mazat ty staré. V takovém případě dokáže pole posloužit jako fronta nebo zásobník. C# definuje datovou strukturu odpovídající této koncepci pole, nazývá se zde však `List` a spadá pod struktury typu `Kolekce`.

Ve staticky typovaných jazycích je poli při deklaraci přiřazen i datový typ, jenž je poté jediným typem, jakého mohou hodnoty v poli nabývat. V případě dynamicky typovaných jazyků je zpravidla možné, aby pole obsahovalo více různých datových typů najednou, a to včetně dalších proměnných typu pole. Pole je takto možné víceúrovňově vnořovat, čímž vznikají tzv. vícerozměrná pole. K jednotlivým prvkům pole se přistupuje pomocí indexu. [16]

Vícerozměrná pole fungují na podobném principu, jako pole jednorozměrná, avšak se liší jejich uspořádání. Zatímco jednorozměrné pole si lze představit jako jediný řádek hodnot, čili jsou uspořádány pouze v rámci jediného rozměru a jejich index značí pouze polohu v jediném směru, vícerozměrná pole pro indexování a tudíž i uspořádání prvků využívají více rozměrů. Poloha prvků v každém z n směrů se pomocí indexu zapisuje do uspořádané n-tice, jako např. [2, 0, 8] – v tomto případě se jedná o uspořádanou trojici a tedy i trojrozměrné pole.

V této diplomové práci se bude pracovat především s dvourozměrnými poli, které lze matematicky reprezentovat za pomocí matice, a je možné si ho představit i jako tabulku. Ne každý programovací jazyk má zabudovanou přímou podporu vícerozměrných polí, tzn. možnost pole jako vícerozměrné přímo deklarovat. Příkladem takového jazyka je PHP, které vícerozměrná pole vytváří s využitím vnořování, čili tzv. pole polí.

V praxi to funguje tak, že se deklaruje jednorozměrné pole, představující řádek, do něhož se poté přímo vkládají další pole jako jeho prvky, které by v tomto případě představovaly sloupce. Takto lze postupovat i na více úrovních. S pomocí této struktury lze dosáhnout obdobných výsledků, avšak je zapotřebí mít na paměti, že funguje trochu odlišně a tudíž je nutné tomu přizpůsobit kód. Souřadnice v poli polí se nezapisují za pomoci uspořádaných

n-tic, ale zapisuje se každá zvlášť, ohraničená hranatými závorkami, takto: [2] [0] [8], kde se pořadí jednotlivých indexů odvíjí od toho, jak jsou do sebe pole zanořená. [16]

Podmínky Podmínky známé též pod českým názvem větvení, případně podmíněný příkaz, se řadí mezi základní řídící struktury programu. Představují prostředek, jímž se v programovacím jazyce zajišťuje možnost, aby se program choval jinak v závislosti na tom, zda je či není splněna určitá podmínka. Tuto podmínku program nejprve vyhodnotí jako pravdivou nebo nepravdivou, následkem čehož buď provede příkaz či sérii příkazů závislých na daném závěru, nebo neprovede nic a program pokračuje dál.

Podmínka samotná se v naprosté většině programů značí pomocí klíčového slova `if` (česky *když*). Je-li podmínka splněna, provede se blok kódu přiřazený tomuto příkazu. Pokud chceme při nesplnění podmínky vykonat jiný blok kódu, je možné doplnit na konec příkaz `else` (česky *jinak*), jemuž tento alternativní blok přiřadíme. Oba tyto bloky kódu se ve většině programovacích jazyků uzavírají do složených závorek, aby byly jasně ohraničeny. Pokud blok obsahuje pouze jediný řádek, závorky se použít nemusí. Výjimku tvoří např. Python, jehož syntax je založena na bílých znacích a každý řádek bloku příslušného podmínce se opatří jedním odsazením navíc oproti řádku s podmínkou. [16]

Cykly jsou po boku podmínek jedněmi ze základních řídících struktur programu. Využití najdou zejména tehdy, je-li zapotřebí provádět určitý set příkazů opakovaně, ať už je počet opakování stanovený pevně a cyklus pouze ušetří zbytečné opisování téhož kódu, nebo je počet opakování závislý na proměnné, jejíž hodnota se určuje za běhu programu, a je tudíž nutné ji nastavit dynamicky, čehož by s pevně zapsaným kódem nebylo možné dosáhnout.

Pokud je ukončení opakování cyklu závislé na splnění určité podmínky, jedná se o *while* cyklus, zatímco pokud je počet opakování stanoven přímo v deklaraci cyklu, jedná se o *for* cyklus. Existuje však i zvláštní případ, jímž je cyklus foreach, jenž je specifický tím, že zadané příkazy provede zvlášť pro každý prvek zadaného jednorozměrného pole. V některých případech může cyklus foreach usnadnit práci, nicméně je třeba se mít na pozoru, neboť jeho zvláštnost tkví v tom, že na rozdíl od ostatních cyklů, které pracují s proměnnými samotnými, *foreach* pracuje pouze s kopií daného pole a neumožňuje tak jeho prvky přepisovat. Tento cyklus je vhodný pro výpis prvků či provedení operace s nimi, která samotné pole nemění. Chceme-li provádění jakéhokoliv cyklu předčasně ukončit, můžeme použít příkaz `break`. [18]

Příkaz `return` představuje klíčové slovo využívané k návratu z metody, v níž bylo specifikováno. Není tedy překvapením, že jeho český název je *návrat*. Tento příkaz lze využít k jednoduchému návratu z metody na konci provedení veškerého kódu v ní obsaženého. Taktéž ale může `return` vracet konkrétní hodnotu, kterou měla metoda získat či vypočítat, zpět do místa, odkud byla zavolána. Ve staticky typovaných jazycích je potřeba u každé metody typ návratové hodnoty specifikovat předem, zatímco v případě dynamicky typovaných jazyků může příkaz vrátit hodnotu jakéhokoliv typu.

1.2.3 MS Excel

Sešit je v MS Excel 2003 celý dokument, který má uživatel po otevření či vytvoření xls. souboru k dispozici. Standardně se skládá ze tří listů, tedy jednotlivých tabulek, které je posléze možno dle libosti přidávat či odebírat. Tuto tabulku tvoří buňky.

Buňky jsou základní jednotky tabulky, představují jednotlivá políčka tabulek a jsou určeny pomocí sloupců, značených velkými písmeny, a řádků, jež jsou pro změnu řazeny čísly. Spojením písmene sloupce a čísla řádku poté získáme označení jednotlivé buňky a můžeme se na ni jeho prostřednictvím odvolávat ve vzorcích užitých v jiných buňkách. Hodnota v buňce může být mnoha různých typů, mezi nejčastější patří text, číslo, datum, čas a procento. Může obsahovat ale též složitější konstrukce jako jsou vzorce a přeprogramované funkce, z nichž některé umí pracovat s celou oblastí buněk naráz. Jednotlivé buňky též můžeme naformátovat dle svých preferencí. [14]

Vzorec vždy zahajujeme v prázdné buňce znaménkem =, které dá Excelu vědět, že řetězec, který se chystáme zadat, je vzorec, a má s ním podle toho pracovat. Krom manuálně napsaných vzorců, které si můžeme sestavit za pomoci závorek a základních matematických operandů, existuje i celá škála přeprogramovaných funkcí, mezi něž patří nejen matematické, ale též například statistické, finanční, textové nebo logické. Mezi často používané funkce logického typu patří především KDYŽ, A a NEBO, které byly hojně využity i v rámci této práce. Pokud v průběhu zadávání vzorce došlo k chybě, vyskočí varovné okénko navrhující úpravu vzorce, případně pokud není vzorec syntakticky špatně, v buňce se objeví chybové hlášení, či úplně jiný výsledek, než jakého chtěl uživatel docílit. Chyba může ale nastat také při zadání neplatných či neočekávaných dat do některé z buněk, na něž vzorec odkazuje. Typickým příkladem takové situace je dělení nulou či textový řetězec vložený do buňky určené pro matematické operace. [14] [19]

Formát či celé vyznačené oblasti buněk se týká především vzhledových úprav, které nemají na funkci vliv a mají sloužit pouze k lepší přehlednosti. Patří sem barva a styl písma, barva pozadí buňky, barva, styl a tloušťka rámečku, a podobně. Také se sem ale řadí volba typu hodnoty v buňce zadané. Nejčastěji formát upravujeme ručně, nicméně existují i funkce umožňující formátování automatizovat za pomoci schématu předpřipraveného v samotném Excelu. Těchto schémat bývá na výběr více a je možné přidat vlastní. Zajímavou funkcí je tzv. podmíněné formátování, které umožňuje automatickou změnu formátu buňky v závislosti na jejím obsahu, či na splnění zadané podmínky, která může dosahovat již značné komplexnosti. Za pomoci podmíněného formátování si můžeme v mnohém usnadnit práci s proměnlivými daty, neboť nám pomůže je lépe vizualizovat a tudíž snáze hledat co potřebujeme, či určovat konkrétní stavy, aniž bychom museli pokaždé kontrolovat hodnoty sami. [14]

Automatické vyplňování slouží ke kopírování obsahu jedné buňky či oblasti buněk (nejčastěji části řádku či sloupce) do větší oblasti. Pro kopírování je nejprve zapotřebí zvolenou buňku či oblast označit myší, následně se v pravém dolním rohu objeví malý čtvereček, jehož přidržením a potažením přes zamýšlenou oblast tuto oblast vyplníme hodnotami. Tímto způsobem je možné vyplnit buňky jak opakující se posloupností, tak

i dokončit posloupnost započatou. Tuto funkci podporují jen některé typy hodnot, jako například číslo, datum a čas.

Doplňování je nastavené na aritmetickou posloupnost, pro niž stačí zapsat první dvě hodnoty, díky nimž získá program informaci o rozdílu, který má být mezi dvěmi sousedními buňkami. Tyto buňky se označí a vyplní směrem, kterým chceme, aby se posloupnost ubírala. Tento postup však na jiný typ posloupnosti aplikovat nelze, neboť i při zadání tří a více po sobě jdoucích hodnot postupuje algoritmus tak, že dojde k průměrnému rozdílu, který poté ke každá další hodnotě přičítá. Pokud chceme vyplnit pole opakující se posloupností, je třeba dané buňky nejprve převést na typ text, neboť na ten se snaha o aritmetickou posloupnost nevztahuje.

V případě kopírování vzorců je třeba mít na paměti, že se při automatickém vyplňování implicitně mění hodnota buňky, na kterou vzorec odkazuje, a to přesně o tolik, o kolik je nová buňka posunuta oproti té původní, horizontálně i vertikálně. Čili buňka A, v níž je odkaz na buňku C4, se při posunutí do buňky B2, odkaz změní na D5. Chceme-li odkaz zachovat v takové podobě, v jaké je, případně zachovat jen sloupec nebo jen řádek, je třeba před číslo řádku / písmeno sloupce doplnit znak dolaru - $. Chceme-li měnit jen sloupec, ale řádek zachovat, bude zápis vypadat následovně: B$2. Pro opačný případ to bude $B2.

2 Metody řešení soustavy lineárních rovnic

Základní metody pro práci s více rovnicemi o více neznámých, které zmiňuji níže, se vyučují již na střední škole, kde se nicméně jedná především o soustavy dvou, nanejvýš tří rovnic, a výpočet tudíž není tak obtížný. Pro složitější případy s vyšším počtem rovnic, s nimiž se často setkáváme v praxi či na vysoké škole, umožňují numerické metody řešení mnohem elegantnější a přehlednější. Nelze si však nepovšimnout, že jak analytické tak iterační metody v mnohém vycházejí ze stejného principu jako tyto středoškolské.

Dosazovací metoda funguje na principu vyjádření jedné neznámé z jedné rovnice, načež se za danou proměnnou provede substituce získaným výrazem v ostatních rovnicích, čímž se počet rovnic o jednu redukuje. Stejným způsobem je zapotřebí postupovat i nadále, dokud postupně nedojde k eliminaci všech neznámých krom jedné poslední, vyjádřené jednou zbývající rovnicí. Odsud již není těžké neznámou vypočítat a poté zpětně dopočítávat zbylé proměnné z rovnic, které jsme získali v předchozích krocích. podobného principu zpětného výpočtu využívá i gaussova eliminační metoda.

Abychom se při výpočtu vyhnuli zbytečně složitým kalkulacím se zlomky, je vhodné zvolit proměnnou, kterou bude možné vyjádřit co nejjednodušeji, ideálně s koeficientem jedna či minus jedna, získaný výraz tak bude přehlednější a snáze se nám s ním bude počítat. V případě jiného typu rovnic než jsou lineární je pak lepší se vyvarovat především mocninám a odmocninám, případně dalším operacím, které by nám další postup komplikovaly. Nevýhodou této metody je, že se mnohdy ani při veškeré snaze komplikovaným výrazům především v případě vyššího počtu rovnic nevyhneme. [15]

Srovnávací metoda se v jistém ohledu podobá metodě dosazovací. Rozdíl tkví v tom, že se jedna neznámá vyjádří z ostatních rovnic, a získané výrazy se mezi sebou následně porovnávají. Tento postup je vhodný především pro soustavy dvou rovnic o dvou neznámých, neboť se při vyšším počtu postup nepřiměřeně komplikuje, neboť je zapotřebí zvolit správný klíč porovnávání, a celý postup pakovat tolikrát, kolikrát je zapotřebí, než bude získána jediná zbylá proměnná. Její princip se však vzdáleně podobá metodám iteračním, kde se však z každé rovnice vyjádří jiná proměnná.

Sčítací metoda základem eliminačních metod, z nichž bude v této práci představena Gaussova a Jordanova. Princip, který mají zmíněné postupy společné, tkví v řádkových neboli ekvivalentních úpravách. V obou případech se při výpočtu zaměřujeme na eliminaci jedné proměnné ze zbylých rovnic tak, že pro zvolenou rovnici najdeme vhodný k-násobek, který odečteme od l-násobku druhé rovnice. Tyto dva koeficienty mohou být pro každou dvojici jiné. Rozdíl tkví především v tom, že zatímco eliminační metody implementují pevně daný postup, měnící pořadí řádku pouze při nesplnění podmínky, že nesmí na kritické pozici obsahovat nulový koeficient, sčítací metoda umožňuje ve volbě řádku, jejž budeme od ostatních odčítat, relativní volnost, a s tím i volbu jednodušších výpočtů. Pokud však počítáme se soustavou více rovnic, může být zápis trochu chaotický a člověk se v něm snáze ztratí. [15]

2.1 Výpočet determinantu

Všechny zde využité metody závisí na výpočtu determinantu, jehož hodnota určí, zda je zadaná matice lineárně nezávislá a má tedy smysl na ní provádět jakékoliv výpočty. Zatímco v MC Excel existuje pro výpočet determinantu vestavěná funkce, v prostředí jazyka C# musíme napsat vlastní kód. Bude tedy vhodné si výpočet determinantu vysvětlit nejprve z matematického hlediska, než přejdeme k jeho praktické implementaci.

Matematika nám definuje exaktní postup pro matice velikosti 1, 2 a 3. Pro matici velikosti 1 je to velmi jednoduché – její jediný prvek je rovněž determinantem. V případě matice velikosti 2 získáme determinant použitím jednoduchého vzorce: $a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$. Na matici velikosti tři je již zapotřebí aplikovat Sarrusovo pravidlo, jehož matematické vyjádření je již poněkud složité, nicméně stále snadno algoritmizovatelné. My však budeme pracovat s obecnou maticí, a potřebujeme tedy kód, který dokáže spočítat determinant matice nezávisle na její velikosti.

K tomu nám nejlépe poslouží tzv. Laplaceův rozvoj. Jedná se o metodu v praxi velmi dobře reprezentující tzv. rekurzivní algoritmus. Rekurzivní funkce je definována tak, že v určitém momentě volá sebe sama. Laplaceův rozvoj lze provést buď podle řádku nebo podle sloupce. V tomto případě bude využit rozvoj podle řádku. Nejprve je třeba si zvolit řádek, podle nějž budeme rozvíjet. Při manuálním výpočtu je výhodné vybrat řádek s nejvyšším počtem nulových prvků, neboť nám to velmi zjednoduší práci. Při automatizovaném výpočtu na volbě řádku nezáleží, neboť veškeré výpočty za nás provede počítač. Zde bude využit rozvoj podle prvního řádku. [20]

Laplaceův rozvoj je reprezentován pomocí následujícího vzorce:

$$det\, A \; = \sum_{j=1}^{n} a_{ij} \cdot (-1)^{i+j} \cdot subdet\, A_{ij}$$

Zatímco index i je pevně stanovený, v našem případě na hodnotě 1, index j se v průběhu součtu bude měnit. V praxi to znamená, že pro každý prvek prvního řádku se k výsledku přičte součin onoho prvku a_{ij}, -1 umocněné na součet hodnot obou indexů, a subdeterminantu matice A podle prvku a_{ij}. Subdeterminant podle prvku a_{ij} je roven determinantu matice vzniklé tak, že se z matice A vymaže řádek i sloupec, v němž je obsažen prvek a_{ij}. V tomto místě algoritmu tedy odkazujeme na provedení téhož výpočtu, jen pro matici s velikostí o jednu menší než původní.

2.1.1 Implementace v C#

Jelikož ověření hodnoty determinantu je nutné pro všechny metody, jimiž se tato práce zabývá, bude nejvhodnější si implementaci jeho výpočtu popsat zde, ještě než se budeme věnovat metodám samotným. Tato sekce bude též sloužit k seznámení se základní konstrukcí jazyka C#, alespoň v rámci nástrojů využitých v kódu. Ačkoliv aplikace (viz přílohy) vyvinutá pro tuto diplomovou práci je vytvořena ve frameworku Xamarin forms, umožňujícím tvorbu aplikace s grafickým rozhraním pro Windows, Android i iOS, budou zde uvedeny pouze ukázky z kódu mnou vytvořené třídy Matice, včetně metod na této třídě.

Struktura této třídy je velmi jednoduchá a intuitivně navazuje na matematický model. Stejně tak, jako je matice rozdělena na levou stranu představující koeficienty proměnných v jednotlivých rovnicích a pravou stranu tvořenou sloupcem absolutních členů, je i třída Matice tvořena dvěma objekty představujícími softwarovou alternativu ke zmíněným matematickým strukturám. Matice na levé straně je zde reprezentována strukturou typu dvourozměrné pole, jež je v C# podporováno přímo, a sloupec levých stran je pouze jednorozměrné pole. Obě struktury se skládají z jednoduchých proměnných datového typu float. Jelikož se jedná se o staticky typovaný jazyk, je nutné typ proměnné vždy deklarovat předem.

```
public float[,] nezname { get; set; }
public float[] konstanty { get; set; }
```

Zatímco zápis jednorozměrného pole je zřejmý, tedy nazev[index], v případě matice je zapotřebí specifikovat, která z hodnot odkazuje na číslo řádku a která na číslo sloupce. Máme-li zápis nazev[index1, index2], fakticky není pevně stanoveno, který index považovat za řádkový, neboť jsou oba rozměry totožné. Zde si jako index řádku stanovíme ten první. Stejně jako většina programovacích jazyků indexuje C# od nuly, tudíž bude index vždy o jednu menší než skutečné pořadí prvku v matici.

Jelikož aplikace umožňuje uživateli zvolit velikost matice, je krom základních dvou objektů nutné definovat metodu, která je oba vytvoří a přiřadí jim uživatelem zadané číslo. V C# je totiž vždy nutné uvádět rozměry polí předem a tyto rozměry poté již nejde měnit. Nelze proto nastavit výchozí velikost již při definici proměnných. Funkce bude typu void, což znamená, že nevrací žádnou hodnotu, neboť jejím jediným úkolem je definovat již existující struktury v rámci instance třídy Matice. Vstupem bude hodnota typu celé číslo, která funkci předává zvolenou velikost soustavy.

```
public void Vytvor(int n)
{
    nezname = new float[n, n];
    konstanty = new float[n];
}
```

Na řadě je již samotný výpočet determinantu podle výše popsaného Laplaceova rozvoje. Jak již bylo zmíněno, bude se jednat o rekurzivní funkci, což znamená, že bude v určitém momentě volat sama sebe. Funkce je typu float, což znamená, že vrací hodnotu typu desetinného čísla. Vstupem je dvourozměrné pole typu float, tedy matice soustavy. Jelikož rekurze volající funkci determinant pro matici o jednu menší se musí v určitém kroku zastavit a vrátit nějakou hodnotu, byla pro toto nastavena podmínka, že pokud se velikost matice rovná dvěma, funkce vypočítá determinant podle vzorce zmíněného v předešlé kapitole a tuto hodnotu vrátí.

Na začátku tedy funkce zjišťuje velikost jednoho z rozměrů pole. Jelikož je matice vždy typu nxn, není třeba zjišťovat oba rozměry, neboť jsou v každém případě stejné. Dále se ověří podmínka z předešlého odstavce, která v případě splnění provede výpočet a vrátí hodnotu. V opačném případě nastává využití Laplaceova rozvoje. Suma se zde implementuje ve formě cyklu, který pro každý prvek řádku nejprve zkontroluje, zda je nulový, a pokud ano, jeden krok cyklu přeskočí. Jinak přičte k proměnné determinant hodnotu podle vzorce. Namísto

umocňování -1 se zde použije podmínka, která pro sudý součet koeficientů hodnotu přičte, a pro lichý odečte. Tento postup je s umocňováním -1 ekvivalentní, neboť -1 umocněná sudou hodnotou je 1, zatímco při umocnění lichou hodnotou -1.

```
public float DET(float[,] a)
{
    int n = a.GetLength(0);
    if (n == 2)
        return a[0, 0] * a[1, 1] - a[0, 1] * a[1, 0];
    else
    {
        float det = 0;
        for (int i = 0; i < n; i++)
        {
            if (a[0, i] == 0)
                continue;
            if(i
                det += a[0, i] * DET(SubMatice(i, a));
            else
                det -= a[0, i] * DET(SubMatice(i, a));
        }
        return det;
    }
}
```

Předešlá funkce využívá strukturu submatice, kterou kód implicitně nemá k dispozici a je tudíž nutné vytvořit zvláštní funkci, která ji vytvoří. Jelikož vrací samotnou submatici, funkce vrací dvourozměrné pole typu float. Vstupem je původní matice, a hodnota typu celé číslo, představující index j. Index i jako vstup není zapotřebí, jelikož se v průběhu výpočtu nemění. Funkce vytvoří nové dvourozměrné pole, s velikostí o jednu menší než pole původní. Poté za pomoc dvou cyklů do pole postupně kopíruje hodnoty, z pole původního počínaje druhým řádkem, přičemž vždy ověřuje podmínku, že se index daného prvku nerovná j.

```
private float[,] SubMatice(int y, float[,] a)
{
    int n = a.GetLength(0);
    float[,] sub = new float[n - 1, n - 1];
    int indexX = 0;
    for (int i = 1; i < n; i++)
    {
        int indexY = 0;
        for (int j = 0; j < n; j++)
            if (j != y)
                sub[indexX, indexY++] = a[i, j];
        indexX++;
```

```
    }
    return sub;
}
```

V rámci aplikace je ověření hodnoty determinantu provedeno dříve než krok umožňující uživateli zvolit metodu výpočtu a případně požadovanou přesnost výsledku. Tudíž pokud je determinant nulový, k následujícímu kroku aplikace uživatele nepustí. Namísto toho zobrazí zprávu, jež nás informuje, že matice není regulární.

2.2 Eliminační metody obecně

Eliminační metody se řadí mezi tzv. přímé metody řešení soustav lineárních rovnic. Přímá metoda je definována jako metoda obsahující algoritmus přesného řešení, z čehož vyplývá, že v případě provedení výpočtů bez zaokrouhlování dojdeme k přesnému výsledku. Eliminačními metodami, které si zde ukážeme, jsou Gaussova eliminační metoda a Jordanova eliminační metoda. V rámci obou těchto metod je v první části využit totožný postup, a sice převod matice soustavy na trojúhelníkový tvar.

Než se však začneme zabývat samotným výpočtem, musíme soustavu rovnic pro začátek přepsat do tvaru matice, s koeficienty neznámých na levé straně a absolutními členy na pravé straně – tímto způsobem by měla být upravena již samotná soustava, kterou budeme na matici převádět. Sloupec s absolutními členy bývá zvykem oddělit za pomoci svislé čáry.

Získanou matici budeme poté za využití ekvivalentních neboli řádkových úprav převádět na horní trojúhelníkový tvar. Matice, již tímto postupem získáme, je poté ekvivalentní s maticí původní a v závislosti na užité metodě z ní můžeme snadno vypočítat hodnotu neznámých. Zaokrouhlovacím chybám, jimž se při manuálním výpočtu často nevyhneme, lze do jisté míry předejít použitím vhodného počítačového softwaru. Avšak i zde existují jisté meze a k určitým chybám zaokrouhlení může dojít.

Jelikož se pro obě metody první část postupu totožná, bude její algoritmizace a implementace popsána zde ve společné části, zatímco část postupu specifická pro danou metodu bude uvedena v kapitole jí určené. [21]

2.2.1 Algoritmizace převodu na horní trojúhelníkový tvar pro MS Excel

1. krok

   Když Determinant matice = 0, konec.

   Jinak pokračujeme 2. krokem.

2. krok

   Když $a_{11} \neq 0$

      Pokračujeme 3. krokem.

   Jinak

Najdeme první $i \in \{2,3,4\}$ takové, že $a_{i1} \neq 0$, prohodíme první řádek s i-tým a pokračujeme 3. krokem.

3. krok

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

Vynásobíme i-tý řádek koeficientem a_{11} a odečteme od něj a_{i1}-násobek prvního řádku. Pokračujeme 4. krokem.

4. krok

Když $a_{22} \neq 0$

Pokračujeme 5. krokem.

Jinak

Najdeme první $i \in \{3,4\}$ tak, že $a_{i2} \neq 0$, prohodíme druhý řádek s i-tým a pokračujeme 5. krokem.

5. krok

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

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

Pokračujeme 6. krokem.

6. krok

Když $a_{33} \neq 0$

Pokračujeme 7. krokem.

Jinak

Prohodíme třetí řádek se čtvrtým a pokračujeme 7. krokem.

7. krok

Vynásobíme čtvrtý řádek koeficientem a_{33} a odečteme od něj a_{43}-násobek třetího řádku.

2.2.2 Implementace v MS Excel

Při implementaci algoritmu pro výpočet Gaussovy eliminační metody v prostředí kancelářského softwaru MS Excel je nejprve nutné vzít v potaz omezení plynoucí ze statické povahy zmíněného prostředí. Zásadní rozdíl oproti programovacímu jazyku C# i matematickému výpočtu jsme limitováni přesně daným počtem a polohou jednotlivých

buněk, které nemůže program bez manuálního zásahu uživatele sám měnit, a tudíž je nutné jakoukoliv automatizaci výpočtu uvažovat v předem pevně zadané tabulce.

Z toho vyplývá nutnost zachování vždy téhož počtu kroků, bez ohledu na splnění či nesplnění vstupních podmínek. Zatímco MS Excel nedokáže v případě potřeby přidávat další kroky, vynechání některých předem nastavených kroků zde naopak nevadí. Přesněji řečeno kroky nelze vynechat úplně, nicméně je možné nastavit podmínku, která zajistí, že buňka za určitých okolností pouze přepošle výchozí hodnoty dále, místo aby s nimi prováděla jakékoliv výpočty. [2]

Abychom mohli sestavit kompletní schéma pro výpočet, je zapotřebí určit maximální potřebný počet maticových úprav. K tomu nám poslouží algoritmizace pro MS Excel z předešlé podkapitoly. Zde se v rámci kroků 1 až 6 využívaly střídavě podmínky a cykly. Stejný princip byl využit i v jazyce C# v kapitole následující, kde si můžeme všimnout vnějšího cyklu, v němž se vždy opakuje nejprve ověření podmínky, že člen hlavní diagonály na řádku, který budeme od ostatních odčítat, je nenulový.

Pokud ano, algoritmus může pokračovat dalším krokem, jímž je vnitřní cyklus provádějící samotný odečet řádků od všech řádků pod ním. Pokud však podmínka splněna není, program najde nejbližší řádek s nenulovým členem na klíčové pozici a s tím současným ho prohodí. Ačkoliv je princip algoritmu stejný, implementace se bude lišit, neboť má MS Excel poněkud omezené možnosti větvení.

Na rozdíl od flexibilního prostředí, které poskytují programovací jazyky, není v Excelu možné dynamicky měnit velikost soustavy, aniž by bylo nutné manuálně předělat celý výpočet. Jak je již naznačeno v algoritmizaci, budeme v tomto tabulkovém editoru pracovat se soustavou čtyř rovnic pro čtyři neznámé, výsledná matice bude tedy mít čtyři řádky a pět sloupců, počítáme-li pravý sloupec tvořený hodnotami absolutních členů. [2]

V tomto případě bude celkový počet tabulek nutných pro úpravy matice šest, k nimž však ještě přičteme úvodní matici s hodnotami zadanými uživatelem, z níž bude celý výpočet vycházet. Vytvoříme tedy celkem sedm matic umístěných pod sebou. Posloupnost algoritmu je taková, že každá matice se odkazuje pouze na tu předchozí, s jedinou výjimkou, kterou bude tvořit odkaz na matici výchozí. Později si vysvětlíme proč.

Jelikož tyto tabulky představují matici soustavy, bude nejlepší si jednotlivé sloupce nadepsat podle neznámých, v tomto případě w, x, y a z pro sloupce levé části tabulky, a b pro pravou část, neboli sloupec absolutních členů. V zájmu větší přehlednosti je taktéž na místě doporučení k využití vhodného ohraničení tabulek, případně i barevného značení odlišujícího dva typy operací, které budou jednotlivé tabulky zastávat. Nejlepší však bude, když ponecháte tyto úpravy na konec, jelikož při využití automatického vyplňování tabulky opakujícími se vzorci se krom obsahu buněk kopíruje i jejich formátování, které je posléze stejně nutné znovu předělat. Alternativně je možné problém vyřešit tak, že po každém vyplnění klikneme na okénko Možnosti automatického vyplnění, které se na konci výběru objeví, a zaškrtneme možnost Vyplnit bez formátování.

První tabulku tedy ponecháme prázdnou, jelikož do ní bude data zapisovat uživatel. Vstup od uživatele je jako vždy zapotřebí vhodně ošetřit, abychom se vyhnuli nechtěným výjimkám. V tomto případě je na prvním místě nutné zjistit, zda je matice regulární a má

Obrázek 1: Úvodní tabulka

w	x	y	z	b
0	0	0	4	5
1	1	1	1	5
0	3	5	0	3
1	2	5	5	4

tedy vůbec smysl s ní provádět jakékoliv operace. To můžeme ověřit výpočtem determinantu.

Pokud je determinant nulový, matice je singulární a její řádky jsou tedy lineárně závislé. Z toho vyplývá, že při úpravách na horní trojúhelníkový tvar by dříve či později došlo k situaci, kdy bychom získali nulový řádek. Pokud by šlo o celý řádek včetně pravé strany, značilo by to, že má soustava nekonečně mnoho řešení. Nulová levá strana a nenulová pravá strana naopak značí, že soustava nemá žádné řešení, neboť dělící čára mezi levou a pravou stranou představuje rovnítko a nula se nemůže rovnat jinému číslu, tak jako se nic nemůže rovnat něčemu.

My se však spokojíme se zjištěním, že matice je singulární a nemá smysl s ní provádět početní operace. Zde narážíme na dříve zmíněnou výjimku. Potřebujeme zařídit, aby v takovém případě žádná z buněk neprováděla jinou operaci než zobrazení čísla ze stejné pozice v předešlé matici. Pro singulární matici nejsou další výpočty ošetřeny proti dalším výjimkám a kdyby výpočet normálně probíhal, mohlo by docházet k situacím jako je dělení nulou. Excel je na rozdíl od programovacího jazyka schopen dál fungovat, i když v některých jeho buňkách dojde ke stavu vyvolávajícímu chybové hlášení, takže k ničemu natolik závažnému jako je pád programu by nedošlo. Je však lepší mít to ošetřené, aby chybové výstupy uživatele zbytečně nemátly.

Pro regulární matici už algoritmus pokračuje normálně, neboť podmínka nenulového determinantu je splněna. Jako první je zapotřebí se přesvědčit, zda není na místě koeficientu prvku a_{11} zadaná nula a v případě, že ano, prohodit první řádek s jiným řádkem tak, aby byl zmíněný koeficient nenulový. Toho docílíme v prvním kroku, který ještě nebude provádět samotnou eliminaci. Namísto toho pomocí vhodně zvolených podmínek pouze změní řazení.

Prakticky bude řešení, implementované ve druhé tabulce, vypadat následovně. Využity budou postupně vnořované podmínky, tedy funkce KDYŽ. V každé buňce tabulky bude nejprve podmínka, která ověří, jestli je koeficient a11 nenulový. Pokud ano, pouze překopíruje prvek ze stejné pozice v tabulce. V opačném případě už se postup pro jednotlivé řádky liší. V prvním řádku bude další podmínka ověřovat, jestli je koeficient o řádek nižší nenulový, a pokud ano, překopíruje na místo prvního řádku řádek druhý. Pokud ne, pokračuje další podmínka, řešící totéž pro třetí řádek, který pokud má také na požadovaném místě nulový koeficient, překopíruje sem funkce řádek čtvrtý.

K situaci, že by se v této fázi nula nacházela v každém sloupci, dojít nemůže, neboť by to znamenalo, že je matice singulární, a to je podmínka, která byla již na začátku prověřena. Pro ostatní řádky se pak ověřuje stejná posloupnost podmínek, rozdíl však tkví v tom, že

Obrázek 2: Řadící tabulka

w	x	y	z	b
1	1	1	1	5
0	0	0	4	5
0	3	5	0	3
1	2	5	5	4

pouze pokud je tento řádek první s nenulovým koeficientem, kopíruje se na jeho místo řádek první, jinak zůstává daný řádek na svém místě.

Matice je seřazena a algoritmus pokračuje dalším krokem, implementovaným ve třetí tabulce. První řádek se pouze zkopíruje, neboť ten už se po zbytek průběhu metody nezmění. Každý další řádek i je pak vypočítán jako a_{11} násobek řádku i minus a_{ij} násobek řádku 1, tento výpočet probíhá zvlášť pro každou buňku, nicméně vzorec zůstává stejný, tudíž je možné využít automatického vyplňování. Jen je třeba zafixovat řádek i sloupec koeficientu a_{11}, sloupec koeficientu a_{ij} a řádek prvku z prvního řádku – odečítáme ho totiž od všech ostatních řádků. Při správné implementaci by první sloupec třetí tabulky měl krom prvního řádku obsahovat samé nuly.

Obrázek 3: První krok eliminace

w	x	y	z	b
1	1	1	1	5
0	0	0	4	5
0	3	5	0	3
0	1	4	4	-1

Tyto dva kroky nyní analogicky zopakujeme pro další dvě úrovně eliminace. Nejprve seřadíme matici tak, abychom získali nenulový koeficient a_{22}, resp. a_{33}, a v dalším kroku provedeme eliminaci samotnou. Poslední krok je implementován v sedmé tabulce, která by v případě regulární výchozí matice měla zobrazovat matici převedenou na horní trojúhelníkový tvar, který bude v následujících metodách využit k výpočtu samotného výsledku.

Obrázek 4: Horní trojúhelníkový tvar

w	x	y	z	b
1	1	1	1	5
0	3	5	0	3
0	0	7	12	-6
0	0	0	4	5

2.2.3 Implementace v C#

Ve srovnání s kancelářským softwarem MS Excel, kde je počet a umístění buněk a v návaznosti na to i struktura algoritmu pevně dána, umožňuje programovací jazyk C# vývoj algoritmů pružnějšího rázu. Toho lze docílit především díky cyklům, které zde využijeme k provedení opakujících se kroků, a také podmínek, umožňujících kód větvit a provést vždy jen tu část, kterou v daném případě potřebujeme. Mimo úspory času tím docílíme i lepší přehlednosti kódu.

Matici představující koeficienty neznámých a sloupec absolutních členů zde uchováváme ve dvou různých strukturách, což nám sice usnadní práci s determinantem, kde se pracuje pouze se základní maticí soustavy. Nicméně pro samotnou eliminaci, která se k rozšířené matici chová jako k jedné souvislé struktuře bude lepší je spojit. Jelikož spojování a rozdělování zde neprobíhá tak jednoduše jako v prostředí MS Excel, musíme vytvořit nové pole se stejným počtem řádků, ale o jednu vyšším počtem sloupců, do níž pak pomocí cyklu proměnné z původních polí nakopírujeme.

Další výpočty budou probíhat s touto rozšířenou maticí a stojí za to si povšimnout, že právě ji metoda vrací. Vstupním parametrem je v tomto případě jen celé číslo značící velikost matice. Samotnou matici není nutné předávat jako parametr, neboť je již součástí objektu, na němž metoda pracuje, a můžeme na ně tedy odkazovat přímo.

```
private float[,] Eliminace(int n)
{
    float[,] a = new float[n, n + 1];
    for(int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            a[i, j] = nezname[i, j];
        a[i, n] = konstanty[i];
    }
```

Eliminace samotná je opět uzavřena do cyklu, neboť jednotlivé kroky probíhají podle stejného schématu, které je možné pozorovat již v algoritmizaci pro MS Excel. Tento cyklus se pak skládá ze dvou částí. První část nám definuje podmínka, která má za úkol ověřit, jestli se v řádku, jehož koeficient je roven pořadí prováděné iterace, člen s totožným řádkovým i sloupcovým indexem nerovná nule. Pokud ano, kód odkáže na provedení pivotace, neboli změna řazení řádků. V opačném případě se pokračuje dál.

V druhé části se nachází další cyklus, vnořený do cyklu původního, který provede samotnou eliminaci pomocí řádku, s nímž daný krok pracuje. Jelikož bude docházet ke změnám hodnoty některých prvků matice, načte si cyklus v každém kroku koeficienty, jimiž bude dané řádky násobit. Až poté proběhne další, již třetí cyklus, jenž se postará o postupné odečtení příslušných hodnot v každém sloupci zvlášť. Jelikož v každém kroku máme k dispozici řádek sloužící k odečtu, jehož prvky před hlavní diagonálou jsou již nulové, má smysl poslední cyklus provádět až od pořadí prvního nenulového členu – tedy toho, jež se nachází na hlavní diagonále.

```
    for (int i = 0; i < n - 1; i++)
    {
        if (a[i, i] == 0)
            a = Pivotace(a, i, n);
        for (int j = i + 1; j < n; j++)
        {
            float koefX = a[i, i];
            float koefY = a[j, i];
            for (int k = i; k <= n; k++)
                a[j, k] = a[j, k] * koefX - a[i, k] *
                koefY;
        }
    }
    return a;
}
```

Pro přehlednost je pivotace též umístěna v samostatné metodě. I tato metoda vrací matici, neboť její jedinou funkcí je změna pořadí řádků. Vstupní parametry jsou pak původní matice, celkový počet řádků, a index řádku, s nímž se aktuálně pracuje. Kód samotný je pak velmi jednoduchý. Cyklus postupně testuje řádky s vyšším indexem než je ten vstupní, jestli na potřebném místě obsahují nenulový koeficient. Pokud testovaný řádek nenulový koeficient obsahuje, následuje cyklus, který jej po jednotlivých prvcích postupně prohodí s řádkem aktuálním, a výslednou matici vrátí. Jinak je provedena další iterace, pro řádek s indexem o číslo vyšším. Že kód uspěje předem dáno tím, že bylo předem zajištěno, aby byla matice k samotnému výpočtu odeslána pouze pokud test determinantu potvrdí, že je regulární. Nemůže se tedy stát, že by ani jeden řádek odpovídající požadavku na nenulový člen nalezen nebyl.

```
private float[,] Pivotace(float[,] matice, int i, int n)
{
    for (int j = i + 1; j < n; j++)
        if (matice[j, i] != 0)
        {
            for (int k = 0; k <= n; k++)
            {
                float zastupce = matice[i, k];
                matice[i, k] = matice[j, k];
                matice[j, k] = zastupce;
            }
            return matice;
        }
    return null;
}
```

2.3 Gaussova eliminační metoda

Jak již bylo nastíněno výše, jeden ze základních principů má Gaussova eliminace společný s ostatními eliminačními metodami. Tímto principem je převod matice soustavy na horní trojúhelníkový tvar, jehož poměrně snadno dosáhneme za pomoci algoritmu uvedeného v předešlé kapitole. Právě tato část postupu se nazývá eliminace, proto je společná všem metodám, které nazýváme eliminační. Specifikum Gaussovy eliminace spočívá v postupu, jenž následuje. Byla-li matice regulární, eliminací získáme matici, na jejíž hlavní diagonále nejsou žádné nuly.

Odsud už je velice snadné vypočítat hodnoty jednotlivých neznámých. Jelikož poslední řádek matice nyní představuje lineární rovnici o jedné neznámé, její výpočet nepředstavuje žádný problém. Nalezená hodnota se poté vezme a dosadí za tuto neznámou do předposledního řádku, čímž díky eliminaci neznámé tím, že ji nahradíme konkrétním číslem, získáme opět lineární rovnici o jedné neznámé. Tímto způsobem se pokračuje dále vzhůru, dokud se postupně nenaleznou všechny neznámé. Zatímco v excelu je zapotřebí každý vzorec sestavit zvlášť, v prostředí programovacího jazyka si ukážeme způsob, jakým se dá výpočet zobecnit. V rámci algoritmizace pro MS Excel již metoda obsahuje jen jediný, poslední krok. [21]

2.3.1 Algoritmizace pro MS Excel

8. krok

Neznámou z vyjádříme z posledního řádku matice tímto způsobem: $z = b_4/a_{44}$

Neznámou y vyjádříme ze třetího řádku matice a proměnné z získané dříve tímto způsobem: $y = (b_3 – a_{34} * z)/a_{33}$

Neznámou x vyjádříme ze druhého řádku matice a proměnných z a y získaných dříve tímto způsobem: $x = (b_2 – a_{24} * z – a_{23} * y)/a_{22}$

Neznámou w vyjádříme z prvního řádku matice a proměnných z, y a x získaných dříve tímto způsobem: $w = (b_1 – a_{14} * z – a_{13} * y – a_{12} * x)/a_{11}$

2.3.2 Implementace v MS Excel

Větší část implementace této metody byla již vysvětlena v předchozí části pro obě eliminační metody společně. Nyní zbývá už jen samotný výpočet výsledků z horního trojúhelníkového tvaru, do něhož jsme matici převedli. Tato část již nebude graficky nijak náročná, jelikož nám bude stačit jednoduchá tabulka o dvou sloupcích a čtyřech řádcích. V prvním sloupci každého řádku bude uvedena neznámá, již odsud dopočítáváme. Ve druhém sloupci pak již samotný výsledek, nebo případně oznámení, že matice není regulární.

Každá z těchto buněk bude začínat podmínkou znovu a naposledy ověřující hodnotu determinantu výchozí matice. V případě nulového determinantu vypíše hlášku zmíněnou výše. V opačném případě začne samotný výpočet. Ten bude, podle samotného trojúhelníkového tvaru, začínat odzdola, tedy od proměnné z. Tu vypočítáme velmi snadno, vydělíme hodnotu v pravé části koeficientem a_{44}. Další proměnné už budou o něco složitější. Vzorec počítající hodnotu proměnné y vezme hodnotu pravé strany předposledního řádku a odečte od ní koeficient proměnné z vynásobený její hodnotou. Výsledek poté vydělí koeficientem a_{33}. Zbylé dvě proměnné získáme postupem analogickým s tímto, vždy za využití příslušného řádku a hodnot již známých proměnných. [2]

Obrázek 5: Tabulka s výsledky

w	0,75
x	6
y	-3
z	1,25

2.3.3 Implementace v C#

Kód implementovaný v předešlé kapitole upravil rozšířenou matici soustavy na trojúhelníkový tvar. S tímto tvarem budeme nadále pracovat, abychom z něj získali konkrétní řešení. Metoda implementovaná v této části již nese název Gaussovy eliminace jelikož jsou v kódu skutečně uvedeny všechny potřebné kroky. Metoda vrací jednorozměrné pole typu float, které již představuje samotné hodnoty neznámých. Vstupní parametry nemá žádné, neboť využívá pouze struktury existující v daném objektu.

Nejprve zjistí délku pole s absolutními členy, jenž pak předává jako vstupní parametr metodě Eliminace, jež byla vysvětlena v předešlé kapitole, a současná metoda Gauss ji volá, čímž získává soustavu s horním trojúhelníkovém tvaru. Dále je třeba vytvořit pole, do nějž bude kód zapisovat výsledky a následně s nimi počítat. Toto pole bude mít stejnou délku jako pole s absolutními členy.

Zatímco v prostředí MS Excel jsme mohli jednotlivé vzorce vypsat postupně, jelikož jsme pracovali výhradně se soustavami velikosti 4, nyní je zapotřebí vymyslet způsob, jak daný postup zobecnit, aby byl algoritmus proveditelný na matici jakékoliv velikosti. Abychom vyhověli požadavku, že v době výpočtu každé neznámé musí již být k dispozici hodnoty těch předchozích, využijeme cyklu, jenž bude začínat výpočtem proměnné na posledním řádku, a postupně půjde nahoru. Hodnota iterace bude poněkud netypicky začínat na nejvyšším čísle, a postupně se snižovat.

V každém kroku se do pomocné proměnné nejprve přiřadí hodnota konstanty na řádku, s nímž se pracuje, a jenž bude také využit ve zbylých výpočtech tohoto kroku. Následuje cyklus začínající hodnotou o jednu vyšší než je hodnota současné iterace. Tento cyklus má zároveň podmínku, že tato hodnota nesmí být větší nebo rovna počtu řádků. Z toho vyplývá, že pokud právě pracujeme s posledním řádkem, tento vnitřní cyklus se vůbec neprovede.

Pokud ale ano, v každém jeho kroku se od pomocné proměnné odečte součin koeficientu prvku, jehož pořadí v daném řádku odpovídá hodnotě iterace vnořeného cyklu, a odpovídající již vypočítané proměnné. Po skončení tohoto cyklu se výsledek vydělí hodnotou prvku, jehož pořadí odpovídá i pořadí řádku, tedy koeficientu u proměnné, kterou zrovna počítáme. Tento výsledek se přiřadí na odpovídající místo v poli s výsledky.

```
public float[] Gauss()
{
    int n = konstanty.Length;
    float[,] a = Eliminace(n);
    float[] vysledky = new float[n];
    for (int i = n - 1; i >= 0; i--)
    {
        float konst = a[i, n];
        for (int j = i + 1; j < n; j++)
            konst -= a[i, j] * vysledky[j];
        vysledky[i] = konst / a[i, i];
    }
    return vysledky;
}
```

2.4 Jordanova eliminační metoda

Spolu s Gaussovou eliminací spadá i Jordanova metoda mezi eliminační metody, což znamená, že počáteční postup, a sice převod matice soustavy do horního trojúhelníkového tvaru, je pro obě metody totožný. Specifická je až další část, v níž se z trojúhelníkového tvaru získávají výsledky samotné. Na rozdíl od Gaussovy metody, který neznámé vyjadřuje přímo z upravené matice soustavy za pomoci vzorců, dopracovává se Jordanova metoda k výsledkům i nadále pomocí ekvivalentních úprav.

Cílem je tentokrát převést trojúhelníkový tvar až na redukovaný trojúhelníkový tvar. V případě Jordanovy eliminační metody je výsledným tvarem jednotková matice na levé straně a sloupec s čísly, jež jsou výslednými hodnotami jednotlivých neznámých. Pro snazší vizualizaci odůvodnění, proč je tomu právě tak, si jednotlivé řádky matice představme opět jako rovnice. Zjistíme, že každý z řádků je lineární rovnice, jejíž levá strana představuje neznámou s koeficientem jedna, a pravá strana absolutní člen. Tedy nám rovnice rovnou říká, jakou hodnotu daná neznámá zastává.

Ve srovnání s Gaussovou metodou má Jordanova metoda řešení, jež je na první pohled velmi dobře čitelné. Za nevýhodu se poté dají považovat mnohdy pracnější a složitější výpočty, při nichž se většinou nevyhneme zlomkům či desetinným číslům. To se však dá za problém považovat pouze v případě manuálního výpočtu na papír. [21]

2.4.1 Algoritmizace pro MS Excel

8. krok

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

Vydělíme i-tý řádek koeficientem a_{ii}

Pokračujeme 9. krokem.

9. krok

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

Odečteme od i-tého řádku a_{i4} násobek čtvrtého řádku.

Pokračujeme 10. krokem.

10. krok

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

Odečteme od i-tého řádku a_{i3} násobek třetího řádku.

Pokračujeme 11. krokem.

11. krok

Odečteme od prvního řádku a_{12} násobek druhého řádku.

Pokračujeme 12. krokem.

12. krok

Ze získané matice určíme hodnotu neznámých tímto způsobem: $w = b_1, x = b_2, y = b_3, z = b_4$

2.4.2 Implementace v MS Excel

Postup využitý v Jordanově eliminační metodě je velmi podobný tomu, který jsme využívali na začátku, abychom matici převedli do horního trojúhelníkového tvaru. Nyní bude naším cílem tzv. redukovaný trojúhelníkový tvar. V praxi se v našem případě bude jednat o jednotkovou matici velikosti čtyři na levé straně a sloupci s hodnota mi na straně pravé. Stejně jako v předešlém případě, i nyní si musíme předem určit počet kroků, neboť na tom závisí počet tabulek, které si pro tuto část výpočtu vytvoříme.

I k tomuto účelu nám poslouží algoritmizace pro MS Excel popsaná výše. Vidíme zde, že úprava na redukovaný trojúhelníkový tvar probíhá v krocích 8 až 11, z čehož není těžké odvodit, že se bude jednat celkem o čtyři kroky. Tabulky budou rovněž čtyři, neboť na rozdíl od samotné eliminace, zde nebudeme jako výchozí matici používat vstup od uživatele, ale poslední, tedy sedmou matici předcházejícího postupu. Odkazovat tedy budeme přímo na ni.

Opět je zapotřebí pro všechny buňky provádějící jiné operace než prosté kopírování implementovat podmínku, že Determinant matice zadané uživatelem nesmí být nulový. Pokud je, buňka pouze zkopíruje příslušnou hodnotu. Stejně jako při předchozím postupu se vyplatí nadepsat si buňky označením neznámých a sloupec absolutních členů písmenem b, značícím pravou stranu matice. I zde doporučuji po zadání všech výpočetních vzorců tabulku naformátovat tak, aby se zlepšila přehlednost a vizuální dojem.

Na rozdíl od Eliminace, nyní již při úpravách nehrozí, že bychom na hlavní diagonále dostali nulu, tudíž odpadá nutnost tabulku nejprve řadit. Můžeme tedy začít rovnou s výpočty. Vzhledem k povaze samotného postupu bude nejlepší začít tím, že každý řádek vydělíme hodnotou jeho řídícího prvku, což je v tomto případě vždy prvek na hlavní diagonále, a zároveň první nenulový prvek daného řádku. Získáme tak na hlavní diagonále samé jedničky, což nám již může napovídat, že jsme učinili první krok k získání jednotkové matice na levé straně. Zároveň jsme si tímto krokem zjednodušili výpočty, jež budou následovat. [2]

Obrázek 6: Tabulka s upravenou diagonálou

w	x	y	z	b
1	1	1	1	5
0	1	1,666667	0	1
0	0	1	1,714286	-0,85714
0	0	0	1	1,25

Nyní již máme na hlavní diagonále samé jedničky a pod ní samé nuly zbývá nám tedy už jen upravit prostor nad hlavní diagonálou, aby se i tam nacházely samé nuly. Toho docílíme v následujících třech krocích. Výpočty v druhé tabulce budou spočívat v tom, že postupně odečteme ai4-násobek, kde $i \in \{1, 2, 3\}$, čtvrtého řádku od každého z řádků nad ním. Jelikož hodnota čtvrtého koeficientu ve čtvrtém řádku je jedna, není potřeba zbylé řádky ničím násobit. Hodnotu řádku, který budeme odečítat, však musíme vynásobit hodnotou čtvrtého koeficientu řádku, od kterého budeme odčítat. Důvod je stejný jako v případě eliminace – abychom na tomto místě měli nulu.

Ve třetí matici můžeme tento postup zopakovat tak, že odečteme a_{i3}-násobek, kde $i \in \{1, 2\}$, třetího řádku od řádků nad tím. Princip je úplně stejný, a jelikož je ve čtvrté buňce třetího řádku nula, nemusíme se obávat, že by tento krok jakkoliv narušil výsledek předchozího. V poslední tabulce poté odečteme a_{12}-násobek druhého řádku od řádku prvního.

Vidíme, že výsledná matice má skutečně redukovaný trojúhelníkový tvar, a pokud jsme již předtím dělali Gaussovu metodu pro stejnou soustavu rovnic, můžeme si všimnout, že ve sloupci absolutních členů vidíme řešení soustavy. Stejně jako u předešlé metody nám zbývá vytvořit tabulku výsledků, do níž tentokrát pouze vložíme vzorce s odkazy na příslušné buňky ve sloupci absolutních členů poslední tabulky.

Obrázek 7: Redukovaný trojúhelníkový tvar

w	x	y	z	b
1	0	0	0	0,75
0	1	0	0	6
0	0	1	0	-3
0	0	0	1	1,25

2.4.3 Implementace v C#

Metoda implementující průběh Jordanovy eliminace se na první pohled podobá metodě pro Gaussovu eliminace, neboť stejně jako ona vrací jednorozměrné pole typu float a její deklarace neobsahuje žádný vstupní parametr. I první tři řádky kódu v těle metody jsou totožné, neboť i zde je třeba provést eliminaci odkazem na příslušnou metodu a vytvořit pro výsledky pole o patřičné délce.

```
public float[] Jordan()
{
    int n = konstanty.Length;
    float[,] a = Eliminace(n);
    float[] vysledky = new float[n];
```

Stejně jako v případě algoritmu pro MS Excel, i zde se následující část kódu věnuje převodu na redukovaný trojúhelníkový tvar, počínaje konkrétně úpravami, které mají za cíl zajistit, aby se na hlavní diagonále matice nacházely samé jedničky. I zde je tento úkon velice jednoduchý, v tomto případě realizovaný za pomoci dvou cyklů a jedné podmínky. První cyklus se jednoduše stará o to, aby algoritmus prošel všechny řádky.

Pro každý z řádků se poté do pomocné proměnné uloží koeficient prvku ležící na hlavní diagonále, načež podmínka ověří, zda se náhodou nerovná jedné. V případě, že ne, následuje cyklus, který postupně všechny prvky daného řádku tímto koeficientem vydělí. Aby se zamezilo provádění zbytečných operací, kdy by se dělily nulové členy před hlavní diagonálou, má cyklus počátek až u prvku na stejném indexu, jako je pořadí řádku. Tedy jinými slovy, na prvku hlavní diagonály.

```
    for (int i = 0; i < n; i++)
    {
        float koef = a[i, i];
        if (koef != 1)
            for (int j = i; j <= n; j++)
                a[i, j] = a[i, j] / koef;
    }
```

Následující krok se velmi podobá metodě obsluhující eliminaci na horní trojúhelníkový tvar z předešlých kapitol, ovšem s jedním rozdílem. Zde se již není třeba starat o možnost, že by se na hlavní diagonále nacházel nulový prvek, neboť všechny takové eventuality již

ošetřila eliminace samotná. Navíc při výpočtech zde ani nehrozí, že by nula na hlavní diagonále vznikla v průběhu algoritmu.

I v tomto případě bude kód sestávat ze dvou do sebe vnořených cyklů. První zajišťuje, aby byl daný úkon proveden pro každý řádek, avšak tentokrát začíná odspoda, jelikož se nejprve eliminuje poslední sloupec za pomoci posledního řádku, načež předposlední sloupec za pomoci řádku předposledního, atd. Toto pořadí zajišťuje, že řádek, s nímž pracujeme, bude ovlivňovat pouze příslušný sloupec a sloupec absolutních členů, jelikož bude mít na všech ostatních místech samé nuly.

Vnitřní cyklus poté provede eliminaci daným řádkem. Jelikož násobek tohoto řádku odečítáme vždy jen od řádků nad ním, začíná i tento cyklus dole, konkrétně na řádku přímo nad tím, se kterým vnější cyklus pracuje. Zde se od absolutního členu každého řádku odečte součin koeficientu členu na stejném indexu, jako řádek, s nímž právě pracujeme, a absolutního členu tohoto řádku. Člen, jímž jsme absolutní člen našeho řádku násobili, nastavíme na nulu. Korektnější by bylo odečíst od něj násobek jeho samotného a řídícího prvku řádku, s nímž pracujeme, nicméně výsledek je tak či tak v každém případě nulový.

Po ukončení vnitřního cyklu se v rámci cyklu vnějšího uloží do výsledků absolutní člen řádku, s nímž jsme pracovali. Jelikož s prvním řádkem už žádné výpočty neprovádíme, uloží se jeho absolutní člen do výsledků zvlášť po ukončení obou cyklů.

```
    for (int j = n - 1; j > 0; j--)
    {
        for (int i = j - 1; i >= 0; i--)
        {
            a[i, n] -= a[j, n] * a[i, j];
            a[i, j] = 0;
        }
        vysledky[j] = a[j, n];
    }
    vysledky[0] = a[0, n];
    return vysledky;
}
```

2.5 Iterační metody

Na rozdíl od eliminačních metod, dosahujících při výpočtech bez zaokrouhlování přesných výsledků, jsou iterační metody založeny na postupné aproximaci výsledků, což znamená, že dosahují pouze přibližných hodnot, vždy s určitou předem stanovenou mírou přesnosti. Jeden krok dané metody označujeme jako jednu její iteraci, odtud také pochází název iterační metody. V rámci každé iterace vzniká nová aproximace. Hodnoty těchto aproximací by za ideálních podmínek měly konvergovat směrem k řešení dané soustavy rovnic. V rámci této práce se seznámíme s Jacobiho metodou a Gauss-Siedelovou metodou, které jsou, jak si později ukážeme, založeny na velice podobném principu.

Pro výpočet každé aproximace se využívá stejná série vzorců, jež byly na začátku odvozeny ze samotných rovnic. Pro každou neznámou se zvolí jiná rovnice, z níž se hodnota neznámé následně vyjádří s pomocí vzorce. Takový vzorec sestává z absolutního členu, od něhož jsou odečteny všechny ostatní neznámé, a toto celé je vyděleno koeficientem neznámé, kterou vyjadřujeme. U Jacobiho metody se za neznámé vždy dosazují aproximované hodnoty z předešlého kroku, v případě, že se jedná o první krok, využijeme počáteční aproximaci, kterou si sami zvolíme. V tomto jediném se liší od Gauss-Siedelovy metody, u níž se k výpočtu využívá vždy nejnovější aproximace, která je k dispozici. V praxi to znamená, že pokud je v současné aproximaci již vypočítána neznámá x, použije se při výpočtu neznámé y již tato nová hodnota, namísto té z předešlé aproximace. [21]

Ani jedna z těchto metod však nemusí konvergovat. Pro obě metody existuje tzv. nutná a postačující podmínka konvergence. Zatímco splnění postačující podmínky nám zajistí, že soustava konvergovat bude, opačným směrem tato implikace neplatí, což znamená, že soustava může konvergovat, i když tuto podmínku nesplní. Naproti tomu nutná podmínka je splněna právě tehdy, když soustava konverguje. Jedná se tudíž o oboustrannou implikaci, neboli ekvivalenci. Postačující podmínkou je pro obě metody, aby byla matice diagonálně dominantní, tedy aby byl každý člen na hlavní diagonále větší než součet ostatních prvků v daném řádku. Nutnou podmínkou je, aby byl spektrální poloměr matice menší než 1. Vzhledem k náročnosti výpočtu spektrálního poloměru zde byl zvolen numerický způsob ověření konvergence, který bude v každé z následujících podkapitol uveden.

Pokud metoda konverguje, můžeme řešení dané soustavy rovnic nahradit aproximací splňující úvodní podmínku přesnosti, v případě MS Excel s pevným počtem kroků nahrazujeme řešení aproximací v posledním z nich, neboť ta dosahuje nejvyšší přesnosti. V obou zmíněných případech se jinak nekonečný proces konvergence nahrazuje konečným s tím, že se smíříme s určitou nepřesností. Chceme-li požadované přesnosti dosáhnout, potřebujeme v některých případech větší objem výpočtů než tomu je u přímých metod. Využitý algoritmus je na druhou stranu jednodušší, díky čemuž je implementace v prostředí MS Excel i jazyka C# méně náročná. Při automatizaci jsou tyto metody často vhodnější. Při manuálních výpočtech jsou naopak jednoznačně lepší přímé metody.

Zatímco u eliminačních metod se podmínky zajišťující funkčnost metody řeší postupně v průběhu, vzhledem k povaze statických iteračních metod, kdy se celou dobu počítá s tou samou maticí bez možnosti ji upravovat, je nutné toto vyřešit hned na začátku. Zde se omezení týká vzorce jednotlivých iterací, v němž je v rámci každého výpočtu zapotřebí dělit jedním z prvků hlavní diagonály matice. Aby takový výpočet dával z matematického hlediska smysl, nesmí se tento prvek rovnat nule. Zatímco ověřit tuto podmínku nepředstavuje problém, nalézt vhodné pořadí v případě, že původní matice nulu na hlavní diagonále má, je poněkud složitější. Jako první je ale stejně jako u eliminačních metod nutné ověřit, jestli je daná matice regulární, což se dá ověřit skrze výpočet determinantu, který pokud je nenulový, matice regulární je a soustava rovnic má jediné řešení.

2.5.1 Algoritmizace pro MS Excel

1. krok

   Když Determinant matice = 0

      Konec.

   Jinak

      Pokračujeme 2. krokem.

2. krok

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

      Pokračujeme 4. krokem.

   Jinak

      Pokračujeme 3. krokem.

3. krok

   Získáme všechny existující permutace pořadí řádků {1, 2, 3, 4} a pro každou z jejích buněk určíme, jestli by matice v případě tohoto pořadí obsahovala na hlavní diagonále člen s nulovým koeficientem.

   Určíme první permutaci, v jejímž případě matice nulu na hlavní diagonále neobsahuje a seřadíme podle ní původní matici.

   Pokračujeme 4. krokem.

4. krok

   Získáme uživatelem zadanou požadovanou přesnost p 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_1, x_1, y_1 a z_1.

   Pokračujeme 5. krokem.

5. krok

   Dokud pro všechna $e \in \{w,x,y,z\}$ a číslo i značící pořadí současné iterace platí $|e_i - e_{i-1}| > p$ a zároveň $i \leq 100$:

      Když používáme Jacobiho metodu:

      $$w_{i+1} = (b_1 - x_i * a_{12} - y_i * a_{13} - z_i * a_{14})/a_{11}$$
      $$x_{i+1} = (b_2 - w_i * a_{21} - y_i * a_{23} - z_i * a_{24})/a_{22}$$

$$y_{i+1} = (b_3 - w_i * a_{31} - x_i * a_{32} - z_i * a_{34})/a_{33}$$
$$z_{i+1} = (b_4 - w_i * a_{41} - x_i * a_{42} - y_i * a_{43})/a_{44}$$

Když používáme Gauss-Siedelovu metodu:

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

Pokračujeme 6. krokem

6. krok

Když pro všechna $e \in \{w, x, y, z\}$ platí $|e_{100} - e_{99}| > p$

Když pro všechna $e \in \{w, x, y, z\}$ platí $|e_{100} - e_{99}| \leq p$

Algoritmus našel řešení o požadované přesnosti.

Jinak

Počet kroků nestačil pro dosažení požadované přesnosti.

Jinak

Metoda diverguje.

2.5.2 Implementace v MS Excel

Zatímco než se v C# program pustí do výpočtu iterací metody samotné, provede několik testů, zda soustava rovnic pro danou metodu splňuje všechny náležité podmínky. Jelikož MS Excel funguje na jiném principu, neumožňuje nám selektivní provedení kroků v závislosti na splnění podmínek do takové míry jako programovací jazyk. Stejně jako u předchozích metod můžeme některé kroky ponechat prázdné, provést se však musí tak jako tak, bez ohledu na to, zda bylo podle předešlých podmínek vyhodnoceno, že to má smysl nebo ne. Jelikož zde nepředstavuje problém, pokud některý z výpočtů skončí chybou, bude nejjednodušší nejprve implementovat metodu, a až posléze ošetřit všechny potřebné podmínky, které pouze ovlivní výpis výsledku.

Pro začátek je i v tomto případě zapotřebí si nejprve vytvořit tabulku pro matici soustavy, nadepsanou názvy proměnných na levé straně a písmenem b na straně pravé, stejně jako u předchozích metod. I tentokrát tabulku doporučuji vhodně naformátovat. Do této tabulky zadá uživatel hodnoty jednotlivých koeficientů. I když se jedná o iterační metodu, a řešení tedy nebude probíhat pomocí maticových úprav, úplně se jim přesto nevyhneme, a tedy potřebujeme matici ještě jednu. Jediná úprava samotné matice bude spočívat v případné změně pořadí řádků. [2]

Jak již bylo zmíněno, iterační metody využívají vzorce, v nichž vždy probíhá dělením koeficientem vyjádřené neznámé. Je proto zapotřebí se ujistit, že tento koeficient nebude v žádném případě nulový. Pokud vyjadřujeme proměnné z rovnic v pořadí, v jakém jsou

zadány nahoře, dojdeme k zjištění, že nuly nesmí být na hlavní diagonále matice. Tato podmínka se ověří velmi jednoduše, ovšem byli bychom rádi, kdybychom krom identifikace nepoužitelného rozložení matice dokázali přijít i s řešením tohoto problému. V případě obecné matice by to bylo poněkud složitější, nicméně my pracujeme s maticí velikosti čtyři, pro niž nebude následující algoritmus představovat problém.

Pro každou matici velikosti n existuje n! různých permutací jejích řádků. Jak již nejspíš tušíme, prověřování každé z nich je algoritmus složitosti n!, což je složitost, jíž se při programování snažíme vyhnout. Pro n velikosti čtyři však faktoriál představuje pouhých 24 kombinací. O několik sloupců stranou si vytvoříme tabulku o šesti sloupcích a čtyřech řádcích, v níž postupně vytvoříme podmínky pro ověřování, jestli by při seřazení řádků podle každé z permutací byla na některém poli hlavní diagonály nula. V případě, že ne, funkce vypíše tuto permutaci jako číslo. Pokud ano, funkce může vypsat libovolný nečíselný znak, například pomlčku.

Z hlediska přehlednosti doporučuji zvolit si pro umisťování permutací nějaký systém, který nám pomůže udržet povědomí o tom, která permutace má kde být, a především pak jestli už jsme ji zapisovali, či nikoliv. Zobrazovanou permutaci bychom ideálně vraceli ve formě pole, nicméně MS Excel tuto datovou strukturu v jediné buňce nepodporuje, tudíž si budeme muset vystačit s číslem. V případě, že je matice regulární, vždy bude existovat alespoň jedna permutace, pro niž námi stanovená podmínka platí. Tuto permutaci musíme z tabulky vybrat, a rozdělit její jednotlivé cifry ve správném pořadí.

Vedle tabulky si vytvoříme sloupec o velikosti čtyři. První buňka představuje řádek, který se v naší permutaci vyskytuje na prvním místě v pořadí, pro ostatní řádky analogicky. V každém tomto políčku s pomocí funkce MIN vybereme minimální číslo zobrazené v tabulce permutací – tím získáme jednu z možností uspořádání tabulky. S pomocí správně zvolené kombinace funkcí MOD, jež vrací zbytek po dělení, a ZAOKR.DOLŮ, představující celočíselné dělení, postupně v každé buňce získáme jednotlivé cifry v pořadí vybrané permutace. Je samozřejmě vždy nutné dělit mocninou deseti, abychom získali správné číslo. Až budeme hotovi s následujícím krokem, můžeme všechny tyto buňky skrýt, nebo změnit barvu textu tak, aby odpovídala barvě pozadí, a eliminovat tak rušivý element v následném vizuálním dojmu. Chceme-li naopak zvýraznit, o co jde, je možné tabulku vhodně naformátovat a nadepsat. [2]

Do druhé připravené tabulky nyní můžeme s pomocí vnořených podmínek nastavit nakopírování řádků z tabulky původní v takovém pořadí, jaké jsme v předešlém kroku určili. Teď už bychom měli mít tabulku, která na hlavní diagonále neobsahuje žádné nuly, a může být využita k samotnému výpočtu jednotlivých iterací. Může nastat případ, že funkce z minulého kroku vrací nulu, neboť žádná permutace odpovídající požadavku neexistuje. To by ovšem znamenalo, že je matice singulární, a nemá smysl pro ni výpočet provádět. Jelikož však všechny kroky musíme projít tak jako tak, vrátíme se k této možnosti až později a tabulku náležitě ošetřit.

Pro implementaci samotných iterací vytvoříme tabulku námi zvolené délky. V tomto případě bude mít použitá tabulka sto řádků, jak již bylo naznačeno v algoritmizaci pro MS Excel, do nichž samozřejmě nepočítáme záhlaví. Tabulka bude mít čtyři sloupce, ne-

boť zde není zapotřebí sloupce absolutních členů. Každý sloupec bude nadepsán označením proměnné, jejíž hodnota je v něm aproximována. První řádek ponecháme volný pro zápis, do něj totiž uživatel zadá počáteční aproximaci. Pro testovací účely ji můžeme zvolit sami, v tomto příkladě využiji samé nuly. [2]

Od druhého řádku dále se bude opakovat stále tatáž sada vzorců vyjadřujících jednotlivé proměnné z matice, přesně jak je uvedeno v algoritmizace. Důležité je v tomto vzorci počítat s odkazy na druhou, správně seřazenou tabulku, a tyto odkazy opatřit znakem $ před jeho číselnou částí. Taktéž v souladu s předlohou v algoritmizaci budou tyto vzorce odkazovat na předešlou aproximaci, tedy předchozí řádek, jímž je v prvním případě počáteční aproximace, tyto odkazy se však budou s každým řádkem měnit, a tudíž je znakem $ nezafixujeme. Nyní již můžeme vzorci vytvořenými ve druhém řádku automaticky vyplnit zbytek tabulky. I zde doporučuji tabulku vhodně naformátovat.

Základní struktura metody, neboli výpočet samotný, je v této fázi hotov. Nyní je na řadě ověření podmínek a nastavení chování dokumentu podle toho, zda byly splněny či nikoliv. Stejně jako u eliminačních metod, i zde je nejprve zapotřebí ověřit, jestli je matice regulární, a tedy zda má smysl pokoušet se o výpočet. K tomu nám pomůže funkce DETERMINANT, aplikovaná na původní tabulku.

Je-li determinant nulový, matice je singulární a my můžeme nastavit podmíněný formát druhé tabulky matice tak, aby nezobrazovala hodnoty, tedy barvu písma nastavíme totožnou s barvou pozadí, ideálně nějaký světlejší odstín červené. Takto obarvíme i pozadí prvního řádku tabulky s výpočty, a u všech ostatních řádků nastavíme pozadí průsvitné a písmo bílé, abychom vyvolali zdání, že tabulka více řádků nemá. Stejným způsobem zbarvíme i buňku s determinantem, abychom tento pomocný výpočet skryli. Pro všechny buňky provádějící výpočty můžeme nastavit, aby v případě nulového determinantu zobrazovaly prázdný řetězec a vyhnout se tak chybovým hlášením.

Další podmínka, kterou je zapotřebí ověřit, je konvergence metody. Stejně jako u determinantu nám k tomu poslouží jediná buňka se vzorcem, který však v tomto případě musíme sestavit sami. Jak již bylo zmíněno, ověřovat spektrální poloměr by bylo neúměrně složité, a proto využijeme jiný postup. Pokud necháme proběhnout všech sto iterací, pak i kdyby na konci náhodou ještě nebyla dosažena požadovaná přesnost, hodnoty si sobě budou už velice blízké a rozdíly budou v rámci desetinných míst. Naopak při divergenci narůstají rozdíly do velmi vysokých rozměrů. Postačí nám tedy ověřit, že pro každý sloupec je absolutní hodnota rozdílu posledních dvou hodnot menší než 1. Pokud ano, buňka vrátí hodnotu 1, v opačném případě hodnotu 0.

Nakonec musíme ověřit, zda bylo v rámci omezeného počtu iterací dosaženo přesnosti zadané uživatelem. Pro zadání přesnosti vytvoříme okénko, například sloučením více buněk, nadepsané jako požadovaná přesnost, kterou uživatel zadá pomocí desetinného čísla, nejčastěji záporné mocniny deseti. Tuto podmínku bude nutné ověřit zvlášť v každém řádku iterační tabulky. Pro tento účel si vybereme dosud prázdný, nebo alespoň na pohled prázdný, sloupec, jehož šířku na konci zkrátíme na nulu, aby hodnoty v něm nekazily vizuální dojem.

Jelikož přesnost má smyl ověřovat až od druhé aproximace, první podmínku umístíme tam. Doporučuji podmínku umístit na stejný řádek jako je aproximace, kterou prověřuje.

Podmínka bude vždy pro každý ze sloupců porovnávat, zda je rozdíl současné a předešlé hodnoty menší nebo roven požadované přesnosti. Pokud alespoň pro jeden sloupec splněna není, funkce vrátí hodnotu -1. Pokud však ano, znamená to, že přesnosti dosaženo bylo, nicméně je ještě třeba ověřit, zda je to poprvé či nikoliv. Podmínka tedy zkontroluje, jestli funkce na předešlém řádku vrátila -1 nebo ne. Jestli ano, jedná se o první dostatečně přesnou iteraci, a funkce vrátí hodnotu 0, v opačném případě vrátí hodnotu 1.

Tabulku iterací je nyní třeba vhodně naformátovat za účelem zvýraznění prvního vyhovujícího výsledku a skrytí všech řádků za ním. Využijeme vestavěné funkce Excelu, podmíněné formátování. Tento nástroj nabízí možnost přidat až tři různé formáty, což je přesně tolik, kolik potřebujeme. Použité podmínku budou velice jednoduché. Po označení tabulky od třetího řádku dále vytvoříme prozatím dva formáty, odvolávající se na hodnoty podmínek z předešlého kroku.

Bude-li se hodnota této funkce rovnat 0, formát nastaví sytější barvu pozadí, tučné písmo a přidá ohraničení. V případě hodnoty 1, nebo nulového determinantu, či nenalezení vhodné permutace, bude pozadí nastaveno průhledné, ohraničení zrušeno, a barva písmo nastavena na bílou, čímž se dané řádky zneviditelní. Vždy ve vzorci odkazujícím na podmínku ověřovanou v každém řádku je třeba zafixovat označení sloupce znakem $. V případě odkazů na determinant a permutaci je nutné zafixovat i označení řádku. Po dokončení formátování tabulky označíme samotný poslední řádek a přidáme poslední, třetí podmínku. Ta v případě, že se hodnota pomocné funkce bude rovnat -1, přidá ohraničení a obarví pozadí na světle červené, značící neúspěch. Pro ostatní řádky hodnota -1 formátování nemění.

Na závěr vytvoříme stejné okénko jako pro zadání přesnosti. V tomto případě bude postupně ověřovat splnění všech podmínek a v závislosti na tom zobrazovat výsledek. Využity budou tedy vnořené podmínky. Jako první bude ověřena hodnota determinantu. Bude-li nulový, okénko uživatele informuje, že matice není regulární. Dále se uvěří konvergence. V případě jejího nedosažení okénko zobrazí, že metoda diverguje. A na závěr, pokud budou obě předešlé podmínky splněny, bude ověřeno dosažení přesnosti, tedy zda se hodnota pomocné funkce na posledním řádku iterační tabulky nerovná -1. Není-li tato podmínka splněna, okénko vypíše, že nebylo dosaženo požadované přesnosti. V opačném případě uživateli sdělí, že výsledek najde na konci tabulky. Můžeme také nastavit podmíněné formátovaní okénka, aby bylo v případě splnění všech podmínek nastaveno světle zelené pozadí, v případě nesplnění potom červené.

2.5.3 Implementace v C#

Na rozdíl od eliminačních metod vysvětlených v předešlých kapitolách bude implementace iteračních metod o něco složitější. Roli v tom hraje především, že v případě eliminačních metod bylo v případě nenulové hodnoty determinantu jasné, že metoda dojde správnému řešení. U metod iterační je však zapotřebí ověřit více podmínek. První z nich je správné seřazení matice, neboť vzhledem k povaze vzorců určených k výpočtu jednotlivých iterací

Obrázek 8: Rozhraní pro iterační metody

w	x	y	z	b
0	0	0	4	5
1	1	1	1	5
0	3	5	0	3
1	2	5	5	4

w	x	y	z	b
1	1	1	1	5
0	3	5	0	3
1	2	5	5	4
0	0	0	4	5

w	x	y	z
0	0	0	0
5,00	1,00	-0,60	1,25
3,35	2,00	-1,92	1,25
3,67	4,20	-2,86	1,25
2,41	5,77	-3,24	1,25
1,22	6,40	-3,26	1,25
0,60	6,43	-3,14	1,25
0,47	6,23	-3,04	1,25
0,55	6,06	-2,98	1,25
0,67	5,97	-2,97	1,25
0,75	5,96	-2,98	1,25
0,78	5,97	-2,99	1,25
0,77	5,99	-3,00	1,25
0,76	6,00	-3,00	1,25
0,75	**6,00**	**-3,00**	**1,25**

← zde zadejte počáteční aproximaci

Zadejte požadovanou přesnost:

0,01

Výsledek najdete na konci tabulky

je zapotřebí zajistit, aby se na hlavní diagonále nenacházela žádná nula, a pokud se tam vyskytuje, najít takové řazení, které bude předešlému požadavku odpovídat.

K tomuto účelu je třeba si vytvořit vlastní metodu, jež bude pracovat přímo se strukturami objektu Matice, a nebude tudíž seřazenou matici vracet. Namísto toho vrací hodnotu typu bool, která bude hlavní metodu informovat, zda bylo seřazení úspěšné či nikoliv. Metoda pro začátek ověří, zda je vůbec zapotřebí matici řadit. K tomu využije cyklus, který každý z prvků hlavní diagonály otestuje, jestli není roven nule. Pokud žádný takový nenajde, metoda končí a vrací true, neboť je matice již seřazená správně.

```
private bool Razeni(int n)
{
    bool diagonala = true;
    for (int i = 0; i < n; i++)
        if (nezname[i, i] == 0)
        {
            diagonala = false;
            break;
        }
    if (diagonala) return true;
```

Doteď byl kód velmi jednoduchý. Problém nastává v případě, že je nutné matici seřadit tak, aby na hlavní diagonále nebyla nula. Zatímco v MS Excel, kde jsme pracovali pouze

s maticí velikosti 4, nebyl problém projít všechny možné kombinace a zvolit takovou, která splňuje podmínky, v případě algoritmu pro matici obecné velikosti to tak snadné není. Algoritmus procházející všechny možné permutace je složitosti n! (n-faktoriál), což znamená, že s rostoucím n narůstá objem výpočtů neúměrně rychle a při dostatečně velkém čísle by takový program mohl běžet příliš dlouho. Musíme tedy najít jiný způsob.

V této práci byl k danému účelu zvolen algoritmus spadající pod tzv. heuristiku. Tento pojem slouží k označení algoritmů, které ve většině případů rychle poskytují dostatečně přesné řešení. Nelze se však spolehnout, že algoritmus najde řešení vždy. Stále je však možné navrhnout jej tak, aby se pravděpodobnost neúspěchu minimalizovala, jak bude ostatně vysvětleno níže.

Pro začátek je třeba si u každého řádku zjistit, na kterých místech se může vyskytovat, aniž by porušil podmínku, že na hlavní diagonále nesmí být nula. K uchování těchto informací použijeme strukturu dictionary, ukládající data na principu klíč-hodnota. Klíč bude v tomto případě představovat číslo značící současné pořadí řádku, a hodnotu list celočíselných hodnot, obsahující všechna pořadí, která přicházejí v úvahu. Všechna tato čísla se budou ukládat jako indexy, tudíž budou o jedno menší než skutečné pořadí, neboť indexování začíná od nuly. Seznam typu list volíme namísto pole proto, že bude v rámci kódu nutné do seznamu přidávat prvky a následně je mazat, což u pole není možné. Následující kód postupně projde všechny řádky matice a uloží do příslušné proměnné indexy všech nenulových členů – tedy takových, které by mohly být na hlavní diagonále.

```
Dictionary<int, List<int>> seznam = new
Dictionary<int, List<int>>();
for(int i = 0; i < n; i++)
{
    List<int> radek = new List<int>();
    for (int j = 0; j < n; j++)
        if (nezname[i, j] != 0)
            radek.Add(j);
    seznam.Add(i, radek);
}
```

Následuje nejobtížnější část. Zde je nutné zkonstruovat algoritmus tak, aby byla šance selhání co nejmenší. Základní princip pokrývá cyklus, který postupně prochází všechny řádky, pro každý z nich zvolí ze seznamu vhodné umístění, a následně toto umístění vymaže z ostatních seznamů, aby se nestalo, že bude dvěma řádkům přiřazeno stejné místo. Pokaždé je nejprve testována podmínka, že danému řádku zbývá alespoň jedno umístění, které přichází v úvahu. Pokud je však příslušný seznam prázdný, metoda se ukončí a vrátí false na znamení, že se matici nepodařilo seřadit.

Zásadní však je volba pořadí, jakým bude procházení probíhat. Je myslím na první pohled jasné, že ponechání výchozího pořadí by byl krok velmi nepraktický. O něco lepší je alternativa, kdy se seznam nejprve seřadí tak, aby na začátku byl řádek s nejnižším počtem nenulových členů, a na konci ten s nejvyšším. Nyní si již lze představit docela rozumný průběh programu. Tento postup však lze ještě vylepšit, a to tak, že bude na začátku každého

kroku seznam seřazen znovu, pro případ, že by se mezitím poměry změnily. Takto je již šance na selhání metody velmi malá.

```
int[] poradi = new int[n];
for(int i = 0; i < n; i++)
{
    var radek = seznam.OrderBy(k => k.Value.Count).First();
    if (radek.Value.Count == 0)
        return false;
    poradi[radek.Key] = radek.Value[0];
    seznam.Remove(radek.Key);
    foreach(var x in seznam)
        seznam[x.Key].Remove(radek.Value[0]);
}
```

Na závěr zbývá matici jen seřadit. Za tímto účelem si nejprve vytvoříme struktury velikostí a typem odpovídající matici levých stran soustavy rovnic a sloupci pravých stran. Do těchto struktur budeme v závislosti na pořadí přiřazeném v předešlém kroku přiřazovat řádky ze struktur původních. Na závěr původní struktury těmito novými přepíšeme.

```
float[,] matice = new float[n, n];
float[] konst = new float[n];
for(int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
        matice[poradi[i], j] = nezname[i, j];
    konst[poradi[i]] = konstanty[i];
}
nezname = matice;
konstanty = konst;
return true;
}
```

Jako další na řadě by mělo být ověření, zda bude metoda pro danou matici konvergovat. Tato část je zde mnohem důležitější než byla v Excelu, neboť kdybychom zde nechali probíhat cyklus podmíněný dosažením přesnosti, aniž bychom nejprve ověřili podmínku konvergence, mohlo by snadno dojít k případu, že by se program zacyklil a nikdy neskončil.. Stejně jako v případě MS Excel, i tady si budeme muset vystačit s numerickým přístupem, neboť výpočet skrze spektrální poloměr by byl neúměrně složitý. I zde bude ověření této podmínky provedeno až v rámci iterace samotné.

Nyní před sebou máme hlavní metodu, jež stejně jako u eliminačních metod vrací jednorozměrné pole s výsledky. V čem se však liší jsou vstupní parametry. Zatímco u eliminačních metod jsme se obešli bez nich, zde budeme potřebovat dva vstupy od uživatele. Prvním z nich je požadovaná přesnost, zadaná ve formě desetinného čísla, a tedy reprezentovaná typem float, a proměnná typu bool, která nám říká, zda se jedná o Jacobiho metodu (stav true) nebo

Gauss-Siedelovu (stav false). Tyto dvě metody jsou totiž natolik podobné, že krom jediného řádku bude kód pro jejich výpočet naprosto identický.

Zde v těle hlavní metody se vyskytují dvě místa, z nichž je možné, aby metoda místo výsledků vrátila hodnotu null. První z nich najdeme hned na začátku, kdy se null vrací v případě že se matici nepodařilo seřadit tak, aby odpovídala požadavku na nenulové prvky na hlavní diagonále.

```
public float[] Vypocet(float presnost, bool jacobi)
{
    int n = konstanty.Length;
    if (!Razeni(n)) return null
```

Výpočet samotný zde bude rozdělen do dvou fází. První, neboli kontrolní, slouží k ověření konvergence metody a je omezen na patnáct opakování cyklu. Nejprve se definuje pole počáteční aproximace, do nějž jsme v tomto případě nastavili samé nuly, a kontrolní pole stejné délky. Také zde nastavíme proměnnou typu bool, jež bude ukazatelem, zda bylo dosaženo požadované přesnosti. Následně se provede patnáct iterací za pomoci volání funkce, která jednotlivou iteraci vypočítá a vrátí novou aproximaci a hodnotu bool. Jak je vidět, funkce může být nastavena i tak, že vrací dvě a více hodnot. Na závěr se provede další aproximace, jejíž výsledek se uloží do kontrolního pole.

```
    float[] kontrol, apx =
    Enumerable.Repeat(0.0f, n).ToArray();
    bool hotovo;
    for (int i = 0; i < 15; i++)
        (hotovo, apx) = Iterace(apx, n, presnost, jacobi);
    (hotovo, kontrol) = Iterace(apx, n, presnost, jacobi);
```

V dalším kroku se porovnají hodnoty poslední a předposlední aproximace, z nichž jedna je uložena v kontrolním poli. Pro každé dva prvky na stejném indexu musí platit, že je absolutní hodnota jejich rozdílu menší než 1. Pokud podmínka splněna není, metoda končí a vrací null. Je-li podmínka splněna pro všechny indexy, máme již v podstatě vyhráno.

```
    for (int i = 0; i < n; i++)
        if (Math.Abs(kontrol[i] - apx[i]) > 1)
            return null;
```

Následuje cyklus typu while. Ten se od for cyklu liší tím, že nemá pevně stanovený počet opakování. Naopak je jeho další provedení opatřeno podmínkou, že proměnná informující, zda bylo dosaženo přesnosti, nabývá hodnoty false, a je tudíž zapotřebí dále iterovat.

```
    while (hotovo == false)
        (hotovo, apx) = Iterace(apx, n, presnost, jacobi);
    return apx;
}
```

Metoda provádějící jednu iteraci je specifická tím, že namísto jediné proměnné vrací dvě. V C# je toho možné docílit velice snadno, a to uvedení obou proměnných

do kulatých závorek, přičemž je oddělíme čárkou. Stejný postup je třeba aplikovat i u přiřazování výsledků metody do proměnných v metodě hlavní. Jako vstupní parametry tu máme předchozí iteraci, velikost soustavy (tedy i pole s iterací), požadovanou přesnost a proměnnou informující program, zda jde o Jacobiho či Gauss-Siedelovu metodu. Právě zde bude totiž na tomto rozdílu záležet.

Výpočet nové iterace je prostý. Nejprve se vytvoří nové pole pro novou iteraci, do nějž se nakopíruje ta původní. Dále bude následovat cyklus, který provede pro každý prvek v iteraci tentýž výpočet. Pro výpočet hodnoty do nové iterace je opět volána jiná metoda. A právě tady se jako vstupní parametr do výpočetní metody pošle původní aproximace, pokud se jedná o Jacobiho metodu, případně nekompletní nová aproximace, pokud se jedná o Gauss-Siedelovu metodu. Když je nová hodnota přiřazena na odpovídající místo v poli, provede se kontrola, zda je rozdíl této hodnoty a té z předešlé aproximace menší než požadovaná přesnost. Pokud ne, nastaví se pomocná proměnná typu bool na false.

```
private (bool, float[]) Iterace(float[] apx1, int n,
float presnost, bool jacobi)
{
    bool hotovo = true;
    float[] apx2 = new float[n];
    apx1.CopyTo(apx2, 0);
    for (int i = 0; i < n; i++)
    {
        if (jacobi) apx2[i] = Hodnota(apx1, i, n);
        else apx2[i] = Hodnota(apx2, i, n);
        if (Math.Abs(apx1[i] - apx2[i]) > presnost)
            hotovo = false;
    }
    return (hotovo, apx2);
}
```

Poslední metodou, kterou si zde popíšeme, je výpočet jednotlivé hodnoty v dané aproximaci. Zatímco v MS Excel jsme tyto vzorce psali zvlášť pro každou proměnnou, zde je nutné výpočet zobecnit, neboť pracujeme s obecnou soustavou rovnic. Když se podíváme na vzorce popsané v algoritmizaci pro MS Excel výše, lze vypozorovat určité opakující se schéma. Vždy se nejprve od absolutního členu odečtou prvky předešlé aproximace vynásobené příslušným koeficientem, ovšem s výjimkou prvku, který právě počítám, a k němuž příslušným koeficientem získaný výsledek vydělíme.

Tento postup se dá snadno zobecnit tak, že se do pomocné proměnné uloží absolutní člen daného řádku matice soustavy a následně se v cyklu postupně odečítají součiny členů s koeficienty z této matice. Přičemž než k odečtu dojde, ověří program podmínku, zda se index prvku nerovná indexu, jež byl metodě předán jako parametr. Pokud ano, tento krok cyklus přeskočí. Po ukončení cyklu se výsledek vydělí koeficientem z matice levých stran soustavy.

```
private float Hodnota(float[] promenne, int i, int n)
{
    float konst = konstanty[i];
    for (int j = 0; j < n; j++)
        if(i != j)
            konst -= promenne[j] * nezname[i, j];
    konst = konst / nezname[i, i];
    return konst;
}
```

Závěr

Tato diplomová práce měla za cíl seznámit čtenáře s různými postupy a možnostmi použití numerických metod za účelem řešení soustavy lineárních rovnic, a to jak s pomocí kancelářského softwaru MS Excel, tak s využitím programovacího jazyka C#.

Splnění zadaného cíle bylo dosaženo ve čtyřech podkapitolách zaměřených na konkrétní numerické metody. Z eliminačních metod byly zvoleny Gaussova a Jordanova, zatímco mezi využité iterační metody se řadí Jacobiho a Gauss-Siedelova metoda. Postup implementace každé ze zmíněných metod byl pro obě aplikovaná prostředí podrobně objasněn, přičemž nechybělo ani vysvětlení, z jakého důvodu se postupovalo právě takto, a v případě MS Excel byl postup doplněn o snahu optimalizovat vizuální stránku věci.

Pro každou z metod bylo zvolené řešení, které co nejlépe odpovídalo požadavkům na přehlednost a jeho aplikace byla smysluplná. Nalezly se ovšem případy, v nichž hledání elegantního řešení představovalo skutečnou výzvu, nicméně si troufám říci že se tato snaha nakonec setkala s úspěchem. Příkladem budiž seřazení matice pro iterační metody tak, aby se na hlavní diagonále nevyskytovala nula.

Čtenáři zběhlí v oboru informatiky nejspíše ocení implementaci v prostředí známého programovacího jazyka C# včetně využití frameworku Xamarin Forms, který dodává výsledné aplikaci grafickou podobu a nejen že tak usnadňuje přehlednou práci, nicméně také využívá rozličných metod pro kontrolu správnosti zadání, čímž předchází nechtěným pádům aplikace při zadání chybného či neočekávaného vstupu.

Jelikož se jedná o jazyk z rodiny C, pro většinu programátorů s těmito jazyky obeznámených by nemělo představovat problém kód adaptovat pro jazyk příbuzný. V této snaze by mělo být nápomocno i slovní vysvětlení postupu.

V praxi je možné tuto práci využít jako podklad k přípravě výuky numerických metod řešení soustav lineárních rovnic, rovněž i pro případné samostudium, pokud by někdo projevil individuální zájem, či se z jakéhokoliv důvodu nemohl účastnit prezenční výuky v budově školy.

Případné rozšíření práce by mohlo pokrýt výpočty kořenů nelineárních rovnic. Některé aspekty tohoto problému by nejspíše představovaly značnou výzvu pro obě prostředí využitá v původní diplomové práci.

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] EL GHARRED, Alex. *MS Excel jako nastroj řešení úloh numerických metod*. Hradec Králové, 2021. Bakalářská práce. UHK. Vedoucí práce Hubálovský Štěpán, doc. RNDr. Ph.D.

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

[4] HASÍK, Karel. *Numerické metody*.

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

[6] 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/

[7] 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

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

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

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

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

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

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

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

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

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

[17] 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/

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

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

[20] MARVAN, Michal. *Učební texty k přednášce ALGEBRA I, zimní semestr 2000/2001*. Matematický ústav Slezské univerzity v Opavě, 2000.

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

Seznam obrázků

Seznam příloh

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Alex El Gharred**

Studium: I2100057

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Porovnání možností a účinnosti prostředků MS Excel a C# při řešení soustav lineárních rovnic**

Název diplomové práce AJ: Comparison of possibilities and efficiency of MS Excel and C # in solving systems of linear equations

Cíl, metody, literatura, předpoklady:

Rámcová osnova:

- Úvod do problematiky numerických metod (co to je, chyby při výpočtech, nástroje)

- Definice pojmového aparátu

- Vysvětlení matematického principu jednotlivých metod řešení soustav lineárních rovnic

- Algoritmizace zmíněných metod

- Implementace v MS Excel

- Implementace v C#

- Porovnání výhod a nevýhod obou implementací

Vybrané prostředky: MS Excel, C#

Vybrané algoritmy:

Eliminační (Gaussova eliminace, Jordanova eliminace)

Iterační (Jacobiho metoda, Gauss-Siedelova metoda)

Prostředky a algoritmy porovnány z hlediska komfortu, složitosti a času na řešení testových příkladů.

EL GHARRED, Alex. MS Excel jako nástroj řešení úloh numerických metod. Hradec Králové, 2021. Bakalářská práce. Univerzita Hradec Králové. Vedoucí práce Hubálovský Štěpán, doc. RNDr. Ph.D.

- TROJOVSKÁ, Eva. Základy numerické matematiky.

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

- ČÁPKA, David. Základní konstrukce jazyka C# .NET: Online kurz. ITnetwork [online]. [cit. 2021-

03-13]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady

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

další lit. bude specifikována

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Datum zadání závěrečné práce: 26.1.2021